Triangulation and Clique Separator Decomposition of Claw-Free Graphs

Anne Berry and Annegret Wagler

LIMOS UMR CNRS 6158*
Ensemble Scientifique des Cézeaux, Université Blaise Pascal,
63 173 Aubière, France
{berry,wagler}@isima.fr

Abstract. Finding minimal triangulations of graphs is a well-studied problem with many applications, for instance as first step for efficiently computing graph decompositions in terms of clique separators. Computing a minimal triangulation can be done in $O(nm)$ time and much effort has been invested to improve this time bound for general and special graphs. We propose a recursive algorithm which works for general graphs and runs in linear time if the input is a claw-free graph and the length of its longest path is bounded by a fixed value k. More precisely, our algorithm runs in $O(f + km)$ time if the input is a claw-free graph, where f is the number of fill edges added, and k is the height of the execution tree; we find all the clique minimal separators of the input graph at the same time. Our algorithm can be modified to a robust algorithm which runs within the same time bound: given a non-claw free input, it either triangulates the graph or reports a claw.

Keywords: claw-free graph, minimal triangulation, clique separator decomposition.

1 Background and Motivation

Chordal graphs are an important class, with properties similar to those of trees, and corresponding efficient algorithms. A graph is *chordal*, or *triangulated*, if it has no induced chordless cycle on 4 or more vertices; any non-chordal graph can be embedded into a chordal graph by adding a set of 'fill' edges, a process called 'triangulation'. Adding a minimum number of fill edges is an NP-complete problem [39], but adding an inclusion-wise minimal set of edges, thus obtaining a 'minimal triangulation', is polynomial.

Minimal triangulations have many applications (see [4], the recent survey [23] and references therein). Originally, the problem stemmed from sparse matrix computation [23,35], where triangulation was needed for Gaussian elimination in sparse symmetric systems, but it is also useful in other fields such as database management [1,38].

* Research partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

One of the more surprising applications for minimal triangulation is that it is currently a mandatory first step for efficiently computing a decomposition by clique separators. *Clique separator decomposition* was introduced by Tarjan [37] as hole- and C_4- preserving, and refined to a unique and optimal decomposition using clique *minimal* separators by [28]; this process consists in repeatedly finding a clique minimal separator and copying it into the connected components it defines (see [9] for details). This decomposition has attracted recent attention in the context of characterizing graph classes [12,13], for Bayesian networks [31] and for clustering gene expression data [26].

Regarding minimal triangulation, the seminal paper of Rose, Tarjan and Lueker [36] presented an $O(nm)$ time algorithmic process. Several other $O(nm)$ time algorithms have appeared recently [2,3,6,8]. Efforts have been invested into improving this $O(nm)$ time bound for the general case: [27] offer an $O(n^{2.69})$ time bound, later improved by [24] to $O(n^{\alpha} logn) = o(n^{2.376})$, where n^{α} is the time required to do matrix multiplication, currently $n^{2.376}$. For special graph classes, there are surprisingly few results which improve the time bound of the general case: so far, chordal bipartite graphs and hole-and-diamond-free graphs have been shown to have an $O(n^2)$ time algorithm [7]; co-comparability graphs and AT-free claw-free graphs have a linear-time triangulation algorithm [30]; co-bipartite graphs, a subclass of AT-free claw-free graphs, have an even better time, since only a subset of edges needs to be traversed [11].

In this paper, we address the issue of improving the computation of a minimal triangulation for claw-free graphs. For this class, many significant results were obtained during the past twenty years. A particularly active field of research is improving the running times for computing maximum (weight) stable sets in claw-free graphs [20,21,32,33], partly based on decomposing claw-free graphs in an appropriate way [16,17,18].

A further recent result by [15] showed the presence of a hole containing a pair of vertices when there is no clique separator between them. However, not much is known on minimal triangulations and separators for claw-free graphs. Investigating this aspect, we present some interesting properties, which enable us to taylor an algorithm for computing both a minimal triangulation and the clique minimal separators. We show that this algorithm runs in linear time if the input is a claw-free graph where the length of the longest chordless path is bounded. More precisely, the algorithm constructs a tree in the graph and runs in $O(f + km)$, where f is the number of fill edges and k is the length of a longest branch of the tree. In fact, we present a 'robust' algorithm which, given any graph as input, will either triangulate it in $O(f + km)$ or report a claw as negative certificate. Moreover, as the recognition of claw-free graphs in general is in $O(m^{1.69})$ [25], our algorithm may improve finding a claw on some inputs.

The paper is organized as follows: we will give in Section 2 some preliminaries on minimal triangulation, minimal separation, and clique separators. In Section 3, we present and explain our algorithmic process. Section 4 gives the algorithm and proves its correctness and complexity. In Section 5, we discuss the robustness of our algorithm. We conclude with some final remarks.

2 Preliminaries

Basics. All graphs in this work are connected, undirected and finite. A graph is denoted by $G = (V, E)$, with $|V| = n$ and $|E| = m$. We say that a vertex x *sees* a vertex y if $xy \in E$. The *neighborhood* of a vertex x in a graph G is $N_G(x)$, the *closed neighborhood* is $N_G[x] = N_G(x) \cup \{x\}$. The neighborhood of a set X of vertices is $N_G(X) = \cup_{x \in X} N_G(x) - X$. A *clique* is a set of pairwise adjacent vertices; we say that we *saturate* a set X of vertices when we add all the edges necessary to turn X into a clique. A *clique module* is a set X of vertices such that $\forall x, y \in X, N[x] = N[y]$. $G(X)$ denotes the subgraph induced by the set X of vertices, but we will sometimes just denote this by X. A *co-bipartite* graph is a graph whose vertex set can be partitioned into two cliques, K_1 and K_2; we will call an edge *external* if it is neither inside K_1 nor inside K_2. A *claw* is an induced subgraph on 4 vertices $\{x, y_1, y_2, y_3\}$ with 3 edges: xy_1, xy_2, xy_3 where x is called the *center* of the claw. The reader is referred to [22] and [14] for classical graph definitions and results.

Separators. A set S of vertices of a connected graph G is a *separator* if $G(V - S)$ is not connected. A separator S is an *xy-separator* if x and y lie in two different connected components of $G(V - S)$. S is a *minimal xy-separator* if S is an xy-separator and no proper subset S' of S is also an xy-separator. A separator S is said to be *minimal* if there are two vertices x and y such that S is a minimal xy-separator. For any minimal separator S in a connected graph, there are at least two connected components C_1 and C_2 of $G(V - S)$ such that $N(C_1) = N(C_2) = S$ (called *full components*); this means that for any pair (x, y) of vertices from the Cartesian product $C_1 \times C_2$, S is a minimal xy-separator; note also that every vertex of S sees both C_1 and C_2, and that, equivalently, any path from x to y must go through S.

Property 1. [5] In any non-complete graph, there is a clique module whose neighborhood is a minimal separator, called a moplex; the vertex numbered 1 by LexBFS belongs to a moplex.

A *clique minimal separator* is a minimal separator which is a clique.

The minimal separators included in the neighborhood of a vertex x are called the *substars* of x [6,29]. Computing these substars can be done as follows: let $\{C_1 \dots C_p\}$ be the connected components of $G(V - N[x])$; then S is a substar of x if (and only if) $S = N_G(C_i)$ for some $i \in [1, p]$. Thus the substars are exactly the neighborhoods of the components defined when x and its neighborhood are removed from the graph.

Chordal Graphs and Minimal Triangulations. A graph is *chordal* (or triangulated) if it contains no chordless induced cycle of length 4 or more. Chordal graphs can be recognized in linear time using Algorithm LexBFS [36]. A graph is chordal if and only if all its minimal separators are cliques [19]. Given a non-chordal graph $G = (V, E)$, the supergraph $H = (V, E + F)$ is a *triangulation* of G if H is chordal. F is the set of *fill edges*, $|F|$ is denoted by f. The triangulation is *minimal* if for any proper subset of edges $F' \subset F$, the graph $(V, E + F')$ fails to be chordal.

There is a strong relationship between minimal triangulations and minimal separators [34]. Minimal separators S and S' are said to be *crossing separators* in a connected graph if S' has at least one vertex in every connected component of $G(V - S)$ (the crossing relation is symmetric). A minimal triangulation can be computed by saturating a maximal set of pairwise non-crossing minimal separators (this set is of size less than n, whereas there may be an exponential number of minimal separators in a non-chordal graph). Saturating a minimal separator S causes all the minimal separators which were crossing with S to disappear. Thus a minimal separator is a clique if and only if it crosses no other minimal separator [6,34]. The process of repeatedly choosing a (non-clique) minimal separator of a graph and saturating it, until there is no non-clique minimal separator left, results in a minimal triangulation. The substars of a given vertex x are pairwise non-crossing [6].

Given a graph $G = (V, E)$ and a minimal triangulation $H = (V, E + F)$ of G, any minimal separator S of H is a minimal separator of G, and $G(V - S)$ has the same connected components as $H(V - S)$. Any clique minimal separator of G is a minimal separator of H [9,34].

Clique Separator Decomposition. The decomposition by clique minimal separators decomposes a graph (in a unique fashion) into *atoms* (also called *MP-subgraphs* [28]), which are characterized as maximal connected subgraphs containing no clique separator. In a chordal graph, the atoms are the maximal cliques.

In [9] it was proved that the decomposition into atoms can be obtained by repeatedly applying the following decomposition step, which we will call *block decomposition step*, until none of the subgraphs obtained contains a clique separator: let $G = (V, E)$ be a graph, S a clique minimal separator of G, $\{C_1 \dots C_p\}$ be the connected components of $G(V - S)$; decompose the graph into the following subgraphs: $G((C_1) \cup N(C_1)) \dots G((C_p) \cup N(C_p))$, which we will call *blocks*. Note that $N(C_i)$ is included in S and is a clique minimal separator in its own right.

Property 2. [9] After an application of the block decomposition step on a connected graph G using a clique minimal separator S, all the other minimal separators of G are partitioned into the blocks obtained, *i.e.* each minimal separator is included in exactly one block.

3 Algorithmic Process

In a previous work, we used substars to compute a minimal triangulation [2,6]. The basic step, which is then applied to each vertex of the graph successively, is very simple: choose an unprocessed vertex x; compute the substars of x and saturate them. In this work, we will use a variant of this algorithm: we will combine this basic step with a decomposition step by clique minimal separators. Since after saturation, the substars become clique minimal separators of the resulting graph, we can apply the block decomposition step to all the substars, in order to obtain the corresponding blocks. This results in the following decomposition step for a vertex x:

Decomposition Step 1

- *compute the set* $\{C_1 \dots C_p\}$ *of connected components of* $G(V - N[x])$ *and the corresponding substars* $\{S_1 = N(C_1) \dots S_p = N(C_p)\}$*;*
- *saturate all the substars, thus obtaining a graph* G'*;*
- *decompose the resulting graph into the neighborhood piece* $G'(N[x])$ *and the blocks of the form* $G'(C_i \cup S_i)$*.*

After applying Decomposition Step 1, we will independently triangulate the neighborhood piece; we will also recursively apply the algorithm to each of the resulting blocks.

In any new block obtained during the recursive decomposition process, we will carefully choose the next vertex x which will be the center of the next substar, such that x is in the new separator but not in any of the other previously defined ones:

Property 3. Let $G = (V, E)$ be a graph, let x_1 be a vertex of G, let C_1 be a connected component of $G(V - N[x_1])$, let $S_1 = N_G(C_1)$ be the corresponding substar of x_1, let B_1 be the block $S_1 \cup C_1$, let x_2 be a vertex of S_1 which is not universal in B_1, let C_2 be a connected component of $V - (N[x_2] \cup S_1)$ in the graph induced by B_1, let $S_2 = N_G(C_2)$ be the corresponding substar of x_2; then $S_2 - S_1$ contains at least one vertex.

Proof. Suppose by contradiction that $S_2 - S_1$ is empty; since S_1 is a minimal separator of G, x_2, which is in S_1, must see some vertex z in C_1; thus z is in $C_1 - C_2$, so $C_1 \neq C_2$, so $S_1 \neq S_2$. Let v be any vertex of C_2; since C_2 is connected, there must be a chordless path P in C_2 from v to z; let w be the first vertex of P to be in $C_1 - C_2$; w by definition is in S_2, as it is in $N(C_2)$, but it is not in S_1, a contradiction.

The benefit is that in the new block $B_2 = S_2 \cup C_2$, all the already computed fill edges are incident to x_2, and thus will not be traversed when searching for the substars of x_2 in block B_2. Moreover, we will be moving along a chordless path during the recursive descent, as the newly chosen vertex x_2 will be adjacent to its father vertex x_1, but not to the father of x_1. The algorithm stops when there are no new substars defined for the chosen vertex x in the new subgraph, which means that x is universal in the block.

By Property 2, after an application of Decomposition Step 1, the remaining minimal separators are partitioned into the various subgraphs obtained. Therefore, when processing these subgraphs separately, we obtain the fill edges defining a minimal triangulation H of G, the minimal separators of H and the clique minimal separators of G.

We can initiate the algorithm with a vertex which belongs to a moplex, which is easy to find according to Property 1; after this minimal separator is saturated, the neighborhood piece is a clique, so no additional effort is required to triangulate it.

This procedure works on general graphs. However, the time complexity may not be interesting, because the successive neighborhood pieces are not easy to triangulate except for the first one, and the substars defined at a given step may overlap. In fact, checking whether the substars are clique separators and if not, computing the fill edges necessary to saturate them, can be costly or require a non-trivial data structure as is the case of the algorithm from [6]. Finally, when the blocks overlap, a given vertex may be processed many times, with possibly huge edge overlaps caused by copying the saturated substars.

For claw-free graphs, however, we will establish the following properties, which will make this an efficient process.

- The first property is that in a claw-free graph, when we initiate the algorithm by choosing a vertex x in a moplex, x defines a single substar (Corollary 1) and thus a single block to start with.
- The second property is that if we initiate our algorithm as described above, at each step of the process, the neighborhood piece obtained is a co-bipartite graph with a universal vertex added to it (Theorem 1 c)). A co-bipartite graph can be triangulated in time proportional to the number of external edges [11]. Moreover, the external edges of the different neighborhood pieces encountered do not overlap, so traversing the fill edges will globally cost at most f, where f is the number of fill edges.
- The third property is an important invariant of Decomposition Step 1: each block obtained remains claw-free, even if the resulting global triangulation is not claw-free (Theorem 1 b)).
- Finally, at each step processing vertex x, the substars are pairwise disjoint, and the blocks are also pairwise disjoint (Theorem 1 a)). As a result, a given vertex will not be processed more than once; moreover, not all vertices will be processed, as only one vertex per substar is processed, which results in a possibly linear complexity.

The algorithm moves along what we will refer to as its *execution tree* T: it will define a succession of centers of substars, forming a partial subgraph which is a tree. We will now prove the properties above, which are special for claw-free graphs.

Lemma 1. *Let G be a connected claw-free graph, let S be a minimal separator of G; then $G(V-S)$ has exactly two components (which are thus both full components).*

Proof. Consider two full components C_1 and C_2 of $G(V-S)$. By definition, every vertex of S sees both C_1 and C_2. If there is a third component C_3 of $G(V-S)$, then at least one vertex $x \in S$ has at least one neighbor $c_3 \in C_3$, forming a claw with two of its neighbors $c_1 \in C_1$ and $c_2 \in C_2$.

Corollary 1. *In a claw-free graph $G = (V, E)$, any vertex x belonging to a moplex defines only one substar S. Moreover, $C = V - N[x]$ induces a connected graph, and $S = N(C)$.*

Lemma 2. *Let $G = (V, E)$ be a claw-free graph, S a minimal separator of G, C_1 and C_2 the full components of $G(V - S)$. For any vertex x of S, the sets $N(x) \cap C_1$ and $N(x) \cap C_2$ are cliques.*

Proof. As C_1 and C_2 are full components of $G(V - S)$, any vertex x of S sees both C_1 and C_2. If there were two non-adjacent vertices in $N(x) \cap C_i$, say $c_1, c_1' \in C_1$, then x would form a claw together with c_1, c_1' and any of its neighbors $c_2 \in C_2$ (or vice versa).

Theorem 1. *Consider a claw-free graph $G = (V, E)$ and apply Decomposition Step 1 using a vertex x. Let G' be the graph obtained from G after saturating all the substars of x. Furthermore, for any connected component C of $G(V - N[x])$, let $S = N_G(C)$ be the corresponding substar of x and $B = G'(S \cup C)$ be the corresponding block. Then we have:*

a) the substars of x as well as the resulting blocks are pairwise disjoint;
b) all the blocks are claw-free subgraphs of G';
c) in any block $B = G'(S \cup C)$, let us choose a vertex x' in S; the neighborhood piece $G'(N[x'])$ is a co-bipartite graph with x' as additional universal vertex.

Proof. Consider a claw-free graph G and adopt the above notations:

a) The substars of x are pairwise disjoint: suppose vertex x has 2 non-disjoint substars, $S_i = N(C_i)$ and $S_j = N(C_j)$, and let y be a vertex in $S_i \cap S_j$; y must see some vertex y_i in C_i, and y must see some vertex y_j in C_j; $\{y, y_i, y_j, x\}$ form a claw with center y. The blocks are pairwise disjoint, as each substar results in a unique block: no substar can define more that 2 connected components by Lemma 1; one component contains x, so the other, C, defines a unique block.
b) Every block B is claw-free: suppose by contradiction that B fails to be claw-free; since the only added edges compared to the input graph in B are the fill edges inside S, the claw in B must have its center in S. Let s be the center of a claw; since S is a clique in B, there must be 2 non-adjacent vertices in C which participate in the claw; but by Lemma 2, the neighborhood of s in C must be a clique, a contradiction.
c) The neighborhood piece $G'(N[x'])$ is a co-bipartite graph with the universal vertex x' added to it: one part of this neighborhood is a minimal separator S which has been saturated, as the processed vertex is chosen inside the minimal separator which defined the new block, and the other part is a clique by Lemma 2.

Example 1. Let us illustrate our process with the simple example shown in Figure 1.
a) Vertex x_0 is chosen first.

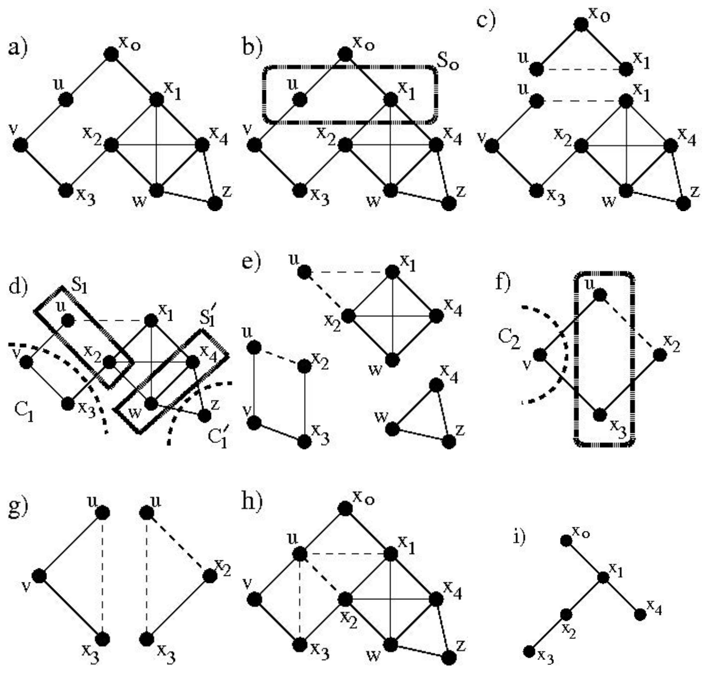

Fig. 1. The decomposition steps applied to a claw-free graph G

b) $N(x_0)$ is a minimal separator, which is $S_0 = \{u, x_1\}$, corresponding to the connected component $V - \{x_0, u, x_1\}$ of $V - N[x_0]$.
c) Fill edge ux_1 is added, and the neighborhood piece $\{x_0, u, x_1\}$ as well as the block $B_1 = V - x_0$ are defined.
d) In block B_1, x_1 is chosen in S_0; $N_{B_1}(x_1) = \{u, x_2, w, x_4\}$; the connected components of $V - N[x_1]$ in B_1 are: $C_1 = \{v, x_3\}$ and $C_1' = \{z\}$; the corresponding minimal separators are $S_1 = \{u, x_2\}$ and $S_1' = \{w, x_4\}$; saturating S_1 adds fill edge ux_2; S_1' is already a clique and thus is a clique minimal separator of the input graph G.
e) Block $B_2 = \{u, x_2, v, x_3\}$ is defined by S_1 and block $B_4 = \{w, x_4, z\}$ is defined from S_1'. The neighborhood piece $P_1 = N[x_1]$ defines a co-bipartite graph $N(x_1) = \{u, x_2, w, x_4\}$, which is already chordal, so no fill edge is added; there is only one minimal separator in this co-bipartite graph, namely the articulation point $\{x_2\}$, so $\{x_2\} \cup \{x_1\} = \{x_1, x_2\}$ is the unique associated (clique) minimal separator of G.
f) In block B_2, vertex x_2 is chosen in $S_1 - S_0$; in B_2, $N(x_2) = \{u, x_3\}$, defining only one connected component: $\{v\}$; the corresponding substar is $\{u, x_3\}$, fill edge ux_3 is added. The vertex x_2 becomes simplicial, so the corresponding neighborhood piece is complete and does not need to be processed.
g) Block $B_3 = \{u, x_3, v\}$ is then recursively defined; vertex x_3 is chosen; x_3 is simplicial and universal, so this branch of the recursive algorithm stops.

Afterwards, it only remains to choose x_4 in block B_4; x_4 is likewise simplicial and universal, so the algorithm terminates.
h) The resulting minimal triangulation is not claw-free, although each block and each piece was claw-free: the triangulation has a claw $\{u, v, x_2, x_0\}$ with center u.
i) The execution tree has a height of 3, requiring 3 linear-time graph searches globally; only 5 vertices are processed, out of a total of 9 vertices. Note that in the input graph, the longest path is of length 6.

4 The Algorithm

We now present our algorithm, consisting in a main algorithm (Claw-Free TRI-DEC), which initiates the process and a recursive algorithm (REC), which is called by the main algorithm. REC in turn uses another algorithm [11] which for a co-bipartite graph computes a minimal triangulation represented by the set F' of fill edges, the set $\mathscr{S}'$ of pairwise non-crossing minimal separators, and the set $\mathscr{K}'$ of clique minimal separators of the co-bipartite graph. We first present the algorithms Claw-Free TRI-DEC and REC in Section 4.1 and, for the sake of completeness, in Section 4.2 also the algorithmic process from [11] to compute a minimal triangulation of a co-bipartite graph.

4.1 Algorithms Claw-Free TRI-DEC and REC

Theorem 2. *Given a claw-free graph $G = (V, E)$, Algorithm Claw-Free TRI-DEC computes a minimal triangulation of G, represented by a set F of fill edges, and the minimal separators of $G' = (V, E + F)$, as well as the set of clique minimal separators of G in $O(f + km)$ time, where $f = |F|$, k is the height of the execution tree and $m = |E|$.*

Proof. To initialize our algorithm, let y be the vertex numbered 1 by an execution of LexBFS on a claw-free graph; note that y belongs to a moplex due to Property 1. By Corollary 1, y has only one substar, which is $S_0 = N(C_0)$, where $C_0 = V - N[y]$; this costs $O(m)$ time.

As the center of each new substar, we select a vertex x_i in $N_G(C_i) - S$, which always exists according to Property 3.

The Decomposition Step 1 is obviously properly applied in REC and produces a neighborhood piece and the necessary blocks with a single graph search. All the resulting blocks are again claw-free by Theorem 1 b) which guarantees a proper recursion. No fill edge is traversed in a given block, as all fill edges are incident to the vertex chosen as center of the next substar, so this search can be done in the input graph in $O(m)$ time.

Because the substars and blocks are pairwise disjoint at each step (Theorem 1 a)), there will be no extra cost incurred by searching the substars to determine which are clique minimal separators and which need to be filled.

The neighborhood piece without x is co-bipartite according to Theorem 1 c); it was shown in [11] that, given a co-bipartite graph with set of external edges

```
ALGORITHM Claw-Free TRI-DEC
Input  : A claw-free graph $G = (V, E)$.
Output: A set $\mathscr{S}$ of minimal separators defining a minimal triangulation of $G$,
        the corresponding set $F$ of fill edges, the set $\mathscr{K}$ of clique minimal
        separators of $G$, and an execution tree $T$ of $G$.

$y \leftarrow$ vertex 1 by LexBFS ; $C_0 \leftarrow V - N[y]$; $S_0 \leftarrow N_G(C_0)$; $x_0 \leftarrow$ a vertex of $S_0$;
$F \leftarrow \emptyset$; $T \leftarrow \{y\}$; // $T$ is a tree with only one node;
$\mathscr{S} \leftarrow S_0$; //minimal separators of the triangulation;
$\mathscr{K} \leftarrow \emptyset$; //clique minimal separators of $G$;
if $S_0$ is a clique then $\mathscr{K} \leftarrow \mathscr{K} + \{S_0\}$;
REC($S_0$, $C_0$, $x_0$);

REC
Input  : a minimal separator $S$ of $G$, a set $C$ of vertices which together with $S$
        defined a block in the father graph, and a vertex $x$ of $S$.
Output: modification of global variables.

$B \leftarrow G(S \cup C)$; // $B$ is the new block;
$K \leftarrow N_G(x) \cap C$; $X \leftarrow K \cup S$; // $X$ is the closed neighborhood of $x$ in $B$;
$G' \leftarrow G(X - \{x\})$; // $G'$ is the co-bipartite graph;
foreach connected component $C_i$ of $B(V - X)$ do
    $S_i \leftarrow N_G(C_i)$; $\mathscr{S} \leftarrow \mathscr{S} + S$; //$S_i$ is a substar of $x$;
    if $S_i$ is not a clique then $F_i' \leftarrow$ set of fill edges needed to saturate $S_i$;
    else $\mathscr{K} \leftarrow \mathscr{K} + S$;
    $G' \leftarrow G' + F_i'$; $F \leftarrow F + F_i'$;
$(F', \mathscr{S}', \mathscr{K}') \leftarrow$ set of fill edges, set of minimal separators and set of clique
minimal separators obtained by computing a minimal triangulation of the
co-bipartite graph $G'$;
$F \leftarrow F + F'$;
foreach minimal separator $S$ in $\mathscr{S}'$ do $\mathscr{S} \leftarrow \mathscr{S} + \{S + \{x\}\}$;
foreach clique minimal separator $S$ in $\mathscr{K}'$ do $\mathscr{K} \leftarrow \mathscr{K} + \{S + \{x\}\}$;
foreach substar $S_i$ of $x$ do
    CHOOSE a vertex $x_i$ in $N_G(C_i) - S$;
    $T \leftarrow$ add node $x_i$ and edge $xx_i$ to $T$;
    REC($S_i$, $C_i$, $x_i$);
```

E_{ext}, one can compute in $O(|E_{ext}|)$ time: a minimal triangulation $H = (V, E+F)$ of G, the set of minimal separators of H, the set of fill edges F, and the set of clique minimal separators of G. It is easy to see that if G' is a graph with a universal vertex x, then S is a minimal separator of G if and only if $S - \{x\}$ is a minimal separator of $G - \{x\}$, and S is a clique minimal separator of G if and only if $S - \{x\}$ is a clique minimal separator of $G - \{x\}$. The only fill edges to add are external edges; no two neighborhood pieces have common external edges: in each new block $S \cup C$ defined, each external edge has one endpoint in S and the other in C; the previous neighborhood piece contained S but no vertex of C; thus we avoid encountering each fill edge more than once. As a result, processing a neighborhood piece requires $O(m + f')$ time, where f' is the set of fill edges which must be added to the neighborhood piece to saturate the substars of x, and globally, processing all the neighborhood pieces costs $O(m + f)$ time.

In the course of Algorithm Claw-Free TRI-DEC on a claw-free graph G, all the minimal separators of G are encountered, and thus all the clique minimal separators of G are encountered: each minimal separator of G belongs either to a neighborhood piece or is a substar at some step; each minimal separator of the co-bipartite graph corresponding to the neighborhood piece is produced [11].

Each time the execution tree T has a branch, the graph is partitioned into several disjoint blocks, by Theorem 1 a). Therefore the set of edges of G is also partitioned, so each of the k layers of T can be processed globally in linear time. Thus if k is the height of T, Algorithm Claw-Free TRI-DEC runs in $O(f + km)$.

Note that in some graphs and with some executions, k may be small compared to the length of a longest chordless path in the graph, as in Example 1.

4.2 Minimal Triangulation of a Co-bipartite Graph

Recently [11], an efficient process for computing a minimal triangulation of a co-bipartite graph was presented. Given a co-bipartite graph $G = (K_1 + K_2, E)$ built on clique sets K_1 and K_2, this algorithm works in the complement $\overline{G} = (K_1 + K_2, \overline{E})$ of G; $\overline{G}$ is thus a bipartite graph.

The process works as follows: first, a maximal chain of the lattice formed by all the maximal bicliques of $\overline{G}$ is computed, thus computing a minimal subgraph of $\overline{G}$ which is a chain graph:

```
ALGORITHM MAX-CHAIN[11]
Input  : A bipartite graph $G' = (K_1 + K_2, E')$
Output: A maximal chain $\mathscr{C}$
prefix ← $\emptyset$ ; $\mathscr{C} \leftarrow \emptyset$;
repeat
    Choose a vertex $x$ of maximum degree in $K_1$; $X \leftarrow \{x\}$;
    $Y \leftarrow N(x)$;
    $G' \leftarrow$ remove $x$ and $K_2 - Y$ from $G'$;
    $U \leftarrow$ set of universal vertices of $G'$;
    $X \leftarrow X + U$;
    $G' \leftarrow$ remove all vertices of $U$ from $G'$;
    add $(prefix + X, Y)$ to $\mathscr{C}$;
    prefix ← $prefix + X$;
until $G'$ is empty;
```

Theorem 3. *[11] Algorithm* MAX-CHAIN *computes a maximal chain in* $O(min\,(|E'|, |\overline{E'}|))$ *time.*

Given a maximal chain of a bipartite graph $((X_1, Y_1), (X_2, Y_2), \ldots, (X_p, Y_p))$, the following inclusions hold: $X_1 \subset X_2 \subset \ldots Y_1$ and $Y_k \subset \ldots Y_2 \subset Y_1$.

It was shown in [10] that the sets $((K_1 - X_1) \cup (K_2 - Y_1)) \ldots ((K_1 - X_p) \cup (K_2 - Y_p))$ form a maximal set of pairwise non-crossing minimal separators of the corresponding co-bipartite graph. A corresponding minimal triangulation of the co-bipartite graph is obtained by adding to the co-bipartite graph any missing edge from the Cartesian products $(K_1 - X_1) \times (K_2 - Y_1), (K_1 - X_2) \times (K_2 - Y_2) \dots (K_1 - X_k) \times (K_2 - Y_k)$.

Since $X_1 \subset X_2 \subset \dots Y_1$ and $Y_k \subset \dots Y_2 \subset Y_1$, we need only to check for the absence of an edge in $(K_1 - X_1) \times (K_2 - Y_1), (K_1 - X_2) \times (Y_1 - Y_2), \dots, (K_1 - X_k) \times (Y_{k-1} - Y_k)$, which can also be done in $O(min(|E'|, |\overline{E'}|))$ time.

5 Robustness Properties of Algorithm Claw-Free TRI-DEC

Algorithm Claw-Free TRI-DEC can be used with a non-claw-free input, and will sometimes yield a minimal triangulation. If it fails to do so, we can detect a claw.

Let us consider the situation when the input graph G is not claw-free.

On the one hand, a neighborhood piece $G'(N[x])$ may fail to define a co-bipartite graph $G'(N(x))$ with cliques K_1 and K_2; while K_1 is a clique by saturation, K_2 is not necessarily a clique. In a claw-free graph, K_2 is a clique by virtue of Lemma 2; if this is not the case, let v, v' be two non-adjacent vertices in K_2, let y be the father of x in the execution; clearly, $\{x, v, v', y\}$ induce a claw with center x in G. Testing at each step whether these neighborhoods are cliques costs $O(m)$ time, so these claws will be found at no extra cost.

On the other hand, the substars of a vertex x may fail to be disjoint. In this case the graph has a claw: let y be the father of x in the execution; let C_1 and C_2 be two connected components of $G'(V - N[x])$ defining non-disjoint substars and $S_2 = N_{G'}(C_2)$ of x; let v be in $S_1 \cap S_2$, let v_1 be in $C_1 \cap N_{G'}(v)$, let v_2 be in $C_2 \cap N_{G'}(v)$; clearly, $\{v, v_1, v_2, y\}$ induce a claw with center v in G. As above, testing at each step whether these substars are disjoint costs $O(m)$ time, so these claws will be found at no extra cost.

When at each step the neighborhood piece is indeed co-bipartite and the substars are indeed disjoint, the algorithm will run correctly, even if the input graph contains a claw; such a claw could for example be included inside the first minimal separator S_0; the saturation of S_0 will 'hide' that claw, but will not prevent Algorithm Claw-Free TRI-DEC from running correctly.

6 Conclusion

We introduce new structural properties for the much-studied class of claw-free graphs. This leads to a new algorithm, which computes a minimal triangulation of a claw-free graph in $O(f + km)$ time, where f is the number of fill edges, and k is the height of the execution tree. When the graph is P_k-free for some bounded value of k, the algorithm runs in linear time. Even in the case where the graph is not P_k-free, the algorithm runs in linear time if the height of the execution tree is small. In any case, for a claw-free input, the algorithm runs in optimal $O(nm)$ time, but we expect it to run faster than the other minimal triangulation algorithms.

Our algorithm computes, at the same time, the set of all minimal separators of the minimal triangulation, as well as all the clique minimal separators of the input graph; the decomposition by clique minimal separators can be computed at no extra cost. The algorithm also defines a connected dominating set of the graph, since the execution tree obtained is dominating by construction.

It is worth examining whether the algorithm can be streamlined to run in linear time on quasi-line graphs, a subclass of claw-free graphs where the neighborhood of every vertex partitions into two cliques.

The algorithm can be re-written into a robust algorithm, which, in the same time bound, either computes a minimal triangulation of the input graph or reports a claw. It would be interesting to identify some criteria for non-claw-free graphs such that our robust version works without being disturbed by a claw.

A further question is whether the algorithm runs fast with other special classes, such as some sparse graphs.

References

1. Beeri, C., Fagin, R., Maier, D., Yannakakis, M.: On the Desirability of Acyclic Database Schemes. Journal of the ACM (JACM) 30(3) (July 1983)
2. Berry, A.: A wide-range efficient algorithm for minimal triangulation. In: Proceedings of SODA 1999, pp. 860–861 (1999)
3. Berry, A., Blair, J.R.S., Heggernes, P., Peyton, B.W.: Maximum cardinality search for computing minimal triangulations of graphs. Algorithmica 39(4), 287–298 (2004)
4. Berry, A., Blair, J.R.S., Simonet, G.: Preface to Special issue on Minimal Separation and Minimal Triangulation. Discrete Mathematics 306(3), 293 (2006)
5. Berry, A., Bordat, J.P.: Separability generalizes Dirac's theorem. Discrete Applied Mathematics 84(1-3), 43–53 (1998)
6. Berry, A., Bordat, J.-P., Heggernes, P., Simonet, G., Villanger, Y.: A wide-range algorithm for minimal triangulation from an arbitrary ordering. Journal of Algorithms 58(1), 33–66 (2006)
7. Berry, A., Brandstädt, A., Giakoumakis, V., Maffray, F.: The atomic structure of hole- and diamond-free graphs (submitted)
8. Berry, A., Heggernes, P., Villander, Y.: A vertex incremental approach for maintaining chordality. Discrete Mathematics 306(3), 318–336 (2006)
9. Berry, A., Pogorelcnik, R., Simonet, G.: An introduction to clique minimal separator decomposition. Algorithms 3(2), 197–215 (2010)
10. Berry, A., Sigayret, A.: Representing a concept lattice by a graph. Discrete Applied Mathematics 144(1-2), 27–42 (2004)
11. Berry, A., Sigayret, A.: A Peep through the Looking Glass: Articulation Points in Lattices. In: Domenach, F., Ignatov, D.I., Poelmans, J. (eds.) ICFCA 2012. LNCS (LNAI), vol. 7278, pp. 45–60. Springer, Heidelberg (2012)
12. Brandstädt, A., Hoàng, C.T.: On clique separators, nearly chordal graphs, and the Maximum Weight Stable Set Problem. Theoretical Computer Science 389, 295–306 (2007)
13. Brandstädt, A., Le, V.B., Mahfud, S.: New applications of clique separator decomposition for the Maximum Weight Stable Set Problem. Theoretical Computer Science 370, 229–239 (2007)

14. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. In: SIAM Monographs on Discrete Math. Appl., Philadelphia, vol. 3 (1999)
15. Bruhn, H., Saito, A.: Clique or hole in claw-free graphs. Journal of Combinatorial Theory, Series B 102(1), 1–13 (2012)
16. Chudnovsky, M., Seymour, P.: The Structure of Claw-free Graphs. In: Surveys in Combinatirics 2005. London Math. Soc. Lecture Note Series, vol. 327, pp. 153–171 (2005)
17. Chudnovsky, M., Seymour, P.: Claw-free Graphs IV. Decomposition theorem. Journal of Combinatorial Theory. Ser. B 98, 839–938 (2008)
18. Chudnovsky, M., Seymour, P.: Claw-free Graphs V. Global structure. Journal of Combinatorial Theory. Ser. B 98, 1373–1410 (2008)
19. Dirac, G.A.: On rigid circuit graphs. Abh. Math. Sem. Univ. Hamburg 25, 71–76 (1961)
20. Faenza, Y., Oriolo, G., Stauffer, G.: An algorithmic decomposition of claw-free graphs leading to an $O(n^3)$-algorithm for the weighted stable set problem. In: Proceedings of SODA 2011, pp. 630–646 (2011)
21. Faenza, Y., Oriolo, G., Stauffer, G.: Solving the maximum weighted stable set problem in claw-free graphs via decomposition (submitted)
22. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press (1980)
23. Heggernes, P.: Minimal triangulations of graphs: A survey. Discrete Mathematics 306(3), 297–317 (2006)
24. Heggernes, P., Telle, J.A., Villanger, Y.: Computing Minimal Triangulations in Time $O(n^{\alpha} logn) = o(n^{2.376})$. SIAM Journal on Discrete Mathematics 19(4), 900–913 (2005)
25. Kloks, T., Kratsch, D., Müller, H.: Finding and Counting Small Induced Subgraphs Efficiently. In: Nagl, M. (ed.) WG 1995. LNCS, vol. 1017, pp. 14–23. Springer, Heidelberg (1995)
26. Kaba, B., Pinet, N., Lelandais, G., Sigayret, A., Berry, A.: Clustering gene expression data using graph separators. In Silico Biology 7(4-5), 433–452 (2007)
27. Kratsch, D., Spinrad, J.: Minimal fill in $O(n^{2.69})$ time. Discrete Mathematics 306(3), 366–371 (2006)
28. Leimer, H.-G.: Optimal decomposition by clique separators. Discrete Mathematics 113, 99–123 (1993)
29. Lekkerkerker, C.G., Boland, J.C.: Representation of a finite graph by a set of intervals on the real line. Fund. Math. 51, 45–64 (1962)
30. Meister, D.: Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. Discrete Applied Mathematics 146(3), 193–218 (2005)
31. Olesen, K.G., Madsen, A.L.: Maximal prime subgraph decomposition of Bayesian networks. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics 32(1), 21–31 (2002)
32. Oriolo, G., Pietropaoli, U., Stauffer, G.: A New Algorithm for the Maximum Weighted Stable Set Problem in Claw-Free Graphs. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 77–96. Springer, Heidelberg (2008)
33. Oriolo, G., Stauffer, G., Ventura, P.: Stable sets in claw-free graphs: recent achievements and future challenges. Optima 86 (2011)
34. Parra, A., Scheffler, P.: Characterizations and Algorithmic Applications of Chordal Graph Embeddings. Discrete Applied Mathematics 79(1-3), 171–188 (1997)

35. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Graph Theory and Computing, pp. 183–217. Academic Press, NY (1973)
36. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM Journal on Computing 5, 266–283 (1976)
37. Tarjan, R.E.: Decomposition by clique separators. Discrete Mathematics 55, 221–232 (1985)
38. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM Journal on Computing 13, 566–579 (1984)
39. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM J. Alg. Disc. Meth. 2, 77–79 (1981)