Fully dynamic algorithm for chordal graphs with $O(1)$ query-time and $O(n^2)$ update-time

Mauro Mezzini

Department of Computer Science.
Sapienza University of Rome, Italy
http://www.di.uniroma1.it/

May 2012

Abstract

We propose dynamic algorithms and data structures for chordal graphs supporting the following operation: determine if an edge can be added or removed from the graph while preserving the chordality in $O(1)$ time. We show that the complexity of the algorithms for updating the data structures when an edge is actually inserted or deleted is $O(n^2)$ where n is the number of vertices of the graph.

Key words: Chordal graphs; Dynamic algorithms, Minimal Triangulations

1. Introduction

A *cycle* of length k, $k \geq 3$, of a graph, is a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $v_i v_{i+1}$ is an edge of the graph, $i = 0, \ldots, k-1$ and all vertices are distinct except v_0 and v_k which do coincide. A *chord* in a cycle is an edge of the graph joining two non-consecutive vertices of the cycle. A graph is *chordal* (or *triangulated*) if every cycle of length greater than three has a chord (see [7] for a tutorial on chordal graphs). The class of chordal graphs is an important and well-studied class of graphs [7, 24, 11]. Chordal graphs arises in many practical and relevant fields such as computing the solutions of large sparse symmetric systems of equations [22, 3, 4, 6, 14, 15, 7, 17, 16, 26, 27, 19, 13], in database systems [12, 8, 26, 2], artificial intelligence [25, 9, 20, 1] and biology [5].

For example chordal graphs are used in the field of statistics and artificial intelligence [20, 10, 25, 9, 1]. In this context one wants to estimate an n-dimensional discrete probability distribution from a finite set of given marginals in order to use a small amount of machine memory. To this aim, one models joint probability distributions of n discrete random variables by *Markov networks* (also called *graphical models*). Such models use undirected graphs to capture conditional dependencies among subsets of the n random variables involved. Particular Markov networks, called *decomposable Markov networks* (DMNs), use chordal graphs. DMNs enjoy a number of desirable properties, one of which is that the approximate distribution has a simple 'product form' [21]. Therefore, given a joint probability distribution, one is interested in finding an approximation of it which is a decomposable Markov network. A possible approach, called *backward selection*, to solve this problem is the following: starting from the complete graph on n vertices (no assumption of independence among the random variables of the joint distribution is made) one recursively removes an edge from the graph while chordality is preserved. The opposite approach, called *forward selection*, does exactly the inverse procedure, that is, it starts from an empty graph and adds edges to the graph while preserving the chordality.

Note that in such applications one wants to find as quickly as possible the sets A and R of edges eligible, respectively, for addition and deletion to the graph G, without destroying chordality. Then one chooses, at each step, an edge in the set A or an edge in the set R that satisfies some specified property. For example, in the above application, one chooses at each step the edge to add or delete, that minimizes the *information* *divergence* [18]. Once the edge is added or deleted the sets A and R should be updated to reflect the changes in the graph.

So one wants to set up a dynamic algorithm that, as edges are added or removed from the graph, updates the data structure needed to find the sets of edges which are eligible for addition or deletion from the graph while preserving the chordality.

A fully dynamic algorithm for chordal graph has been developed [17] supporting the following operations:

1. Delete-Query(uv): return "yes" if uv can be deleted from G while maintaining the chordality and "no" otherwise.
2. Insert-Query(uv): return "yes" if uv can be added to G while maintaining the chordality and "no" otherwise.
3. Delete(uv): update the data structures needed to perform operations (1) and (2) when an edge uv is deleted from G.
4. Insert(uv): update the data structures needed to perform operations (1) and (2) when an edge uv is added to G.

In the first implementation proposed in [17], all the above operations have time complexity of $O(n)$, where n is the number of vertices of the graph. In a second implementation, the operation Insert-Query has $O(log^2 n)$ time complexity and operation Insert has $O(n)$ time complexity while the operation Delete-Query has $O(n)$ time complexity and the operation Delete has $O(n\, log\, n)$ time complexity.

In [9] a dynamic algorithm is proposed to insert edges in a chordal graph while preserving chordality. In this algorithm the operation Insert-Query has $O(1)$ time complexity and the operation Insert has $O(n^2)$ time complexity. In [23] a dynamic algorithm is proposed, that beginning from a complete graph, removes edges while preserving chordality. In this algorithm operation Delete-Query has $O(1)$ time complexity and the operation Delete has $O(n^2)$ time complexity.

Here we propose a fully dynamic algorithm that runs in $O(1)$ for operations Delete-Query and Insert-Query and requires $O(n^2)$ time to perform operations Delete and Insert. This compares better with respect to the first implementation of the algorithm of [17] since this requires $O(nm)$ to find an edge eligible for deletion, where m is the number of edges of the graph or $O(n\overline{m})$ to find an edge eligible for insertion where $\overline{m}$ is the number of edges of the graph's complement. Therefore if one wants to add or remove k edges, the algorithm requires $O(kn^3)$ time. With our algorithm, on the other hand, it requires $O(kn^2)$ to perform a sequence of k inserts or deletes.

We also show that our algorithm can be used to find a *minimal triangulation* of a graph. A *triangulation* of a graph G is a chordal graph G^+, obtained from G by adding a set F of edges. The edges in F are called *fill* edges. A triangulation is *minimal* when F is a minimal set, with respect to set inclusion, of edges that added to G make it a chordal graph. Let G^+ be a triangulation of G, F the set of fill edges and m the number of edges of the input graph. Then the minimal triangulation algorithm we propose, has $O(n|F|+|F|^2+m)$ time complexity, which is comparable to existing minimal triangulation techniques [6, 23].

The work is organized as follows. In Section 2 we give some definitions and preliminary results. In Section 3 we recall the fully dynamic algorithm proposed in [17]. In Section 4 we discuss the algorithms and data structures for supporting the Delete-query operation and in Section 5 we discuss the algorithms and data structures for supporting the Insert-query operation. Finally in Section 6 we give the algorithms Insert and Delete that update the data structures to support the Insert-query and Delete-query respectively.

2. Definitions and preliminaries

Let $G = (V(G), E(G))$ be a graph where $V(G)$ is the *vertex set* and $E(G)$ is the *edge set* of G; furthermore $n = |V(G)|$ and $m = |E(G)|$. A set $\{r, s\}$ of two vertices will be called an edge and will be denoted as rs. Let $\mathcal{E}(G) = \{rs : r \in V(G), s \in V(G)\}$. Denote by $\overline{E(G)} = \mathcal{E}(G) - E(G)$ and by $\overline{m} = |\overline{E(G)}|$. When it is not clear from the context we will explicitly indicate if an element of $\mathcal{E}(G)$ belongs to $E(G)$ or to $\overline{E(G)}$.

Two vertices are *adjacent* if they are endpoints of an edge of G. The *neighborhood* of a vertex u in G is denoted by $N_G(u) = \{v : uv \in E(G)\}$. Given two vertices u and v the *common neighborhood* of uv, denoted by $CN_G(uv)$, is the set $N_G(u) \cap N_G(v)$. A *clique* of G is a set of pairwise adjacent vertices. A clique is *maximal* if it is not properly contained in another clique.

Let S be a subset of $V(G)$; the subgraph of G *induced* by S, denoted by $G(S)$, is the graph with vertex set S and edge set $\{uv \in E(G) : (u \in S) \wedge (v \in S)\}$. Let S be a non-empty set of vertices; then by $G - S$ we denote the subgraph of G induced by $V(G) - S$. If $S = \{v\}$ is a single vertex we write $G - v$ to denote $G - \{v\}$. Let $uv \in E(G)$; then by $G - uv$, we denote the graph with vertex set $V(G)$ and edge set $E(G) - uv$. Analogously, if $uv \in \overline{E(G)}$, we denote by $G + uv$, the graph with vertex set $V(G)$ and edge set $E(G) + uv$.

The *contraction* of an edge uv of G gives as a result a graph G' which is obtained from G by adding a new vertex x, replacing the edge wu with wx for all $w \in N_G(u) - v$, replacing the edge zv with zx for all $z \in N_G(v) - u$ and deleting the vertices u and v and the edge uv.

Let G be a connected graph; then a subset of vertices S is a *separator* of G if $G - S$ has two or more connected components. If two vertices u and v are in the same connected component of G and in two different connected components of $G - S$, then S is said to be a *uv-separator*. A *minimal uv-separator* S is an uv-separator such that no proper subset of S separates u and v in two connected components.

Let G be a chordal graph and $uv \in \overline{E(G)}$; then we say that uv is an *attachable* edge if $G + uv$ is a chordal graph. Similarly an edge $uv \in E(G)$ is *removable* if $G - uv$ is a chordal graph.

A *triangulation* of a graph G is a chordal graph G^+ such that $V(G) = V(G^+)$ and $E(G) \subseteq E(G^+)$. The set $F = E(G^+) - E(G)$ is called the set of *fill* edges.

A triangulation of a graph G is *minimal* when no proper subset of F if added to G, makes it a chordal graph.

Lemma 1 ([24]). *Let G be a chordal graph. An edge $uv \in E(G)$ is removable from G if it is not the unique chord of any 4-cycle of G.*

By Lemma 1 it follows

Lemma 2 ([17, 23]). *Let G be a chordal graph. Then $uv \in E(G)$ is removable if and only if $CN_G(uv)$ either is empty or is a clique of G.*

By Lemma 1 and Lemma 2, we have the following

Theorem 1 ([17, 9]). *Let G be a chordal graph. An edge $uv \in E(G)$ is removable if and only if it belongs to exactly one maximal clique of G.*

Furthermore we need the following

Lemma 3. *Let G be a chordal graph and $uv \in E(G)$. Then u and v are disconnected in $G - uv - CN_G(uv)$.*

Proof: Suppose, by contradiction, that u and v are connected in $G - uv - CN_G(uv)$. Let $P = (u, x_1, \ldots, x_k, v)$ be a chordless path connecting u and v in $G - uv - CN_G(uv)$; it follows that, $x_1 \notin CN_G(uv)$. Therefore, $k > 1$ and $P + uv$ should be a chordless cycle of length greater than three in G (contradiction). □

Theorem 2 ([17, 9]). *Let G be a chordal graph. An edge $uv \in \overline{E(G)}$ is attachable if and only if either u and v belong to different connected components of G or $CN_G(uv)$ is an uv-separator of G*

3. Related work

We briefly recall in this section the algorithms and data structures proposed by Ibarra [17] for supporting the Insert-query, Delete-query, Insert and Delete operations. We will use these data structures in our algorithm. We refer the reader for all the details to [17].

Given a chordal graph G denote by $\mathcal{K}_G$ the set of maximal cliques of G. A *clique tree* T of G is a tree structure with vertex set $\mathcal{K}_G$ that has the *induced subtree property*: given any vertex v of G, the subtree induced by the cliques containing v is a subtree of T. This property is equivalent to the *clique intersection property* : given any two cliques K_x and K_y then $K_x \cap K_y$ is contained in every clique in the path connecting K_x and K_y in T.

It is well-known [7] that a graph is chordal if and only if it has a clique tree. Another important and well-known property of a clique tree is that given an edge K_xK_y of T then $K_x \cap K_y$ is a minimal uv-separator for any $u \in K_x - K_y$ and $v \in K_y - K_x$.

Next we describe the algorithms to update a clique tree T of a chordal graph G when an edge is added or deleted from G. In both algorithms each edge K_xK_y of T is labeled by the weight $w(K_xK_y) = |K_x \cap K_y|$. We will later discuss the operations to determine whether an edge is removable or attachable.

The algorithm AddEdge-CliqueTree takes as input an attachable edge $uv \in \overline{E(G)}$ and a clique tree T. The algorithm is based on the following

Theorem 3 ([17]). *Let G be a chordal graph. An edge $uv \in \overline{E(G)}$ is attachable if and only if there exist a clique tree T of G and an edge K_xK_y of T such that $u \in K_x$ and $v \in K_y$.*

In order to find, if it exists, a clique tree that satisfies the condition of Theorem 3, one can use the following

Theorem 4 ([17]). *Let G be a chordal graph, T a clique tree of G and $uv \in \overline{E(G)}$. Let K_x and K_y be the closest vertices of T containing respectively u and v and let $P = (K_{i_0}, K_{i_1}, \ldots, K_{i_h})$ be the path in T connecting $K_x = K_{i_0}$ and $K_y = K_{i_h}$. Then uv is attachable if and only if the minimum of $w(K_{i_j}K_{i_{j+1}})$ for $j = 0, \ldots, h-1$, is equal to $|K_x \cap K_y|$.*

The algorithm AddEdge-CliqueTree (see Fig. 1) therefore, works as follows. It finds the closest vertices K_x and K_y of T, containing respectively u and v. Then it constructs a new clique tree T', by first removing the minimum weight edge of the path in T connecting K_x and K_y. Then, T' is modified by adding a new clique $K_z = (K_x \cap K_y) \cup \{u, v\}$ and two new edges K_xK_z and K_yK_z. It may happen that $K_x \subset K_z$ or $K_y \subset K_z$ or both. In the first case the edge K_xK_z is contracted and K_x is replaced by K_z. Analogously if $K_y \subset K_z$ the edge K_zK_y is contracted and K_y is replaced by K_z.

The algorithm RemoveEdge-CliqueTree (see Fig. 1) takes as input a removable edge $uv \in E(G)$ and a clique tree T of G. If K_x is the clique of G containing uv then it replaces K_x with two new cliques $K_x^u = K_x - \{v\}$ and $K_x^v = K_x - \{u\}$, joined by the edge $K_x^uK_x^v$. Next the following sets:

$$
\begin{aligned}
N_u &= \{K_y : (u \in K_y) \wedge (K_y \in N_T(K_x))\} \\
N_v &= \{K_z : (v \in K_z) \wedge (K_z \in N_T(K_x))\} \\
N_w &= \{K_w : (\{v, u\} \cap K_w = \emptyset) \wedge (K_w \in N_T(K_x))\}
\end{aligned}
$$

are constructed. Then for all $K_y \in N_u$ the edge K_yK_x is replaced with the edge $K_yK_x^u$ and for all $K_z \in N_v$ the edge K_zK_x is replaced with the edge $K_zK_x^v$. Finally for all $K_w \in N_w$ the edge K_wK_x is replaced with the edge $K_wK_x^u$ or the edge $K_wK_x^v$, choosing arbitrarily. It may happen that there exists a $K_y \in N_u$ such that $K_x^u \subset K_y$ or that there exists a $K_z \in N_v$ such that $K_x^v \subset K_z$ or both. In the first case the algorithm chooses arbitrarily one $K_y \in N_u$ such that $K_x^u \subset K_y$, contracts the edge $K_yK_x^u$ and replaces K_x^u with K_y. Analogously, in the second case the algorithm chooses arbitrarily one $K_z \in N_v$ such that $K_x^v \subset K_z$, contracts the edge $K_zK_x^v$ and replaces K_x^v with K_z.

Note that the changes in the clique tree made by algorithms AddEdge-CliqueTree and RemoveEdge-CliqueTree are limited to few vertices of the tree. More precisely we can state the following two remarks which will be useful in the subsequent sections.

Remark 1 ([17]). *Let $uv \in \overline{E(G)}$ be an attachable edge and let T be a clique tree of G such that K_xK_y is an edge of T and $u \in K_x$ and $v \in K_y$. Then all maximal cliques of G different from K_x and K_y are maximal cliques of $G + uv$.*

Algorithm AddEdge-CliqueTree
Input: a chordal graph G, an attachable edge uv and a clique tree T of G.
Output: the chordal graph $G + uv$ and a clique tree T' of $G + uv$.
begin
  the K_x and K_y be closest vertices of T containing respectively u and v;
  remove from T the minimum weight edge on the path connecting K_x and K_y in T;
  let $K_z \leftarrow (K_x \cap K_y) \cup \{u, v\}$;
  add to T the edges $K_x K_z$ and $K_y K_z$;
  if $K_x \subset K_z$ **then** contract $K_x K_z$ and replace K_x with K_z ;
  if $K_y \subset K_z$ **then** contract $K_y K_z$ and replace K_y with K_z;
end

Algorithm RemoveEdge-CliqueTree
Input: a chordal graph G, a removable edge $uv \in E(G)$ and a clique tree T of G.
Output: the chordal graph $G - uv$ and a clique tree T' of $G - uv$.
begin
  let K_x be the clique of T containing uv;
  let $N_u \leftarrow \{K_y : (u \in K_y) \wedge (K_y \in N_T(K_x))\}$;
  let $N_v \leftarrow \{K_z : (v \in K_z) \wedge (K_z \in N_T(K_x))\}$;
  let $N_w \leftarrow \{K_w : (\{v, u\} \cap K_w = \emptyset) \wedge (K_w \in N_T(K_x))\}$;
  let $K_x^u \leftarrow K_x - \{v\}$ and $K_x^v \leftarrow K_x - \{u\}$;
  Replace K_x with vertices K_x^u and K_x^v joined by the edge $K_x^u K_x^v$;
  for all $K_y \in N_u$ **do** replace $K_y K_x$ with $K_y K_x^u$;
  for all $K_z \in N_v$ **do** replace $K_z K_x$ with $K_z K_x^v$;
  for all $K_w \in N_w$ **do**
    replace $K_w K_x$ with $K_w K_x^v$ or $K_w K_x^u$, choosing arbitrarily;
  if $\exists K_y \in N_u$ such that $K_x^u \subset K_y$ **then**
    contract $K_y K_x^u$ and replace K_x^u with K_y;
  if $\exists K_z \in N_v$ such that $K_x^v \subset K_z$ **then**
    contract $K_z K_x^v$ and replace K_x^v with K_z;
end

Figure 1: The algorithms for updating the clique tree when an attachable edge is inserted or a removable edge is deleted

Remark 2 ([17]). *Let $uv \in E(G)$ be a removable edge and K_x the maximal clique of G containing uv. Then all maximal cliques of G different from K_x are maximal cliques of $G - uv$.*

Since a chordal graph has at most n maximal cliques, by Theorem 1, in order to check if an edge is removable one simply counts the number of maximal cliques containing it. This can be done in $O(n)$ time.

By Theorem 4, in order to determine if an edge $uv \in \overline{E(G)}$ is attachable we must find the closest vertices K_x and K_y of T containing respectively u and v and if $P = (K_{i_0}, K_{i_1}, \ldots, K_{i_h})$ is the path in T connecting $K_x = K_{i_0}$ and $K_y = K_{i_h}$, we must check that the minimum of $w(K_{i_j} K_{i_{j+1}})$ for $j = 0, \ldots, h-1$, is equal to $|K_x \cap K_y|$. Since it takes $O(n)$ time to determine $|K_x \cap K_y|$, this can be done in $O(n)$ time.

The algorithm AddEdge-CliqueTree requires $O(n)$ time. In fact as said above finding the closest vertices K_x and K_y of T containing respectively u and v and determining $|K_x \cap K_y|$ can be done in $O(n)$. In order to determine if $K_x \subset K_z$ we must check if $K_z - \{v\} = K_x$. This happens if and only if $|K_x \cap K_y| + 1 = |K_x|$. Similarly in order to determine if $K_y \subset K_z$ we must check if $|K_x \cap K_y| + 1 = |K_y|$. Using the clique tree edges' weight, this can be done in $O(n)$ time.

The algorithm RemoveEdge-CliqueTree also requires $O(n)$ time. In fact finding the sets N_u, N_v and N_w requires $O(n)$ time. By the clique intersection property, there exists a $K_y \in N_u$ such that $K_x^u \subset K_y$ if and only if $K_x^u = K_x \cap K_y$ and this happens if and only if $|K_x^u| = |K_x \cap K_y|$. Analogously there exists a $K_z \in N_v$ such that $K_x^v \subset K_z$ if and only if $|K_x^v| = |K_x \cap K_z|$. Therefore using the clique tree edges' weight, one can determine if there exists a K_y (resp. a K_z) such that $K_x^u \subset K_y$ (resp. $K_x^v \subset K_z$) in $O(n)$ time. Then by what said above we can state the following

Theorem 5 ([17]). *The time complexity of each of the following operations is $O(n)$.*

1. Delete-Query *(uv)*
2. Insert-Query *(uv)*
3. Delete *(uv)*
4. Insert *(uv)*

4. Algorithms and data structures for supporting DELETE-QUERY

We show in this section data structures for supporting a delete-query in $O(1)$ time. We also show the algorithms for updating these data structures when the insert or delete operation is executed and analyze their complexity.

We will use two data structures: the clique tree and a variable CC containing for every edge $uv \in E(G)$ the number of maximal cliques of G containing uv. By Theorem 1, an edge $uv \in E(G)$ is removable if and only if the number of maximal cliques of G containing it is equal to one. Therefore, in order to determine if an edge rs is removable, we simply check if $CC(rs)$ is or is not equal to one, which requires $O(1)$ time. Given a chordal graph G the variable CC may be initialized by determining for each edge of G the number of maximal cliques of G containing it. Since a graph G has at most n maximal cliques it requires $O(nm)$ to initialize CC.

When we insert or remove edges from the graph we will use the clique tree to update the value of CC in time bounded by $O(n+m)$, as described in the following.

The algorithm RemoveEdge1 (see Fig 2) takes as input a graph G, a removable edge $uv \in E(G)$, a clique tree T of G and the value of CC for the edges of G. It computes the new value of CC for the edges of $G - uv$ in the following manner.

Let K_x be the clique of G containing uv. For each edge with both endpoints in $K_x^u \cap K_x^v$, the value of CC is increased by one. If K_x^u is included in a clique K_y, then for each edge with both endpoints in K_x^u, the value of CC is decreased by one. Similarly if K_x^v is included in a clique K_z then for each edge with both endpoints in K_x^v, the value of CC is decreased by one.

Algorithm RemoveEdge1
Input: a chordal graph G, a clique tree T of G, a removable edge $uv \in E(G)$ and the value $CC(rs)$ for all rs of G.
Output: the new values of $CC(rs)$ for all rs of $G - uv$.
begin
  let K_x be the clique of T containing both u and v;
  let $K_x^u \leftarrow K_x - \{v\}$ and $K_x^v \leftarrow K_x - \{u\}$;
  for all $\{r, s\} \subseteq K_x^u \cap K_x^v$ **do** $CC(rs) \leftarrow CC(rs) + 1$;
  if $\exists K_y$ such that $K_x^u \subset K_y$ **then**
    for all $\{r, s\} \subseteq K_x$ such that $v \notin \{r, s\}$ **do** $CC(rs) \leftarrow CC(rs) - 1$;
  if $\exists K_z$ such that $K_x^v \subset K_z$ **then**
    for all $\{r, s\} \subseteq K_x$ such that $u \notin \{r, s\}$ **do** $CC(rs) \leftarrow CC(rs) - 1$;
end

Algorithm AddEdge1
Input: a chordal graph G, a clique tree T of G, an attachable edge $uv \in \overline{E(G)}$ and the value of $CC(rs)$ for all rs of G.
Output: the new values of $CC(rs)$ for all rs of $G + uv$.
begin
  Let K_x and K_y be closest vertices of T containing respectively u and v;
  let $K_z \leftarrow (K_x \cap K_y) \cup \{u, v\}$;
  for all $\{r, s\} \subseteq K_z$ **do** $CC(rs) \leftarrow CC(rs) + 1$;
  if $K_x \subset K_z$ **then**
    for all $\{r, s\} \subseteq K_z$ such that $v \notin \{r, s\}$ **do** $CC(rs) \leftarrow CC(rs) - 1$;
  if $K_y \subset K_z$ **then**
    for all $\{r, s\} \subseteq K_z$ such that $u \notin \{r, s\}$ **do** $CC(rs) \leftarrow CC(rs) - 1$:
end

Figure 2: The algorithms for updating the variable CC when an attachable edge is inserted or a removable edge is deleted

The algorithm AddEdge1 (see Fig 2) takes as input a graph G, an attachable edge $uv \in \overline{E(G)}$, a clique tree T of G and the value of CC for the edges of G. It finds the closest vertices K_x and K_y in T containing respectively u and v. Let $K_z = (K_x \cap K_y) \cup \{u, v\}$. For each edge with both endpoints in K_z, the value of CC is increased by one. If $K_x \subset K_z$ then for each edge with both endpoints in K_x, the value of CC is decreased by one. Similarly if $K_y \subset K_z$ then for each edge with both endpoints in K_y, the value of CC is decreased by one.

Lemma 4. *Let $uv \in \overline{E(G)}$ be an attachable edge. Let T be a clique tree of G such that K_xK_y is an edge of T and $u \in K_x$ and $v \in K_y$. Then, after adding uv, the value of $CC(rs)$ remains unchanged for all rs such that $\{r, s\} - (K_x \cup K_y) \neq \emptyset$.*

Proof: Let rs be such that $\{r, s\} - (K_x \cup K_y) \neq \emptyset$. By the clique intersection property, the set of cliques containing both r and s induces a subtree T_{rs} of T not containing both K_x and K_y. By Remark 1, after the execution of AddEdge-CliqueTree, all the maximal cliques different from K_x and K_y remain unchanged. Therefore if T' is the clique tree of $G + uv$ after the execution of AddEdge-CliqueTree, then the subtree of T' induced by the cliques containing both r and s is equal to T_{rs}. It follows that the value of $CC(rs)$ remains unchanged. □

Analogously we have the following Lemma, the proof of which is very similar to the proof of Lemma 4, and is therefore, omitted.

Lemma 5. *Let $uv \in E(G)$ be a removable edge and K_x the maximal clique containing it. Then, after removing uv, the value of $CC(rs)$ remains unchanged for all $rs \in E(G)$ such that $\{r, s\} - K_x \neq \emptyset$.*

Lemma 6. *The algorithm* RemoveEdge1 *is correct.*

Proof: Let uv be the edge being removed and let K_x be the clique of G containing uv. We must show that the values of $CC(rs)$ after the elimination of uv represent the number of maximal cliques of $G - uv$ containing rs for each $rs \in E(G) - \{uv\}$. By Lemma 5, only edges with both endpoints in K_x may have the value of CC modified.

With reference to the algorithm RemoveEdge-CliqueTree, after the elimination of uv, the clique tree is modified in the following manner. First, two new cliques $K_x^u = K_x - \{v\}$ and $K_x^v = K_x - \{u\}$ are added and one clique, K_x, is deleted. Therefore in $G - uv$ all the edges of K_x incident in u (resp. v) are contained in the same number of cliques of G, while the edges of K_x not incident in u and not incident in v are contained in $K_x^u \cap K_x^v$ and the number of cliques containing them is increased by one. Then if K_x^u is contained in another clique K_y, we replace K_x^u with K_y. But then each edge with both endpoints in K_x^u has the number of cliques containing it decreased by one. Similarly if K_x^v is contained in a clique K_z then each edge with both endpoints in K_x^v has the number of cliques containing it decreased by one. □

Lemma 7. *The algorithm* AddEdge1 *is correct.*

Proof: Let uv be the edge being added and let K_x and K_y be the clique of G containing respectively u and v. By Lemma 4, only edges of $E(G)$ with both endpoints in $K_x \cup K_y$ may have the value of CC modified.

With reference to the algorithm AddEdge-CliqueTree, after the addition of uv, the clique tree is modified in the following manner. First, a new clique $K_z = (K_x \cap K_y) \cup \{u, v\}$ is added. Clearly in $G + uv$, each edge with both endpoints in $(K_x \cap K_y) \cup \{u, v\}$ has the number of cliques containing it increased by one. Then if K_x is contained in K_z, we replace K_x with K_z. But then each edge with both endpoints in K_x has the number of cliques containing it decreased by one. Similarly if K_y is contained in K_z then each edge with both endpoints in K_y has the number of cliques containing it decreased by one. □

The complexity of algorithm RemoveEdge1 is $O(n + m)$. In fact we update the value of CC for each edge rs such that $\{r, s\} \subseteq K_x^u \cap K_x^v$ where uv is the edge being removed. This requires $O(m)$ since there can be at most m edges with both endpoints in $K_x^u \cap K_x^v$. Finding a maximal clique K_y such that $K_x^u \subset K_y$ and finding a maximal clique K_z such that $K_x^v \subset K_z$ requires $O(n)$ time as seen in Section 3. Similarly decreasing the value of CC for each rs with both endpoints in K_x^u (resp. K_x^v) if $K_x^u \subset K_y$ (resp $K_x^v \subset K_z$) requires, at most, $O(m)$ time.

Also the complexity of algorithm AddEdge1 is $O(n + m)$. In fact we update the value of CC for each edge rs, $\{r, s\} \subseteq K_z$. This require $O(m)$ since there can be at most m edges with both endpoints in K_z. Similarly decreasing the value of CC for each rs with both endpoints in K_x (resp. K_y) if $K_x \subset K_z$ (resp $K_y \subset K_z$) requires, at most, $O(m)$ time. Therefore we have the following

Lemma 8. *The complexity of algorithms* RemoveEdge1 *and* AddEdge1 *is* $O(n + m)$

Example. Fig. 3 (a) shows a graph G where each edge is labeled by the corresponding value of CC. Fig. 3 (b) shows the graph $G - be$ and the new values of CC. In Fig. 3 (c), on the left, the initial clique tree of G is shown. The algorithm creates two cliques $K_3^b = \{b, c, d\}$ and $K_3^e = \{c, d, e\}$ (shown in Fig. 3 (c) on the center). Then since $K_3^b \subset K_1$ it contracts the edge $K_3^b K_1$ and replaces K_3^b with K_1 (shown in Fig. 3 (c) on the right).

4.1. Minimal triangulation algorithm

Note that we may use algorithm RemoveEdge1 to find a minimal triangulation of a given graph in time $O(nf + f^2 + m)$ where $f = |F|$ and F is the set of fill edges of a triangulation of G. In order to find a minimal triangulation we may execute the following procedure: find a triangulation G^+ of G. Then recursively delete from G^+ and F a removable edge of F. If no edge of F is removable then the triangulation is minimal. The correctness of the above procedure is based on the following

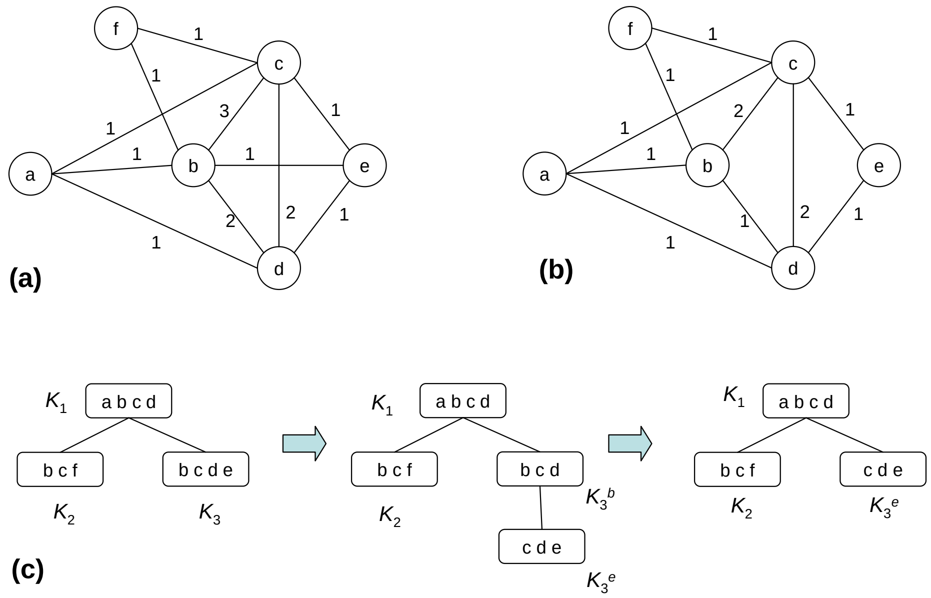

Figure 3: An example of execution of algorithm REMOVEEDGE1. (a) The initial graph where each edge is labeled by the value of CC. (b) The graph after removal of be. (c) The transformation of the clique tree of G: to the left the initial clique tree. In the center the clique tree after replacing K_3 with K_3^b and K_3^e. To the right the clique tree of $G - be$.

Lemma 9 ([24]). *Let G^+ be a triangulation of G. Then $E(G^+) - E(G)$ has at least one removable edge if and only if G^+ is not minimal.*

Given a graph G a triangulation G^+ of G can be found in time $O(n + m + f)$ [26]. Then we initialize and modify the variable CC only for the edges of F. The initialization of CC requires $O(nf)$ time. The execution of algorithm REMOVEEDGE1 requires $O(n+f)$ time since we update the value of CC only for the edges of F. Since at most f edges can be removed, the total running time of the algorithm is $O(f(n+f)+m)$.

5. Algorithms and data structures for supporting INSERT-QUERY

In order to support the operation INSERT-QUERY in $O(1)$ time we use a binary $n \times n$ matrix A containing for each $uv \in \overline{E(G)}$ a boolean value which is one if uv is attachable and zero otherwise. Therefore using the matrix A it requires $O(1)$ time to check if an edge is or is not attachable. Given a chordal graph G one may initialize the matrix A by using operation INSERT-QUERY of Theorem 5 in time $O(n\overline{m})$.

When an edge is inserted or deleted we will use the clique tree in order to update the matrix A in time bounded by $O(n^2)$.

Let $uv \in \mathcal{E}(G)$ and denote by $G * uv$ the graph $G + uv$ if $uv \in \overline{E(G)}$ and $G - uv$ if $uv \in E(G)$. Denote by E_{uv} the subset of $\overline{E(G)}$ such that $rs \in E_{uv}$ if and only if $CN_G(rs) \neq CN_{G*uv}(rs)$.

Fact 1. *Let $uv \in \mathcal{E}(G)$ then*

$$\begin{aligned} E_{uv} = \{rs \in \overline{E(G)} :& (s \in N_G(u) \wedge (r = v)) \vee (s \in N_G(v) \wedge (r = u) \\ & (r \in N_G(u) \wedge (s = v)) \vee (r \in N_G(v) \wedge (s = u)\} \end{aligned}$$

Proof: The proof follows by the definition. □

Denote by $\overline{E_{uv}}$ the set $\overline{E(G)} - E_{uv}$. By definition we have the following

Fact 2. *Let G be a graph and $uv \in \mathcal{E}(G)$. Then for every $rs \in \overline{E_{uv}}$ we have that $CN_G(rs) = CN_{G*uv}(rs)$.*

We also need the following Lemma

Lemma 10. *Let G be a connected chordal graph. Let uv be an attachable edge and $rs \neq uv$ be an attachable edge such that r is connected or coincides with u in $G - CN_G(uv)$ and s is connected or coincides with v in $G - CN_G(uv)$. Then $CN_G(uv) = CN_G(rs)$.*

Proof: By Theorem 2, $CN_G(uv)$ is a minimal uv-separator. Since r and s are connected in G and not connected in $G - CN_G(uv)$, it follows that $CN_G(uv)$ is a rs-separator. Then $CN_G(rs) \subseteq CN_G(uv)$. If there exists a vertex $x \in CN_G(uv) - CN_G(rs)$, then r and s are connected in $G - CN_G(rs)$. But then we have a contradiction because by Theorem 2, $CN_G(rs)$ is a rs-separator. So it must be that $CN_G(uv) = CN_G(rs)$. □

5.1. Updating data structures supporting the INSERT-QUERY operation when an edge is added

Lemma 11 ([9]). *Let G be a chordal graph and $uv \in \overline{E(G)}$ an attachable edge. Let $rs \in \overline{E_{uv}}$ be an attachable edge of G. Then rs is not attachable in $G + uv$ if and only if r is connected to u in $G - CN_G(uv)$ and s is connected to v in $G - CN_G(uv)$.*

Proof: (*if*) Let $rs \in \overline{E_{uv}}$ such that r is connected to u and s is connected to v in $G - CN_G(uv)$. In order to show that rs is not attachable in $G + uv$, we show that r and s are connected in $G + uv - CN_{G+uv}(rs)$. Suppose that u and v are connected in G; it follows, by Lemma 10, that $CN_G(rs) = CN_G(uv)$. Therefore

$$\begin{aligned}
& G + uv - CN_{G+uv}(rs) = && \text{(by Fact 2)} \\
& G + uv - CN_G(rs) = && \text{(by Lemma 10)} \\
& G + uv - CN_G(uv)
\end{aligned}$$

Since r is connected to u and s is connected to v in $G - CN_G(uv)$ they are connected in $G + uv - CN_{G+uv}(uv)$. Then, by Theorem 2, rs is not attachable in $G + uv$.

Suppose now that u and v belong to two different connected components of G; then $CN_G(uv) = \emptyset$. If r is connected to u and s is connected to v then $CN_G(rs) = \emptyset$. Since by Fact 2, $CN_{G+uv}(rs) = CN_G(rs)$ and since r and s are connected in $G + uv$, by Theorem 2, rs is not attachable in $G + uv$.

(*only if*) We discuss only the case where r and s belong to the same connected component of G since the proof of the other case is similar and is omitted. Suppose that $rs \in \overline{E_{uv}}$ is attachable in G and is not attachable in $G + uv$. Then r and s are connected in $G + uv - CN_{G+uv}(rs)$. By Fact 2, $CN_{G+uv}(rs) = CN_G(rs)$ and this implies that r and s are connected in $G + uv - CN_G(rs)$. Since, by Theorem 2, r and s are disconnected in $G - CN_G(rs)$ any path connecting r and s in $G + uv - CN_G(rs)$ must contain uv. Therefore r is connected, say, to u and s is connected to v in $G - CN_G(rs)$. By Lemma 10, it follows that $CN_G(rs) = CN_G(uv)$ and therefore, r is connected to u and s is connected to v in $G - CN_G(uv)$. □

Lemma 12. *Let G be a chordal graph and $uv \in \overline{E(G)}$ an attachable edge. Every $rs \in \overline{E_{uv}}$ that is not attachable in G, remains not attachable in $G + uv$.*

Proof: Since rs is not attachable in G then $G - CN_G(rs)$ contains a path P connecting r and s. By Fact 2, $CN_G(rs) = CN_{G+uv}(rs)$. Then P is in $G + uv - CN_G(rs)$ and rs remains not attachable in $G + uv$. □

Lemma 13. *Let G be a chordal graph and $uv \in \overline{E(G)}$ an attachable edge. If an edge $rs \in E_{uv}$ is attachable in G then it remains attachable in $G + uv$.*

Proof: Since $rs \in E_{uv}$ it follows that $r \in \{u, v\}$. We discuss only the case where $r = v$, being the other case symmetric. Since rs is attachable in G then, by Theorem 2, $CN_G(rs)$ is an rs-separator. Since $su \in E(G)$ then r and s are disconnected in $G - CN_G(uv)$, and by Lemma 10, $CN_G(rs) = CN_G(uv)$. But then r and

Algorithm AddEdge2
Input: a chordal graph G, a clique tree T of G,
an attachable edge uv and the boolean matrix A
Output: the new value of $A(rs)$ for all $rs \in \overline{E(G+uv)}$.
begin
let C_u and C_v be the connected components of $G - CN_G(uv)$
containing u and v respectively;
for all $r \in C_u$ **and for all** $s \in C_v$ **do**
if $rs \in \overline{E_{uv}}$ and $A(rs) = 1$ **then** $A(rs) \leftarrow 0$;
for all $rs \in E_{uv}$ such that $A(rs) = 0$ **do**
begin
using Insert-Query of Theorem 5, check if rs
is attachable in $G + uv$ and update $A(rs)$ accordingly;
end
end

Figure 4: The algorithm AddEdge2 for updating the matrix A when an attachable edge is added to G

s are connected in $G + uv - CN_G(rs)$ only by u. Since $u \in CN_{G+uv}(rs)$ then r and s are disconnected in $G + uv - CN_{G+uv}(rs)$. □

By Lemma 11 and Lemma 12, when an attachable edge uv is added to G, in order to update the matrix A we need to find the set C_u of vertices connected to u in $G - CN_G(uv)$ and the set C_v of vertices connected to v in $G - CN_G(uv)$. Then for every $r \in C_u$ and for every $s \in C_v$ such that $rs \in \overline{E_{uv}}$ and $A(rs) = 1$, we set $A(rs) \leftarrow 0$. Finally by Lemma 13, we need to update the value of $A(rs)$ for all the edges $rs \in E_{uv}$ such that $A(rs) = 0$. We do this by using the operation Insert-query of Theorem 5. In Fig. 4 is shown the algorithm for updating the boolean matrix A when we add an attachable edge uv to G.

By what was said above and by Lemma 11, Lemma 12 and Lemma 13 we can state the following

Lemma 14. *The algorithm* AddEdge2 *is correct.*

Finding the set C_u and C_v requires $O(n+m)$. Then setting $A(rs) \leftarrow 0$ for all $rs \in \overline{E_{uv}}$ such that $r \in C_u$ and $s \in C_v$ require $O(\overline{m})$. Finally, since $|E_{uv}| \leq 2n$ using the operation Insert-query of Theorem 5, we can determine in $O(n^2)$ time which edges of E_{uv} become attachable and which do not after adding uv.

Lemma 15. *The complexity of algorithm* AddEdge2 *is* $O(n^2)$.

5.2. Updating data structures supporting the INSERT-QUERY operation when an edge is deleted

Lemma 16. *Let G be a chordal graph and $uv \in E(G)$ a removable edge. Let $rs \in \overline{E_{uv}}$ be a not attachable edge of G. Then rs is attachable in $G - uv$ if and only if r is connected to u in $G - uv - CN_G(uv)$ and s is connected to v in $G - uv - CN_G(uv)$ and $CN_G(rs) = CN_G(uv)$.*

Proof: (*if*) Suppose that $rs \in \overline{E_{uv}}$ is a not attachable edge of G, r is connected to u in $G - uv - CN_G(uv)$, s is connected to v in $G - uv - CN_G(uv)$ and $CN_G(rs) = CN_G(uv)$. By Lemma 3, u and v are in two different connected components of $G - uv - CN_G(uv)$. Since $rs \in \overline{E_{uv}}$, by Fact 2, $CN_G(rs) = CN_{G-uv}(rs)$. Furthermore, by hypothesis $CN_{G-uv}(rs) = CN_G(uv)$ and r is connected to u and s is connected to v in $G - uv - CN_G(uv)$. It follows that r and s are separated in $G - uv - CN_G(uv)$. By Theorem 2, rs is attachable in $G - uv$.
(*only if*) Suppose that $rs \in \overline{E_{uv}}$ is not attachable in G but is attachable in $G - uv$. Then, by Theorem 2, $CN_{G-uv}(rs)$ is an rs-separator in $G - uv$ or r and s belong to different connected component of $G - uv$.

In the following we discuss only the case when r and s are in the same connected component of $G-uv$, since the proof of the other case is similar. Since rs is not attachable in G, but is attachable in $G-uv$, then r and s are disconnected in $G-uv-CN_{G-uv}(rs)$ but connected in $G-CN_G(rs)$. Since by Fact 2, $CN_G(rs)=CN_{G-uv}(rs)$, it follows that any path connecting r and s in $G-CN_G(rs)$ contains uv. In other words r is connected in $G-uv-CN_{G-uv}(rs)$ to, say u, and s is connected to v in $G-uv-CN_{G-uv}(rs)$. Since uv is an attachable edge of $G-uv$ and rs an attachable edge of $G-uv$ then, by Lemma 10 we have that $CN_{G-uv}(rs)=CN_{G-uv}(uv)=CN_G(uv)$. □

Lemma 17. *Let G be a chordal graph and $uv \in E(G)$ a removable edge. Every $rs \in \overline{E_{uv}}$ that is attachable in G, remains attachable in $G-uv$.*

Proof: Since rs is attachable in G then, by Theorem 2, $CN_G(rs)$ is an rs-separator. By Fact 2, $CN_G(rs)=CN_{G-uv}(rs)$. Therefore if r and s are disconnected in $G-CN_G(rs)$ still they are disconnected in $G-uv-CN_{G-uv}(rs)$. □

Lemma 18. *Let G be a chordal graph and $uv \in E(G)$ a removable edge. If an edge $rs \in E_{uv}$ is not attachable in G then it remains not attachable in $G-uv$.*

Proof: Since $rs \in E_{uv}$, it follows that $r \in \{u,v\}$. We discuss only the case where $r=v$, being the other case symmetric. Since rs is not attachable in G then r and s are not disconnected in $G-CN_G(rs)$. Note that since $u \in CN_G(rs)$ then any path connecting r and s in $G-CN_G(rs)$ does not contains u. Therefore r and s are connected in $G-uv-CN_{G-uv}(rs)$. □

By Lemma 16 and Lemma 17, when a removable edge uv is deleted from G, in order to update the matrix A we need to find the set C'_u of vertices connected to u in $G-uv-CN_G(uv)$ and the set C'_v of vertices connected to v in $G-uv-CN_G(uv)$. Then for every $r \in C'_u$ and for every $s \in C'_v$ such that $rs \in \overline{E_{uv}}$ and $CN_G(uv)=CN_G(rs)$ we set $A(rs) \leftarrow 1$. Finally by Lemma 18, we update the value of $A(rs)$ for all the edges in E_{uv} such that $A(rs)=1$. We do this by using the Insert-query of Theorem 5. By Lemma 16, Lemma 17 and Lemma 18 we can state the following

Lemma 19. *The algorithm* RemoveEdge2 *is correct.*

In order to analyze the complexity of algorithm RemoveEdge2 we need first the following

Lemma 20. *Let $uv \in E(G)$ be a removable edge of G and $rs \in \overline{E_{uv}}$ such that r is connected to u and s is connected to v in $G-uv-CN_G(uv)$. Then $CN_G(uv)=CN_G(rs)$ if and only if $CN_G(uv) \subseteq N_G(r)$ and $CN_G(uv) \subseteq N_G(s)$*

Proof: (*only if*) Clearly if $CN_G(uv)=CN_G(rs)$ then $CN_G(uv) \subseteq N_G(r)$ and $CN_G(uv) \subseteq N_G(s)$.
(*if*) If $CN_G(uv) \subseteq N_G(r)$ and $CN_G(uv) \subseteq N_G(s)$ then $CN_G(uv) \subseteq CN_G(rs)$. Suppose, by contradiction, that there exists a vertex $x \in CN_G(rs)$ such that $x \notin CN_G(uv)$. Then r and s are connected in $G-uv-CN_G(uv)$ by x. Since r is connected to u and s is connected to v in $G-uv-CN_G(uv)$, u and v are not disconnected in $G-uv-CN_G(uv)$ and this contradicts Lemma 3. □

By Lemma 20 in order to find all the edge $rs \in \overline{E(G)}$ such that $CN_G(rs)=CN_G(uv)$ we need to find the sets C'_u and C'_v of vertices connected respectively to u and v in $G-uv$ such that $CN_G(uv) \subseteq N_G(r)$ and $CN_G(uv) \subseteq N_G(s)$. It requires $O(n)$ to find $CN_G(uv)$. Then given a vertex r it requires $O(n)$ time to check if $CN_G(uv) \subseteq N_G(r)$. Therefore it requires $O(n^2)$ to find the sets C'_u and C'_v. Furthermore it requires $O(n^2)$ to set $A(rs) \leftarrow 1$ for all $rs \in \overline{E_{uv}}$ such that $r \in C'_u$ and $s \in C'_v$. Finally, since $|E_{uv}| < 2n$ using the Insert-query of Theorem 5 that requires $O(n)$ to determine if an edge is attachable, we can determine in $O(n^2)$ time which edges of E_{uv} become attachable and which do not after removing uv. By what was said above we have the following

Lemma 21. *The complexity of algorithm* RemoveEdge2 *is $O(n^2)$.*

Algorithm RemoveEdge2
Input: a chordal graph G a clique tree T of G,
a removable edge uv and the boolean matrix A
Output: the new value of $A(rs)$ for all $rs \in \overline{E(G-uv)}$.
begin
let C_u and C_v be the connected components of $G - uv - CN_G(uv)$
containing u and v respectively;
let $C'_u \leftarrow \emptyset$ and let $C'_v \leftarrow \emptyset$;
for all $r \in C_u$ **do if** $CN_G(uv) \subseteq N_G(r)$ **then** add r to C'_u
for all $s \in C_v$ **do if** $CN_G(uv) \subseteq N_G(s)$ **then** add s to C'_u
for all $r \in C'_u$ and **for all** $s \in C'_v$ **do**
if $rs \in \overline{E_{uv}}$ and $A(rs) = 0$ **then** $A(rs) \leftarrow 1$;
for all $rs \in E_{uv}$ such that $A(rs) = 1$ **do**
begin
using Insert-Query of Theorem 5 check if rs
is attachable in $G - uv$ and update $A(rs)$ accordingly;
end
end

Figure 5: The algorithm RemoveEdge2 for updating the matrix A when a removable edge is deleted from G

6. The operations INSERT and DELETE

In Section 4 and Section 5 we discussed the algorithms that update the data structures for supporting the Insert-Query operation and for supporting the Delete-Query operation. In order to obtain the operations Insert and Delete we simply merge the algorithms AddEdge1 and AddEdge2 and the algorithms RemoveEdge1 and RemoveEdge2 as shown in Fig. 6.

By Lemma 8, Lemma 15 and Lemma 21 we have

Theorem 6. *The complexity of operations* Insert *and* Delete *is* $O(n^2)$.

In the work of Ibarra [17] the operations Insert and Delete have $O(n)$ complexity while our algorithms implements both operations in $O(n^2)$ time. However our algorithm performs better in those applications, such as those described in the Introduction or in Section 4.1, where the most frequent operations are the Insert-Query or Delete-Query. Therefore using our algorithm it takes $O(kn^2)$ to insert or delete k edges which is an improvement with respect to the Ibarra's algorithm which would requires $O(kn^3)$ time. Last, note that when we add or remove an edge from a chordal graph, potentially $O(m)$ removable edges can become not removable and at the same time $O(\overline{m})$ attachable edges can become not attachable. Therefore the time bound of $O(n^2)$ for the operations Insert and Delete, is probably, the best that we can obtain if we want to have at the same time a constant time bound for the operations Insert-Query and Delete-Query.

Acknowledgment

The author wishes to thank Louis Ibarra for his comments.

Algorithm INSERT
Input: a chordal graph G, its clique tree T, the boolean matrix A,
the variable CC and an attachable edge uv
Output: the new value of A and the new value of CC
begin
ADDEDGE1;
ADDEDGE2;
end

Algorithm DELETE
Input: a chordal graph G, its clique tree T, the boolean matrix A,
the variable CC and a removable edge uv
Output: the new value of A and the new value of CC

begin
REMOVEEDGE1;
REMOVEEDGE2;
end

Figure 6: The operations INSERT and DELETE

References

[1] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artif. Intell.*, 125(1-2):3–17, 2001.

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

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

[4] A. Berry, J. P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *J. Algorithms*, 58(1):33–66, 2006.

[5] A. Berry, A. Sigayret, and J. Spinrad. Faster dynamic algorithms for chordal graphs, and an application to phylogeny. In D. Kratsch, editor, *WG*, volume 3787 of *Lecture Notes in Computer Science*, pages 445–455. Springer, 2005.

[6] J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comput. Sci.*, 250(1-2):125–141, 2001.

[7] 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, Sparse Matrix Computations: Graph Theory Issues and Algorithms. Springer Verlag. IMA Volumes in Mathematics and its Applications*, 56:1–30, 1993.

[8] A. D'Atri and M. Moscarini. Recognition algorithms and design methodologies for acyclic database schemes. *Advances in Computing Research*, 3:43–67, 1986.

[9] A. Deshpande, M. N. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In J. S. Breese and D. Koller, editors, *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA*, pages 128–135. Morgan Kaufmann, 2001.

[10] G. Ding, R. F. Lax, J. Chen, P. P. Chen, and B. D. Marx. Comparison of greedy strategies for learning markov networks of treewidth k. In *Proceedings of the 2007 International Conference on Machine Learning, Las Vegas Nevada, USA*, pages 294–, 2007.

[11] G. Dirac. On rigid circuit graphs. *Abh. Math. Semin. Univ. Hamb.*, 25:71–76, 1961.

[12] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

[13] A. George and W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989.

[14] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

[15] P. Heggernes and B. Peyton. Fast computation of minimal fill inside a given elimination ordering. *SIAM Journal on Matrix Analysis and Applications*, To appear, 2008.

[16] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $o(n^{\alpha} \log n) = o(n^{2.376})$. *SIAM J. Discret. Math.*, 19(4):900–913, 2005.

[17] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Trans. Algorithms*, 4(4):1–20, 2008.

[18] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist*, 22(1):79–86, 1951.

[19] J. W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw.*, 11(2):141–153, 1985.

[20] F. Malvestuto. Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man and Cybernetics,*, 21(5):1287–1294, 1991.

[21] F. M. Malvestuto. Existence of extensions and product extensions for discrete probability distributions. *Discrete Math.*, 69(1):61–77, 1988.

[22] M. Mezzini. Fast minimal triangulation algorithm using minimum degree criterion. *Theor. Comput. Sci.*, 412(29):3775–3787, 2011.

[23] M. Mezzini and M. Moscarini. Simple algorithms for minimal triangulation of a graph and backward selection of decomposable markov network. *Submitted to Theoretical Computer Science*, 2009.

[24] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[25] N. Srebro. Maximum likelihood bounded tree-width markov networks. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA*, pages 504–511, 2001.

[26] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[27] M. Yannakakis. Computing the minimum fill-in is NP-Complete. *SIAM. J. on Algebraic and Discrete Methods*, 2(1):77–79, 1981.