New results on minimal triangulations

Yngve Villanger
Cand. Scient.

Dissertation for the degree philosophiae doctor (PhD)

Department of Informatics
University of Bergen
Norway

January 2006

Acknowledgements

First and foremost I would like to thank my supervisor Professor Pinar Heggernes, for excellent guidance and motivation. After I graduated from the master program as a Cand. Scient she encouraged me to apply for a PhD position, something that I am very grateful for today. Since then Pinar has been generous with her time, both when discussing my more or less vague ideas and when reading my drafts. Thanks Pinar, you have taught me a lot about research and life in general. :-)

I would also like to thank my co-authors Anne Berry, Jean-Paul Bordat, Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Genevieve Simonet, Karol Suchan, Jan Arne Telle, and Ioan Todinca for interesting discussions and productive collaboration.

In the spring semester of 2005 I spent two periods of almost three months at the LIFO (Laboratoire d'Informatique Fondamentale d'Orléans) of the University of Orléans, France. Many thanks goes to the people at LIFO for making this time interesting and productive. In particular I would like to thank Ioan and Alice Todinca, and Karol Suchan for making these two periods rewarding and memorable, both academically and socially. Especially when I, my wife and our little daughter where living in Orléans as a family.

The people in the Algorithm group at the Department of Informatics in Bergen deserves thanks for an open, friendly, and inspiring work environment. Special thanks goes to Christian Sloper as a colleague and a friend.

This work has been financially supported by a research fellow position from the University of Bergen. In addition to this L. Meltzers Høyskolefond and the AURORA mobility programme for research collaboration between France and Norway have contributed with travel founding.

I would also like to thank my parents Per Magne and Gerd who have always showed interest in my work and been supportive.

Finally I wish to thank my marvellous wife Tone for proof reading the introduction to this thesis. She also deserves thanks for being so understanding and patient especially during my work on completing this thesis.

This thesis is dedicated to our wonderful daughter Kristin.

Bergen, January 2006
Yngve Villanger

Introduction to the Thesis

Yngve Villanger

1 Background and motivation

Computing the *treewidth* and the *minimum fill-in* of a graph are two of the most well studied problems in the field of graph algorithms. Treewidth can be seen as a parameter that describes how close a graph is to a tree, while the minimum fill-in can be seen to describe how close a graph is to a *chordal* graph. The solution of each of these problems is equivalent to embedding a given graph into a chordal supergraph with some special properties. Unfortunately both these problems are *NP-hard* on general graphs [1, 47], thus no efficient polynomial time algorithms are known.

During the work on their graph minor project, Robertson and Seymour [40] introduced several new graph parameters as tools to prove their results, and treewidth was one of them. Later, treewidth has proved useful in many areas, where VLSI layout and evolution theory [11] are some examples. Even more important, treewidth is now widely accepted as one of the most important graph parameters, because a wide range of NP-hard problems on general graphs can be solved in polynomial time when the treewidth is bounded by some constant. (These algorithms are polynomial in the size of the input graph, but they are exponential in the treewidth of the graph.)

The minimum fill-in problem is also known as the *minimum triangulation* problem. An embedding of an input graph into a chordal graph can be obtained by adding edges until the graph becomes chordal. The edges added to the graph are called *fill* edges, and the resulting chordal graph is called a *triangulation*. Minimum triangulation is the problem of obtaining a chordal graph by adding the fewest possible number of fill edges, and the minimum fill-in is the number of such edges. The problem of finding the minimum fill-in was first studied in sparse matrix computations [42], but it has also applications in other areas, like database management [2, 43] and computer vision [15].

A *tree decomposition* is a way of decomposing the input graph into a tree, where each node of the tree corresponds to a vertex subset of the input graph. The definition of treewidth is based on tree decompositions. However it is interesting to notice that a tree decomposition describes a triangulation and every

triangulation describes a tree decomposition, thus these two structures are equivalent. The problem of computing the treewidth can be restated as the problem of computing a triangulation where the size of the largest clique is minimized. A consequence of this is that triangulations and tree decompositions can be used interchangeably when working on one of the two NP-hard problems mentioned above.

A *minimal triangulation* is a triangulation of the input graph such that no subset of the added edges results in a triangulation of the graph. Unlike treewidth and minimum triangulation, a minimal triangulation can be computed in polynomial time, where the best known time bounds are $O(nm)$ [6, 8, 36, 41] for sparse graphs and $O(n^{2.376})$ [26] for dense graphs, for an input graph with n vertices and m edges. For a survey about chordal graphs and minimal triangulation see [24].

Minimum fill-in and treewidth require searching for triangulations with different properties, but the optimal solution for both problems can be found among the minimal triangulations. Thus, minimum fill-in and treewidth problems can be solved by searching through the set of minimal triangulations of the input graph, which might be exponentially large. Minimal triangulations can be characterized in several different ways. Examples are characterizations through a tree decomposition, an *elimination ordering* of the vertices [37], and a set of *minimal separators* [38]. These and other characterizations will be explained in subsection 2.4.

Any ordering of the vertices in a graph defines a triangulation, which can be obtained by an algorithm called *the elimination game* [39]. If no fill edges are added by this algorithm, then the ordering is called a *perfect elimination ordering*. Fulkerson and Gross [21] showed that a graph is chordal if and only if it has a perfect elimination ordering. This was later used by Ohtsuki, Cheung, and Fujisawa [37] to define a minimal triangulation through an elimination ordering. Such an elimination ordering is called a *minimal elimination ordering*.

Vertex separators are structures that are central both in tree decompositions and in triangulations, and these separators can easily be reduced to *minimal separators* without increasing the treewidth or fill-in of a triangulation. The minimal separators of a graph are powerful enough to describe all interesting tree decompositions and triangulations [38], and thus the solution of treewidth, minimum triangulation, and a minimal triangulation can be defined by a set of minimal separators. It follows that each of these three problems can be reformulated as a problem of finding a set of minimal separators with some given property.

In this thesis we study properties of tree decompositions, minimal separators, and elimination orderings, and how these can be used as tools when constructing new algorithms for problems like minimal triangulation. We will now give some examples, where these structures are used. Besides from being a decomposition of a graph, a tree decomposition can also be used as a data structure that stores vertex separators. In contrast to a simple list structure, this tree structure contains

information about the relation between the separators. By using this information, it is possible to compute the union of a set of minimal separators in a more efficient way, as we showed in [6]. In special cases the union of a set of minimal separators can be found even faster by using an alternative representation of the tree decomposition, and an example of this data structure is presented in [8].

Minimal separators do not only separate the graph into at least two connected components, but they also separate the minimal triangulation problem into independent subproblems [31]. Even though this characterization of minimal triangulations has been implicit for some years, almost no algorithm took advantage of this property until 2004. In [26] we use this in combination with other techniques, to improve the running time for minimal triangulation of dense graphs.

Not all minimal triangulation algorithms are able to produce every minimal triangulation of an input graph. Even though two algorithms produce different sets of minimal elimination orderings, it is possible that they produce the same set of minimal triangulations, since many different elimination orderings can define the same triangulated graph. In [29] we define a set of modifications that can be done to an elimination ordering without changing the resulting triangulation. Based on these observations it is shown in [45] that two algorithms, Lex M [41] and MCS-M [4], that produce different sets of elimination orderings, actually produce the same set of triangulations.

As mentioned above, the treewidth and minimum fill-in can be found by searching through the set of minimal triangulations of the input graph. For each minimal triangulation, there exists a tree decomposition of the input graph defining the same set of fill edges. Each of the tree nodes in this tree decomposition is defined by and contains the information of a set of minimal separators in the minimal triangulation. These tree nodes are called *potential maximal cliques* [12] of the input graph, and can be used to define minimal triangulations [12], or to improve the time bound of exponential time algorithms [20].

The purpose of this introduction is to emphasize a set of relations and links between the papers that present the technical results of this thesis. The thesis consists of this introduction and five attached research papers following it, where the introduction is organized as follows. Section 2 provides definitions, and shows how several different structures and definitions that are commonly used in triangulation algorithms can be considered as separators in a graph. Section 3 presents the history behind some of the attached papers, and a summary of the technical results in each paper. The five papers that define the main body of this thesis are listed below. The list of papers is sorted chronologically according to the date each paper was submitted to a journal.

I. Anne Berry, Jean-Paul Bordat, Pinar Heggernes, Genevieve Simonet, and Yngve Villanger. *A wide-range algorithm for minimal triangulation from*

an arbitrary ordering. Journal of Algorithms. Volume 58, Issue 1, Pages 33-66, Year 2006. [6]

II. Anne Berry, Pinar Heggernes, and Yngve Villanger. *A Vertex Incremental Approach for Maintaining Chordality.* Discrete Mathematics. Volume 306, Issue 3, Pages 318-336, Year 2006. [8]

III. Yngve Villanger. *Lex M versus MCS-M.* Discrete Mathematics. Volume 306, Issue 3, Pages 393-400, Year 2006. [45]

IV. Pinar Heggernes, Jan Arne Telle, and Yngve Villanger. *Computing Minimal Triangulations in Time* $O(n^\alpha \log n) = o(n^{2.376})$. SIAM Journal on Discrete Mathematics. Volume 19, Number 4, Pages 900-913, Year 2005. [26]

V. Fedor V. Fomin, Ioan Todinca, Dieter Kratsch, and Yngve Villanger. *Exact algorithms for treewidth and minimum fill-in.* Submitted to SIAM Journal on Computing. [20]

2 Viewing everything as separators

Chordal graphs and the process of creating chordal graphs by adding edges to arbitrary graphs are two subjects that are common to all results presented in this thesis. If a chordal graph is obtained by adding edges to a non chordal graph, then this resulting graph is called a triangulation of the non chordal input graph. Both chordal graphs and triangulations can be characterized in several different ways, which will be discussed further, later in this introduction. These characterizations are useful tools when designing new triangulation algorithms, since each characterization defines a way to recognize or create a chordal graph. New triangulation algorithms are usually obtained by finding a new characterization and then combining this with already known characterizations, or by combining several known characterizations in a new way. Thus, knowing and understanding these characterizations are important when designing such algorithms. But before we can define and discuss these characterizations, some definitions are required. In order to give an alternative view of these problems and definitions, we will redefine several known structures used in triangulation algorithms as different types of separators.

2.1 Basic definitions

Graphs considered in this thesis are simple and undirected. A graph $G = (V, E)$ is a pair consisting of a set of vertices V and a set of edges E. The number of vertices is denoted by n, and the number of edges is denoted by m. Two vertices u, v are considered as *neighbors* if uv is an edge in E. The *neighborhood* of a

vertex u is denoted by the vertex set $N(u)$, where $v \in N(u)$ if $uv \in E$, and the *closed neighborhood* $N(u) \cup \{u\}$ of u is denoted by $N[u]$. For a vertex set $A \subseteq V$ the edge set $E(A)$ is given by $\{uv \in E \mid u, v \in A\}$. Let $G[A]$ denote the subgraph $(A, E(A))$ of G. We call $G[A]$ the subgraph of G *induced* by A. For simplicity we will write $G \setminus A$ for the induced subgraph $G[V \setminus A]$ of G. A vertex set $A \subseteq V$ is a *clique* if $uv \in E$ for every pair $u, v \in A$, and A is a *maximal clique* if there exists no clique A' such that $A \subset A'$. The opposite of a clique is an independent set, and the vertex set $I \subseteq V$ is an independent set if $uv \notin E$ for every pair $u, v \in I$.

An *ordering* of the vertices in a graph G is a function $\alpha : V \leftrightarrow \{1, 2, ..., n\}$. Let $v_0, v_1, ..., v_k$ denote a path from v_0 to v_k in G of length k, i.e., $v_i \neq v_j$ for $i \neq j$ and $v_i v_{i+1} \in E$ for $0 \leq i < k$. In the same way $v_0, v_1, ..., v_k, v_0$ denotes a cycle of length $k + 1$. A vertex set C containing u induces a *connected component* of G, if $v \in C$ for every pair u, v, such that there exists a path from u to v in G. A connected graph G is a *tree*, if G contains no cycles, and for every pair u, v of vertices in V, there exists a path between u and v in G.

2.2 Separators

A *vertex separator* is a vertex set such that a connected component of a graph becomes disconnected by removing this set. A vertex set $S \subset V$ is a *u, v-separator* in a connected graph $G = (V, E)$ with $u, v \in V$ if u and v are contained in different connected components of $G \setminus S$. Given a graph $G = (V, E)$, let $S \subset V$ be a u, v-separator of G, then S is a *minimal u, v-separator* of G if no proper subset of S separates u and v. If S is a minimal u, v-separator of $G = (V, E)$ for some pair $u, v \in V$, then S is a *minimal separator* of G.

Lemma 2.1 (Folklore) *Given a graph $G = (V, E)$, let $S \subset V$, and let C_1, C_2, ..., C_k be the connected components of $G \setminus S$. Then S is a minimal separator if and only if there exists a pair i, j, with $1 \leq i < j \leq k$, such that $S = N(C_i) = N(C_j)$.*

Proof. Let i and j be integers such that $S = N(C_i) = N(C_j)$, where $1 \leq i < j \leq k$, and let u and v be vertices such that $u \in C_i$ and $v \in C_j$. Since $S = N(C_i) = N(C_j)$ then it follows that u and v are contained in the same component of $G \setminus (S \setminus \{x\})$ for every vertex $x \in S$. Thus, S is a minimal u, v-separator since no subset of S separates u and v.

If S is a minimal separator, then there exists a pair u, v such that no subset of S separates u and v. Then u and v are contained in the same component C of $G \setminus (S \setminus \{x\})$ for every vertex $x \in S$, and every path from u to v in C contains the vertex x. Let C_u and C_v be the connected components of $G \setminus S$ containing u and v. Since both u and v have a path through vertices in $V \setminus S$ to every vertex $x \in S$, then it follows that $S = N(C_u) = N(C_v)$. ∎

A connected component C of $G \setminus S$ is called a *full component* of S if $S = N(C)$. Lemma 2.1 can now be restated as follows: A separator is minimal if and only if

it has at least two full components. Actually the neighborhood of any component of $G \setminus S$ is a minimal separator if S is a minimal separator of G, see Lemma 2.2. Another property of minimal separators is that no pair u, v of non adjacent vertices contained in a minimal separator S can be separated by a subset of the vertices in S. Notice that this is only true if $G \setminus S$ contains a connected component C, such that $u, v \in N(C)$. We can now use this component to restate the property: For any non adjacent pair u, v contained in a minimal separator S, there exists a component C of $G \setminus S$ such that $u, v \in N(C)$. Even though this property is trivial for minimal separators since they have at least two full components, it will be useful when we generalize the definition of separators further later in this text.

Lemma 2.2 *Given a graph $G = (V, E)$, let $S \subset V$, and let $C_1, C_2, ..., C_k$ be the connected components of $G \setminus S$. Then S is a minimal separator if and only if*

1. *C_j is a full component of S, for some j satisfying $1 \leq j \leq k$, and*
2. *$N(C_i)$ is a minimal separator of G, for every i satisfying $1 \leq i \leq k$, and*
3. *for any non adjacent pair $u, v \in S$, there exists an i such that $u, v \in N(C_i)$, where $1 \leq i \leq k$. (This requirement follows from the first, since $S = N(C_j)$ and thus $u, v \in N(C_j)$.)*

Proof. Let C_j be a full component of G, where $1 \leq j \leq k$, and let $N(C_1)$, $N(C_2), ..., N(C_k)$ be minimal separators of G. Then S is a minimal separator since $S = N(C_j)$ and $N(C_j)$ is a minimal separator of G.

Let S be a minimal separator of G. Then it follows from Lemma 2.1 that there exist at least two full components C_p and C_q of S, where $1 \leq p, q \leq k$ and $p \neq q$. The set $N(C_i)$ is a minimal separator for $1 \leq i \leq k$, since $G \setminus (C_i \cup S)$ contains one of the two full components of S, and thus there exists a full component of $N(C_i)$ in $G \setminus N[C_i]$. Finally for every non adjacent pair $u, v \in S$, the vertices u, v are contained in $N(C_p)$, since C_p is a full component, and thus $u, v \in S = N(C_p)$. ∎

A vertex separator is defined as a vertex set separating at least two vertices. This can be restated as separating every pair of vertices in a vertex set I, where $|I| = 2$. We can now generalize the definition of separators to separate every pair of vertices in a vertex set I, where $|I| \geq 0$[1]. Obviously none of the vertices contained in I can be adjacent.

It is tempting to define such a separator for a set in the following way. Let $I \subset V$ be an independent set in $G = (V, E)$, let $K \subseteq (V \setminus I)$, and let $C_1, C_2, ..., C_k$

[1]For the purpose of generalization we allow that $|I| < 2$, even though this is a bit against the intuition of separation since, no vertices are separated in this case.

be the connected components of $G \setminus K$. Then K is a *set separator* separating I if $|I \cap C_i| \leq 1$ for $1 \leq i \leq k$.

Since a set separator is a generalization of a vertex separator, we want minimal set separators to have similar properties as minimal separators. Unfortunately, inclusion minimal set separators do not behave like minimal separators. For instance, we want to maintain the property that no subset of a minimal set separator separates vertices contained in that minimal set separator. The following example shows that this does not hold for the above definition of set separators. Consider a simple cycle with eight vertices. Start from any vertex and number the vertices 1 to 8 in a clockwise order. Let I be vertices with odd number, and let K be vertices with even number. The set K is clearly a minimal set separator, since every vertex in K has two vertices from I in its neighborhood. Notice that no component of the graph where K is removed contains both the vertices numbered 2 and 6 or 4 and 8 in its neighborhood. Now we have a counterexample to the property since set $\{4, 8\}$ separates 2 and 6, and the set $\{2, 6\}$ separates 4 and 8.

Another property that we would like to preserve is that the neighborhood of the resulting connected components are minimal separators. The following example shows that this is not the case for the above definition of set separators. Consider a simple path with five vertices. Start in one end of the path and number the vertices successively towards the second end, with the numbers 1 to 5. Let I be the set of odd numbered vertices, and let K be the set of even numbered vertices. The set K is clearly a minimal set separator, since every vertex in K contains two vertices from I in its neighborhood. The vertex with number 3 is one of the resulting connected components when K is removed from the graph. Let us call this component C. This component does not satisfy the desired property, since $K = N(C)$, but K is not a minimal separator in the graph.

We will now define a separator for sets such that the following two properties are preserved in the inclusion minimal version of the separator: No subset of a minimal separator for a set separates vertices contained in the separator, and the neighborhood of any remaining connected component when the separator is removed is a minimal separator. Notice the similarities between Lemma 2.2 and Definition 2.3.

Definition 2.3 *Given a graph $G = (V, E)$, let $I \subset V$ be an independent set, let $K \subseteq (V \setminus I)$, and let $C_1, C_2, ..., C_k$ be the connected components of $G \setminus K$. Then K is a* BT-separator *separating I if*

1. *$|C_i \cap I| \leq 1$ for every i satisfying $1 \leq i \leq k$, and*
2. *$N(C_i)$ is a minimal separator of G, for every i satisfying $1 \leq i \leq k$, and*
3. *for every non adjacent pair $u, v \in K$ there exists an i, such that $u, v \in N(C_i)$, where $1 \leq i \leq k$.*

Definition 2.4 *Given a graph $G = (V, E)$, let $I \subset V$ be an independent set, and let $K \subseteq (V \setminus I)$ be a BT-separator separating I. Then K is a* minimal BT-separator *separating I if no subset of K is a BT-separator separating I.*

If K is a BT-separator separating some independent set I in a graph G, then we say that K is a BT-separator of G. We can now obtain the following result.

Lemma 2.5 *Given a graph $G = (V, E)$, let $K \subseteq V$, and let $C_1, C_2, ..., C_k$ be the connected components of $G \setminus K$. Then K is a* BT-separator *of G if and only if*

1. *$N(C_i)$ is a minimal separator of G, for $1 \leq i \leq k$, and*
2. *for every non adjacent pair $u, v \in K$ there exists an i, such that $u, v \in N(C_i)$, where $1 \leq i \leq k$.*

Proof. Let K be a vertex set such that $N(C_1), N(C_2), ..., N(C_k)$ are minimal separators of G, and such that for every pair $u, v \in K$, there exists an integer i such that $u, v \in N(C_i)$ and $1 \leq i \leq k$. Let I be a vertex set obtained by selecting one vertex from each of the connected components $C_1, C_2, ..., C_k$. The vertex set K separates I and thus it follows from Definition 2.3 that K is a BT-separator of G. The opposite direction of the proof follows directly, since the requirements are a subset of the requirements in Definition 2.3. ∎

Notice that if $G \setminus K$ has a full component for a BT-separator K of G, then K is a minimal separator, and thus K has at least two full components. It follows that a BT-separator has either zero or at least two full components. A BT-separator which is not a minimal separator, and thus has no full components, is known as a potential maximal clique [12]. This will be studied in detail in subsection 2.4.

Two vertex separators S and T of a graph G, are said to be *crossing* if S is a u, v-separator for a pair of vertices $u, v \in T$, or if T is an x, y-separator for a pair of vertices $x, y \in S$. This can be stated even stronger for minimal separators. Two *minimal* separators are said to be *crossing* if S is a u, v-separator for a pair of vertices $u, v \in T$, in which case T is an x, y-separator for a pair of vertices $x, y \in S$ [31, 38]. We will say that two BT-separators K and L of a graph G are *crossing* if there exists a component C_K of $G \setminus K$ and a component C_L of $G \setminus L$, such that $N(C_K)$ and $N(C_L)$ are crossing minimal separators.

Actually a BT-separator can be considered as a well chosen set of non-crossing minimal separators [12]. Notice that this set can actually be the empty set, like the BT-separator V in the complete graph $G = (V, E)$. We will now generalize the definition of separators further, such that a larger set of separators can be represented in the same structure. Crossing separators can be considered as rivals of each other, since at least one of them separates vertices in the other. If we limit our selves to only consider sets of non-crossing separators, then this allows us to represent several separators in a single structure.

Definition 2.6 *Given a graph $G = (V, E)$, let $S_1, S_2, ..., S_k$ be a set of non-crossing separators in G. Then $H = (V, F')$ is a* tree separator *representing the separators $S_1, S_2, ..., S_k$ if $uv \in F'$ for every pair of vertices u, v such that u and v are not separated by S_i for $1 \leq i \leq k$.*

Tree separators can be considered as a generalization of separators and thus also BT-separators, since every separator or BT-separator can be represented by a tree separator, while the opposite is not true. Some properties can be noticed about a tree separator H of G. Since no separator separates adjacent vertices in G, then $E \subseteq F'$, and since all the separators are non-crossing, then the vertex sets $S_1, S_2, ..., S_k$ are all cliques in H. Actually any BT-separator in H is a clique.

Lemma 2.7 *Let $H = (V, E \cup F)$ be a tree separator representing a set of non-crossing separators in $G = (V, E)$. Then every BT-separator of H is a clique.*

Proof. Let K be a BT-separator in H. If K is a clique, then there is nothing to prove. If K is not a clique, then there exists a pair u, v of non adjacent vertices in K. From the definition of H, we know that there exists a separator S represented by H separating u and v. Let S' be an inclusion minimal subset of S separating u and v. Since $uv \notin E \cup F$ and $u, v \in K$, then there exists a component C of $H \setminus K$ such that $u, v \in N(C)$. Let T be the minimal separator $N(C)$ in H. The minimal separator S' is now separating $u, v \in T$. Thus, it follows from [31, 38] that T separates two vertices $x, y \in S'$. This is a contradiction since $S' \subseteq S$, where S is a clique in H. We can now conclude that K is a clique in H, since there exists an edge in H between every pair of vertices in K. ∎

If a tree separator can be defined by a set of BT-separators, then we call this tree separator a *BT-tree separator*. We have now defined several new types of separators, and some of these contain others as special cases. By using these inclusion relations we can create a hierarchy between the definitions. The different inclusion relations are displayed on the left side of Figure 1. Separator definitions can also be partitioned into two groups, depending on whether they are defined through a vertex set or an edge set. The partition is as follows: minimal separators, minimal BT-separators, BT-separators, and vertex separators are defined through a vertex set, while BT-tree separators and tree separators are defined through an edge set. When first looking at Figure 1 it might seam like the tree separator is the optimal structure, since it contains all the other definitions as special cases. But notice that vertex set representation only requires $O(n)$ space, while an edge representation might require as much as $O(n^2)$ space. But a tree separator can also be represented with a set of vertex separators, and thus it can be represented with $O(n)$ times the number of vertex separators of space. This does not make a list of vertex separators an equal structure to a tree separator, since tree separators only store sets of non-crossing separators, while a list can

store any set of separators. In the next section we will see that finding sets of non-crossing separators are of interest to us, so the fact that tree separators never represent crossing separators will be a useful property.

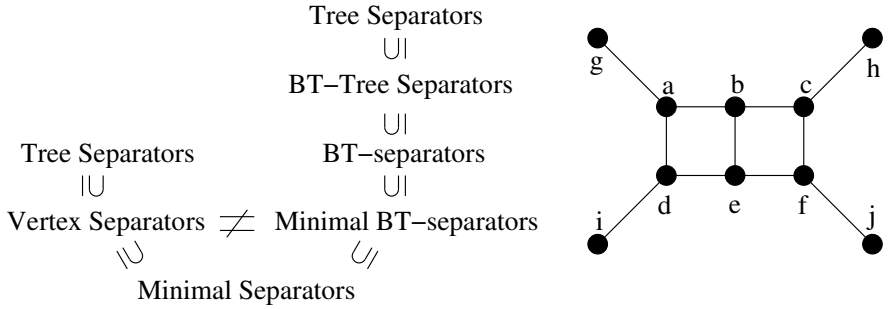

Figure 1: The left part of the figure shows the inclusion relation between the different types of separators defined in this section. We will use the graph on the right part of the figure to prove that the equality relation only holds in one direction. Let us start with the left branch of the inclusion relations to the left. The vertex pair $\{b, d\}$ in the graph to the right is a minimal separator, since it has two full components, and $\{a, b, d, e\}$ is a vertex separator since it separates g and i, and the vertex separators $\{a, b, d, e\}$ and $\{b, c, e, f\}$ define a tree separator, since they are non-crossing. The vertex separator $\{a, b, d, e\}$ is not a minimal separator, since it does not have a full component, and no vertex separator can represent the tree separator defined by the two vertex separators $\{a, b, d, e\}$ and $\{b, c, e, f\}$, since they separate vertices contained in one of the remaining connected components when the other vertex separator is removed. Let us now consider the right branch of the inclusion relations. The vertex pair $\{b, d\}$ is a minimal separator, and the vertex pairs $\{a, g\}$ and $\{b, d\}$ are minimal BT-separators, and $\{a, b, d\}$ is a BT-separator. The BT-separators $\{a, b, d\}$ and $\{c, e, f\}$ define a BT-tree separator, and finally the separators $\{a, b, c, d, e\}$ and $\{a, b, c, e, f\}$ define a tree separator. The minimal BT-separator $\{a, g\}$ is not a minimal separator since there is no full component, and the BT-separator $\{a, b, d\}$ is not a minimal BT-separator, since $\{b, d\}$ is a BT-separator separating the same set of vertices. Using the same arguments as for the left branch, we can argue that no single BT-separator can represent BT-tree separator defined by BT-separators $\{a, b, d\}$ and $\{c, e, f\}$, and the tree separator $\{a, b, c, d, e\}$ and $\{a, b, c, e, f\}$ can not be defined by a set of BT-separators, since no single BT-separator contains both a and c. It might also be noticed that the sets vertex separators $\neq$ minimal BT-separators, since $\{a, b, d, e\}$ is a separator and not a minimal BT-separator, and $\{a, g\}$ is a minimal BT-separator and not a vertex separator.

2.3 Chordal graphs

A graph is *chordal* if every cycle of length more than three has a chord. A chord is an edge between two non consecutive vertices in a cycle. The class of chordal graphs has been thoroughly studied since the early sixties, and has several interesting properties that will be useful to us. One of the first published results was that a chordal graph is either complete, or has two non adjacent simplicial vertices [18]. A vertex u is *simplicial* if $N[u]$ induces a clique in the graph. If the empty graph can be obtained from a graph by repeatedly removing simplicial vertices, then the order in which we remove the vertices is called a *perfect elimination ordering* (PEO) [21].

The definitions of *intersection graphs* and *tree decompositions* are useful when we talk about chordal graphs. A graph $G = (V, E)$ is the *intersection graph* of subtrees of a tree if there is a tree $T = (I, F)$, and for each vertex $u \in V$ a subtree T_u of T, such that for every pair $u, v \in V$, $uv \in E$ if and only if the trees T_u and T_v have at least one vertex in common.

Definition 2.8 *A* tree decomposition *of a graph $G = (V, E)$, is a pair (X, T) in which $T = (V_T, E_T)$ is a tree and $X = \{X_i \mid i \in V_T\}$ is a family of subsets of V such that:*

1. *$\bigcup_{i \in V_T} X_i = V$, and*
2. *for each edge $uv \in E$ there exists an $i \in V_T$ such that both u and v belong to X_i, and*
3. *for all $u \in V$, the set of tree nodes $\{i \in V_T \mid u \in X_i\}$ induces a connected subtree of T.*

Tree decompositions were defined and used by Robertson and Seymour [40] to define the *treewidth* of a graph. The width of a tree decomposition $(X, T = (V_T, E_T))$ of a graph G is the maximum of $|X_i| - 1$ for every $i \in V_T$, and the treewidth of the graph G is the minimum width over all tree decompositions of G. From now on we will simply refer to T when we mention a tree decomposition $(X, T = (V_T, E_T))$. The vertex subsets contained in X will be referred to as *tree nodes* of T, and the vertex set $X_i \cap X_j$ for an edge $ij \in E_T$ will be referred to as *tree edges* of T.

From the definition we know that chordal graphs do not contain chordless cycles. One consequence of this that can be deduced from [18] is that a chordal graph contains at most n maximal cliques. In [10] this property is used in a linear algorithm that lists all the maximal cliques and creates a tree decomposition of the chordal graph, where every tree node is a maximal clique. A result of this is that treewidth and the *clique number*(the size of the largest clique) of a chordal graph can be computed in linear time. But not all problems have efficient

polynomial time algorithms for chordal graphs. Examples of the opposite are *graph isomorphism* [35] and the problem of computing the *pathwidth* [23] for chordal graphs. This property that some problems that are NP-hard for general graphs have efficient algorithms for chordal graphs, while other problems remains NP-hard is one of several reasons that makes chordal graphs interesting. The number of different characterizations of chordal graphs gives an intuition of how well studied this class is. Some characterizations, directly or indirectly related to algorithms or proofs presented in this thesis, are listed below.

1. A graph is chordal if and only if every minimal separator is a clique. (1961 [18])
2. A graph is chordal if and only if every minimal separator contained in the neighborhood of a vertex is a clique. (1962 [34])
3. A graph is chordal if and only if it has a *perfect elimination ordering* (PEO). (1965 [21])
4. A graph is chordal if and only if it is the intersection graph of subtrees of a tree. (1974 [22])
5. A graph G is chordal if and only if there exists a tree decomposition of G such that every tree node is a maximal clique in G, and every tree edge is a minimal separator of G. (1972-74 [14, 22, 46])
6. A graph G is chordal if and only if all minimal separators in G are pairwise non-crossing. (1997 [38])
7. A graph $G = (V, E)$ is chordal if and only if $N(u) \cap N(v)$ is a minimal separator in $(V, E \setminus \{uv\})$, for every edge $uv \in E$. (2004 [8])
8. A graph $G = (V, E)$ is chordal if and only if G is a tree separator representing every BT-separator of G. (2006 here)

The following corollary can now be obtained from Characterization 6.

Corollary 2.9 *A graph G is chordal if and only if all BT-separators in G are pairwise non-crossing.*

Proof. Every minimal separator is also a BT-separator, so we know from Characterization 6 that the graph is chordal if all BT-separators in G are pairwise non-crossing. Now for the opposite direction of the proof. If two BT-separators are crossing, then by the definition of crossing BT-separators, the graph contains two crossing minimal separators. We can now conclude, also by Characterization 6 that the graph is not chordal. ∎

We will now use this new corollary to prove Characterization 8 in the list of characterizations.

Characterization 2.10 *A graph $G = (V, E)$ is chordal if and only if G is a tree separator representing every BT-separator of G.*

Proof. Let $H = (V, F')$ be a tree separator representing every BT-separator of G. Since no BT-separator of G separates consecutive vertices in G, and since all pairwise non consecutive vertices in G are separated by some BT-separator of G, then $F' = E$ and $H = G$. By Lemma 2.7 all BT-separators of H and G are cliques, and as a result all pairs of BT-separators of H and G are non-crossing. Thus, by Corollary 2.9 we can conclude that G is chordal.

Let G be chordal. Then we know from Corollary 2.9 that all BT-separators in G are pairwise non-crossing. From the definition of tree separators it follows that a tree separator $H = (V, F')$ representing every BT-separator of G can be constructed. For every pair of non adjacent vertices in G, there exists a BT-separator in G separating these vertices, and no BT-separator of G separates consecutive vertices in G. Thus, $G = H$, and the proof is complete. ∎

Actually the technique used in the proof of Characterization 2.10 is not the only way to relate tree separators to chordal graphs. The intersection graph of subtrees of a tree can also be used to create such a relation. Let T be a tree decomposition of a graph $G = (V, E)$. For every $u \in V$, let T_u be the tree nodes of T containing the vertex u, and by Definition 2.8 we know that T_u induces a connected subtree of T. The tree T and the subtrees T_u for every $u \in V$ define an intersection graph $H = (V, F')$ of subtrees of a tree. We will say that the graph H is *defined* by the tree decomposition T. From Characterizations 4 and 8 we know that H is chordal and that H is a tree separator representing some of the separators in G, and thus $E \subseteq F'$. A final notice might be that G is not necessarily a chordal graph, and in this case H is a chordal supergraph of G that is obtained by adding edges to G.

2.4 Triangulation and minimal triangulation

As we have seen in the previous subsection, chordal graphs can be obtained from non chordal graphs by adding edges. This is always possible, since a single tree node containing every vertex of the input graph is a tree decomposition of the input graph, and thus also defines a complete chordal supergraph of the input graph. If $H = (V, E \cup F)$ is a chordal supergraph of $G = (V, E)$, where $E \cap F = \emptyset$ then H is called a *triangulation* of G, and edges in F are called *fill* edges. We will say that the edge set F *defines* the triangulation H.

Some natural questions arise. What is the minimum number of fill edges that defines some triangulation of a given graph? Finding such a set of edges is known as the minimum fill-in or the minimum triangulation problem. This problem was conjectured to be NP-hard [41] in 1976, a conjecture that was confirmed by Yannakakis [47] five years later.

A simplification and a polynomial time version of the problem is to find an inclusion minimal set of fill edges, called a *minimal triangulation*. If $H = (V, E \cup F)$ is a triangulation of $G = (V, E)$, then H is a *minimal triangulation* of G if $H' = (V, E \cup F')$ is not a triangulation of G, for any edge set $F' \subset F$. Such triangulations can be obtained by a wide range of algorithms, where [9],[26], and [41] are some examples. Notice that a minimum triangulation is also a minimal triangulation, so the problem of finding a minimum triangulation can be considered as a problem of finding the *right* minimal triangulation.

Trivially a triangulation can be obtained by adding fill edges one by one until the graph becomes chordal. Even though checking if the obtained graph is chordal can be done in linear time [41, 43], it can be time consuming if we do this check at each step. In order to avoid this, triangulation algorithms introduce fill edges in a way that ensures that the resulting graph is chordal. This certificate is usually obtained by producing a perfect elimination ordering, or a tree decomposition which defines the resulting graph.

Without being aware of it, Parter[2] [39] presented the first triangulation algorithm, known as the *elimination game* in 1961. A graph $G = (V, E)$ and a vertex ordering α of G define the input to the algorithm. The elimination game adds edges to the input graph, such that the provided vertex ordering becomes a perfect elimination ordering of the resulting graph [21]. As a result, it follows that any graph produced by the elimination game is chordal. Another nice property of the algorithm is that it can be implemented to run in $O(n + m')$ time [43], where m' is the number of edges in the produced triangulation.

Using a tree decomposition is another way of defining a triangulation of the input graph. A triangulation defined by a tree decomposition can simply be created by completing every tree node in the tree decomposition into a clique. Notice that any triangulation algorithm that is based on finding and completing some separating vertex set into a clique, actually defines a tree separator or equivalently a tree decomposition. These triangulations can also be obtained in $O(n + m')$ time, by using a similar approach as the one used for the elimination game. But there is one difference: it can be easily verified that all triangulations can be defined by a tree decomposition, while this is not always possible by using an elimination ordering. For example, a complete graph can not be generated unless the input graph has a vertex adjacent to all other vertices. Despite this limitation, any minimal triangulation can be defined by a minimal elimination ordering [37]. This was proved by defining a *minimal elimination ordering*, which is an elimination ordering, such that no other ordering defines a triangulation using a strict subset of the fill edges.

Since a triangulation can be obtained in linear time and is defined by the elimination ordering or the tree decomposition, then the problem of finding trian-

[2]His aim was to give an algorithm that simulates Gaussian elimination on sparse matrices.

gulations with interesting properties is reduced to finding interesting elimination orderings or tree decompositions. We will now discuss different ways minimal triangulations can be defined.

In 2001 Bouchitté and Todinca [12] defined a *potential maximal clique* of a graph G to be a maximal clique in some minimal triangulation of G. These potential maximal cliques were key structures when Bouchitté and Todinca [13] showed that treewidth is polynomially tractable for all classes of graphs with a polynomial number of minimal separators. At first glance a potential maximal clique does not seem to be a very interesting structure when we are searching for minimal triangulations. Since a potential maximal clique is defined through a minimal triangulation of the input graph, it may seem that a triangulation should be computed before potential maximal cliques can be found. But this is not the case, since a potential maximal clique can be recognized directly in the input graph, and thus can be used to create the minimal triangulation it was defined from.

Theorem 2.11 ([12]) *Given a graph $G = (V, E)$, let $K \subseteq V$, and let C_1, $C_2, ..., C_k$ be the connected components of $G \setminus K$. Then K is a potential maximal clique if and only if*

1. *there exists no i such that C_i is a full component of K, where $1 \leq i \leq k$, and*

2. *$N(C_i)$ is a minimal separator of G, for $1 \leq i \leq k$, and*

3. *for any non adjacent pair $u, v \in K$ there exists an i such that $u, v \in N(C_i)$, where $1 \leq i \leq k$.*

The second requirement of Theorem 2.11 follows from the first and the third [12], but we add it to make it more similar to a previously defined structure: a BT-separator. It is interesting to notice that the only difference between the definition of a potential maximal clique in Theorem 2.11, and the definition of a minimal separator in Lemma 2.2, is the change in the first requirement from *none* to *some*. When we discussed BT-separators, we pointed out that BT-separators are either minimal separators, or have no full component. If a BT-separator has no full component, then it follows from Theorem 2.11 that this BT-separator is a potential maximal clique. Because of this strong similarity we have called these separators Bouchitté,Todinca-separators, or BT-separators for short. Thus, the set of BT-separators is the union of minimal separators and potential maximal cliques. Minimal BT-separators contain the set of minimal separators, but not the complete set of potential maximal cliques. An example of this is provided by a chordless four cycle. Every triple of vertices is a potential maximal clique, and only the two pairs of non adjacent vertices are minimal BT-separators.

Some of the motivation for finding characterizations of chordal graphs originates from the desire for finding new algorithms for minimal triangulations. Partly by the same motivation, the problem of finding minimal triangulations has been intensively studied. As a result, several characterizations for minimal triangulations have been published. We now give a list of characterizations of minimal triangulations.

1. A triangulation is minimal if and only if every fill edge is the unique chord of a 4-cycle in the triangulation. (1976 [41]) (Alternative formulation: A triangulation is minimal if and only if the removal of any single fill edge results in a non chordal graph.)

2. A triangulation is minimal if and only if it is defined by a minimal elimination ordering. (1976 [37])

3. Let S be a minimal separator of $G = (V, E)$, and let $G' = (V, E')$ be the graph obtained from G by completing S into a clique. Let further $C_1, C_2, ..., C_k$ be the connected components of $G \setminus S$. The graph $H = (V, E' \cup F)$ is a minimal triangulation of G if and only if $F = \bigcup_{i=1}^{k} F_i$, where F_i is the set of fill edges of a minimal triangulation of $G'[S \cup C_i]$. (1997 [31])

4. A triangulation $H = (V, E \cup F)$ is a minimal triangulation of $G = (V, E)$ if and only if it can be obtained by completing a maximal set of pairwise non-crossing minimal separators of G into cliques. (1997 [38])

5. Let K be a potential maximal clique of $G = (V, E)$, and let $G' = (V, E')$ be the graph obtained from G by completing K into a clique. Let further $C_1, C_2, ..., C_k$ be the connected components of $G \setminus K$, and $S_i = N(C_i)$ for $1 \leq i \leq k$. The graph $H = (V, E' \cup F)$ is a minimal triangulation of G if and only if $F = \bigcup_{i=1}^{k} F_i$, where F_i is the set of fill edges of a minimal triangulation of $G'[S \cup C_i]$. (2001 [12])

6. A tree separator $H = (V, E \cup F)$ defines a minimal triangulation of $G = (V, E)$ if and only if u and v are contained in a BT-separator of G represented by H, for every edge $uv \in F \setminus E$. (2006 here)

As mentioned earlier, the set of BT-separators is the union of minimal separators and potential maximal cliques. We can now merge Characterizations 3 and 5 into a single characterization in the following corollary.

Corollary 2.12 ([12, 31]) *Let K be a BT-separator of $G = (V, E)$, let $G' = (V, E')$ be the graph obtained from G by completing K into a clique. Let further $C_1, C_2, ..., C_k$ be the connected components of $G \setminus K$, and $S_i = N(C_i)$ for* $1 \leq$

$i \leq k$. The graph $H = (V, E' \cup F)$ is a minimal triangulation of G if and only if $F = \bigcup_{i=1}^{k} F_i$, where F_i is the set of fill edges of a minimal triangulation of $G'[S \cup C_i]$.

Characterization 4 can also be generalized to include BT-separators.

Characterization 2.13 *A triangulation $H = (V, E \cup F)$ is a minimal triangulation of $G = (V, E)$ if and only if it can be obtained by completing a maximal set of pairwise non-crossing BT-separators of G into cliques.*

Proof. Let $\mathcal{K}$ be a maximal set of pairwise non-crossing BT-separators of G. We will now use the set $\mathcal{K}$ to find a maximal set $\mathcal{S}$ of non-crossing minimal separators. For every BT-separator $K \in \mathcal{K}$, and for every connected component C of $G \setminus K$ add the minimal separator $N(C)$ to $\mathcal{S}$. Then all minimal separators in $\mathcal{S}$ are pairwise non-crossing. We will prove this by contradiction, where we assume that the minimal separators S and T in $\mathcal{S}$ are crossing. By [12] all minimal separators in a BT-separator are pairwise non-crossing, so S and T are added to $\mathcal{S}$ using two different BT-separators K_S and K_T. The BT-separators K_S and K_T are now crossing, since $S = N(C_S)$ and $T = N(C_T)$ for a component C_S of $G \setminus K_S$ and a component C_T of $G \setminus K_T$. This is a contradiction since all BT-separators in $\mathcal{K}$ are pairwise non-crossing, and K_S and K_T are contained in $\mathcal{K}$. Now back to the main proof. The set $\mathcal{S}$ is also a maximal set of minimal separators, since $\mathcal{K}$ is a maximal set of BT-separators, and minimal separators are BT-separators. Thus, any minimal separator that could be added to the set $\mathcal{S}$ could also be added to $\mathcal{K}$, which contradicts that $\mathcal{K}$ is a maximal set. By Characterization 4 it follows that a minimal triangulation of G is obtained by completing every BT-separator in $\mathcal{K}$ into a clique.

Let H be a minimal triangulation of G. By Characterization 4 we know that H can be obtained by completing a maximal set $\mathcal{S}$ of pairwise non-crossing minimal separators into cliques. Since minimal separators are also BT-separators, add every separator of $\mathcal{S}$ to the set $\mathcal{K}$, which contains BT-separators. Let us further add BT-separators to $\mathcal{K}$, such that $\mathcal{K}$ becomes a maximal set of pairwise non-crossing BT-separators. Notice that for every BT-separator $K \in \mathcal{K}$, and for every connected component C of $G \setminus K$ the minimal separator $N(C)$ is contained in $\mathcal{S}$, since K is not crossing any BT-separator in $\mathcal{S}$, and $\mathcal{S}$ is a maximal set of non-crossing minimal separators. Thus, there exists a maximal set $\mathcal{K}$ of non-crossing BT-separators, such that H is obtained by completing every BT-separator in $\mathcal{K}$ into a clique. ∎

Let us finally prove Characterization 6 which is new here.

Characterization 2.14 *A tree separator $H = (V, E \cup F)$ defines a minimal triangulation of $G = (V, E)$ if and only if u and v are contained in a BT-separator of G represented by H, for every edge $uv \in F$ where $E \cap F = \emptyset$.*

Proof. Let H be a tree separator of G, such that for every edge $uv \in F$, the vertices u and v are contained in a BT-separator of G represented by H. Let $\mathcal{K}$ be the set of BT-separators of G, which is represented by H. Only non-crossing BT-separators can be represented by a tree separator, so every pair of BT-separators in $\mathcal{K}$ are non-crossing. Notice that every BT-separator K of G represented by H is also a BT-separator of H, since every edge of G is contained in H, and every pair of vertices separated by K in G, is non adjacent in H. By Lemma 2.7 and Corollary 2.9 every BT-separator of H is a clique, and H is chordal. Thus, a triangulation of G can be obtained by completing the BT-separators in $\mathcal{K}$ into cliques, and by Characterization 2.13 H is a minimal triangulation of G.

Let H be a minimal triangulation of G. Then there exists a set $\mathcal{K}$ of pairwise non-crossing BT-separators of G, such that H is obtained by completing these BT-separators into cliques (Characterization 2.13). Thus, for every $uv \in F$ there exits a BT-separator in $\mathcal{K}$, which contains both u and v. ∎

As we have seen in this section, tree decompositions and potential maximal cliques can be considered as, or defined by, a set of pairs of vertices which are not adjacent. Most papers that discuss some kind of triangulation focus on finding fill edges and not pairs of vertices to separate in the final triangulation. There are at least two reasons for this. The first is that the triangulation problem is defined through a set of fill edges and not pairs of vertices to separate, and secondly it is harder to draw examples when the decision is to separate pairs of vertices and not add fill edges. But these two approaches are not necessarily equal. If some subset of the set of pairs of vertices we have decided to separate defines a BT-separator, then we might as well complete the BT-separator into a clique. But if this set of pairs of vertices only defines a separator and not a BT-separator, then this can be considered as a subproblem of the general triangulation problem. The reason why we can consider this as a subproblem is that some choices are made, and thus there are fewer final triangulations to choose among. Some examples of algorithms using this separating approach can be found in [6],[9], and [26].

3 History and relation to introduction

Apart from this introduction, this thesis consists of five papers, of which all are submitted to journals, and four are accepted for publication[3]. With the exception of Paper III, the main results of all of these papers are presented at some conference. Each of papers II and IV is a final full version of a single paper first presented at a conference. Each of papers I and V is a journal paper based on two separate conference papers. Each of the journal papers I and V was the result of merging a conference paper containing our results with another

[3]Some of the papers attached to this thesis have minor editing changes compared to the submitted version.

conference paper by different authors. Since a thesis should contain the most recent, well written, and full version of each result, we attach the merged journal papers in this thesis, and not the preliminarily conference versions.

This section contains five subsections, one for each paper. Each subsection has two purposes. The first is to tell a bit of the history behind the result. By history we mean to identify the conference papers (if any) the paper is based on, and in some cases also how the project started. The second purpose is to emphasize the relation to the introduction, by explaining how tree decompositions, minimal separators, and elimination orderings are used to obtain the results.

3.1 (Paper I [6]) Tree decomposition as a tool to compute minimal triangulations efficiently

Several different minimal triangulation algorithms have been presented since the mid seventies, where the LB-Triang algorithm by Berry [3] is one of the more recent. LB-Triang was first presented at ACM-SIAM Symposium on Discrete Algorithms (SODA) in 1999 [3], and it is based on Lekkerkerker and Boland's [34] characterization of chordal graphs. This characterization states that a graph is chordal if and only if every minimal separator contained in the neighborhood of a vertex is a clique. The LB-Triang algorithm describes a procedure for adding fill edges, such that this property is established. The time bound for LB-Triang was claimed in [3] to be $O(nm)$ without a proof, and the existence of such an implementation remained unproved until our result in 2002. A straight forward implementation of LB-Triang gives $O(nm')$ time complexity, where m' is the number of edges in the resulting minimal triangulation. Since $O(nm)$ time algorithms already existed [36, 41], the $O(nm')$ time complexity was not satisfactory, although the LB-Triang algorithm has many other nice properties.

An $O(nm)$ time implementation of LB-Triang was presented by Heggernes and Villanger at European Symposium on Algorithms (ESA) [28] in 2002. The implementation is based on a tree decomposition which preserves the information about the set of minimal separators found so far by the algorithm. This tree decomposition structure allows us to compute the information we need from the new fill edges in $O(m)$ time, and thus we obtain an $O(nm)$ time algorithm.

Paper I is based on the two conference papers [3] and [28]. The use of minimal separators in the algorithm, and the use of tree decompositions in the $O(nm)$ implementation, relate this paper to the structures mentioned in the introduction. We will now discuss how, and a bit why, tree decompositions make the difference in the time complexity. The LB-Triang algorithm processes each vertex of the graph in some order. This order is part of the input in the $O(nm)$ implementation, but can be provided in an on-line fashion for the $O(nm')$ implementation. For each vertex, LB-Triang completes the minimal separators contained in the neighborhood of the current vertex into a clique, and then this procedure is repeated

with the next vertex in the ordering. The final result is that every minimal separator contained in the neighborhood of some vertex is a clique. Thus, it follows from [34] that the resulting graph is a triangulation.

Completing minimal separators into cliques is the operation that requires $O(m')$ time for each separator in a naive implementation, and gives an $O(n^2m')$ algorithm. This can easily be improved to $O(nm')$ by recognizing duplicate separators, and complete each minimal separator into a clique only once. To improve the time bound further we need the observation that the added fill edges are only required to compute $N(u)$ if u is the next vertex to be processed. Thus, an $O(nm')$ time algorithm can also be obtained by scanning every minimal separator containing u once from a list containing only unique minimal separators, and in this way compute $N(u)$. Another important observation is that each minimal separator separates the remaining set of separators into subsets. Unlike a list structure, a tree decomposition can be used to store the minimal separators, while preserving this information. This extra information enables us to compute $N(u)$ from a subset of the minimal separators containing u, and to bound the sum of these to m. As a result we obtain an $O(nm)$ time implementation.

3.2 (Paper II [8]) A vertex incremental approach to compute minimal triangulations

An early version of the results of Paper II was presented at the International Symposium on Algorithms and Computation (ISAAC) [7] in 2003. The project resulting in Paper II started with the following question: Given a graph $G = (V, E)$, and a partition V_1, V_2 of V, such that $G[V_1]$ and $G[V_2]$ are chordal graphs, does there always exist a minimal triangulation H of G, such that $u \in V_1$ and $v \in V_2$, for every fill edge uv in H? As we can see in the example of Figure 2, there might not even be such a triangulation. This negative result removes the possibility of finding a minimal triangulation of a graph G, by triangulating $G[V_1]$ and $G[V_2]$ and then adding fill edges between V_1 and V_2, for any partition V_1, V_2 of V.

Let us now try to reformulate the question, such that we can obtain a positive result. By studying the graphs in Figure 2, we can observe that such a triangulation can be obtained if either V_1 or V_2 induces a clique. Let V_1 be the vertex set that induces a clique, and let us fix the size of V_1 to one. Fixing the size of V_1 to one has two advantages, the first is that all new fill edges are incident to the single vertex in V_1. The second is a bit more complicated. Let V_1 and V_2 be a partition of V for a graph $G = (V, E)$, and let H_2 be a minimal triangulation of $G[V_2]$. For a vertex $u \in V_1$ let $\{u\}, V_2$ be a partition of the vertices in $G[V_2 \cup \{u\}]$, notice that H_2 is still a minimal triangulation of $G[V_2]$. A minimal triangulation H_u of $G[V_2 \cup \{u\}]$ can now be obtained by supplying the fill edges of H_2 by some fill edges incident to u. In this way we obtain a new partition $V_1 \setminus \{u\}, V_2 \cup \{u\}$ of

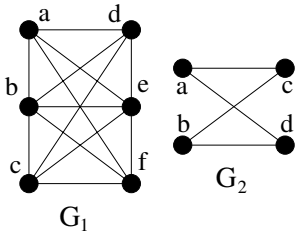

Figure 2: Let the partition of V in G_1 be as follows $V_1 = \{a, b, c\}$ and $V_2 = \{d, e, f\}$. No triangulation of G_1 can be obtained by adding edges between V_1 and V_2, since all these edges are already presented, and a, d, c, f, a is a chordless cycle of length four. If we allow $G[V_1]$ and $G[V_2]$ to be disconnected, then a smaller example can be found. By using the partition $V_1 = \{a, b\}$ and $V_2 = \{c, d\}$ of V in G_2 we obtain a similar example as obtained by G_1. Every edge between vertices in V_1 and V_2 is already present and a, c, b, d, a is a chordless cycle of length four. Unlike the example using G_1, the vertex sets V_1 and V_2 are disconnected, and of size two.

V, where H_u is a minimal triangulation of $G[V_2 \cup \{u\}]$. A minimal triangulation of G can now be obtained by repeating this until $V_1 = \emptyset$. The topic discussed in Paper II, corresponds exactly to this procedure, where $|V_2| \leq 1$ before the first iteration.

Paper II presents a single algorithm that can be used to compute both minimal triangulations and maximal subtriangulations of an input graph. A graph $H = (V, D)$, is a *maximal subtriangulation* of $G = (V, E)$ if D is an inclusion maximal subset of E, such that H is a chordal graph. This algorithm is based on a vertex incremental approach, a new technique in triangulation algorithms, but an already used technique on other problems like different types of graph recognition [17] and [32]. Since then the incremental approach has proved useful on other minimal completion problems [25].

The algorithm in Paper II uses the minimal separators in the chordal subgraph to find the new fill edges every time a new vertex u is introduced to the chordal subgraph. The fill edges incident to u are obtained by adding to $N(u)$ the union of a set of minimal separators in the chordal subgraph. In order to compute this set efficiently we need data structure that allows us to find and compute the union of these minimal separators efficiently. This is obtained by representing a tree decomposition in a new way.

The paper presents also a new way to implement tree decompositions. Unlike a regular representation of a tree decomposition, which represents each tree node as a vertex set, we only store the difference between tree nodes. This enables us to manipulate the tree decomposition such that every tree edge between two tree nodes are minimal u, v-separators for some vertex v, and to compute the union of

these minimal separators in $O(n)$ time, which is not possible in a standard vertex set representation. When this is combined with an amortized time analysis an $O(nm)$ time algorithm for minimal triangulation and maximal subtriangulation is obtained.

3.3 (Paper III [45]) Comparing minimal triangulation algorithms

Lex M [41] is one of the first two minimal triangulation algorithms published in 1976, and is based on a lexicographic breadth-first search. Almost 30 years later Berry, Blair, Heggernes, and Peyton presented a similar minimal triangulation algorithm called MCS-M [4]. MCS-M combines the cardinality labeling of neighbors used in MCS [43] (a linear time recognition algorithms for chordal graphs), with the labeling along paths used in Lex M. Both algorithms find minimal triangulations by producing a minimal elimination ordering. Some of the similarities between these algorithms can be exemplified with a simple cycle of length six. Both algorithms are capable of producing the same set of minimal triangulations, and none of them are able to produce minimal triangulations that are the result of only adding fill edges incident to a single vertex, or to create a minimal triangulation that is the result of adding three fill edges that define a cycle of length three. The difference between Lex M and MCS-M can be exemplified by two simple cycles of length five sharing a single vertex. If the vertex contained in both cycles are chosen to be the last in the elimination ordering, then the set of elimination orderings produced by Lex M and MCS-M are disjoint sets. Despite this, the same set of minimal triangulations is obtained by these sets of elimination orderings. A natural question emerges, which is also left as an open question in [4]: Does Lex M and MCS-M produce the same set of triangulations?

Paper III attends this problem, and shows that the two algorithms actually produce the same set of triangulations. The idea behind the proof is as follows. Each minimal triangulation can be obtained by a set of minimal elimination orderings, and any pair of orderings from this set are said to be equivalent. Paper III proves that an ordering can be obtained by MCS-M if and only if Lex M can produce an equivalent ordering from the same input graph.

3.4 (Paper IV [26]) Combining several new techniques into a faster minimal triangulation algorithm

Until the spring of 2004 it was not known whether or not minimal triangulations could be computed with better time bound than $O(nm)$, which is $O(n^3)$ for dense graphs. Attempts to improve the time complexity of known minimal triangulation algorithms below $O(n^3)$ mainly meet two obstacles. The first is to avoid searching the input graph $O(n)$ times, and the second is to compute the

set of new fill edges after each iteration. In most cases these fill edges can be obtained by completing a set of vertex sets into cliques. This problem can be restated as a binary multiplication problem, and thus solved in time $O(n^{2.376})$ [16]. Independently of each other, two research groups used this technique to find new and faster minimal triangulation algorithms for dense graphs.

Kratsch and Spinrad [33] designed an $O(n^{2.69})$ time algorithm, which is a new implementation of Lex M. It executes several steps of Lex M at once, and then uses binary matrix multiplication to update all the labels in one operation. Heggernes, Telle, and Villanger [26] designed an $o(n^{2.376})$ time algorithm, by using minimal separators and potential maximal cliques to partition the triangulation problem into subproblems, and used binary matrix multiplication to find the new subproblems. The second result is presented as Paper IV. A preliminary version of Paper IV appeared at ACM-SIAM Symposium on Discrete Algorithms (SODA) [27] in 2005.

From [31] and [12] it is known that minimal separators and potential maximal cliques can be used to separate the minimal triangulation problem into independent subproblems. The problem with this approach is to bound the depth of the recursion tree, since it might be $O(n)$ in the worst case. In the algorithm presented in Paper IV, the height of this recursion tree is $O(\log n)$. This is obtained by a partitioning algorithm that finds a set of non-crossing minimal separators, such that no subproblem contains more than some fraction of the non edges in the input graph. Let $O(n^\alpha)$ be the time bound of multiplying two $n \times n$ matrices. Currently the lowest value of α is $2.375 < \alpha < 2.376$ by the algorithm of Coppersmith and Winograd [16]. An $O(n^\alpha \log n)$ time algorithm for minimal triangulation can now be obtained by implementing the balanced partition algorithm to run in $O(n^2 - |E|)$ time for an input graph $G = (V, E)$, and then use matrix multiplication to complete the minimal separators into cliques.

Even though tree decompositions are not used directly in this algorithm, the algorithm can be considered as an algorithm that refines a tree decomposition, until it defines a minimal triangulation. Unlike the algorithm, one of the proof used to claim the time complexity heavily depends on tree decompositions. This makes tree decompositions to one of the basic structures used to obtain this result.

3.5 (Paper V [20]) Computing treewidth in exponential time using potential maximal cliques

Computing the treewidth of a graph is a problem that has received a lot of attention, but it is quite recent that exact exponential time algorithms for treewidth have been published. The first was an exact algorithm for treewidth and minimum fill-in by Fomin, Kratsch, and Todinca [19], presented at International Colloquium on Automata, Languages and Programming (ICALP) 2004. This algorithm requires every potential maximal clique of G as part of the input, and

computes the treewidth of G in time $O(n^3\Pi_G)$, where Π_G is the number of potential maximal cliques in G. Listing the potential maximal cliques of G is the most time consuming operation in [19], and thus the time complexity of the algorithm becomes $O^*(1.9601^n)$, which is the time required to list all potential maximal cliques of the input graph. Improving the time required for listing the potential maximal cliques would thus improve the time complexity of the algorithm, and this was also left as an open question in [19].

The problem of finding a better upper bound for the number of potential maximal cliques in a graph, and to list these more efficiently, is addressed in [44], which will be presented at Latin American Theoretical Informatics Symposium (LATIN) 2006. The $O^*(1.9601^n)$ time algorithm for listing all potential maximal cliques presented in [19], is improved to $O^*(1.8899^n)$, and we show that the number of potential maximal cliques contained in a graph is $O^*(1.8135^n)$. The second result will be the new running time if somebody manages to list all potential maximal cliques of a graph, with a polynomial delay for each potential maximal clique, i.e. listing all the potential maximal cliques in $O^*(\Pi_G)$ time. Such algorithms exist for listing minimal separators [5, 30], so it is an interesting open question if this type of algorithms exists for potential maximal cliques.

Paper V is obtained by combining the results of [19] and [44]. The time bound for algorithm in [19] is obtained by proving that every potential maximal clique can be defined by a set of $2n/5$ vertices. In [20] this is improved, such that any potential maximal clique can be defined by at most $n/3$ vertices, which improves the time bound for listing potential maximal cliques, and thus also for finding the treewidth, to $O^*(1.8899^n)$. One of the results of [13] is that all potential maximal cliques of a graph can be found by finding the *nice* potential maximal cliques, and then use these to generate the potential maximal cliques that are not nice. Another result in [13] is that any nice potential maximal clique can be defined by two crossing minimal separators. These separators and other observations are used in Paper V to partition the vertex set of the graph into three independent sets, such that any of these can be used to define the potential maximal clique.

References

[1] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database systems. *Journal of the Association for Computing Machinery*, 30:479–513, 1983.

[3] A. Berry. A wide-range efficient algorithm for minimal triangulation. In *SODA: Proceedings of the annual ACM-SIAM symposium on Discrete algorithms*, pages 860–861. Society for Industrial and Applied Mathematics, 1999.

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

[5] A. Berry, J.P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. *International Journal of Foundations of Computer Science*, 11(3):397–403, 2000.

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

[7] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for dynamically maintaining chordal graphs. In *ISAAC*, volume 2906 of *Lecture Notes in Computer Science*, pages 47 – 57. Springer Verlag, 2003.

[8] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for maintaining chordality. *Discrete Mathematics*, 306(3):318–336, 2006.

[9] 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.

[10] 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*, pages 1–30. Springer Verlag, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

[11] H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[12] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM Journal on Computing*, 31:212–232, 2001.

[13] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2):17–32, 2002.

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

[15] F.R.K. Chung and D. Mumford. Chordal completions of planar graphs. *Journal of Combinatorial Theory B*, 62(1):96–106, 1994.

[16] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[17] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

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

[19] F.V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 568–580. Springer Verlag, 2004.

[20] F.V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact (exponential) algorithms for treewidth and minimum fill-in. *Submitted to SIAM Journal on Computing*, 2005.

[21] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[22] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory B*, 16:47–56, 1974.

[23] J. Gustedt. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

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

[25] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 403–414. Springer Verlag, 2005.

[26] P. Heggernes, J.A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^{\alpha}logn) = o(n^{2.376})$. *SIAM Journal on Discrete Mathematics*, 19(4):900–913, 2005.

[27] P. Heggernes, J.A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^{\alpha}logn) = o(n^{2.376})$. In *SODA: Proceedings of the annual ACM-SIAM symposium on Discrete algorithms*, pages 907–916. Society for Industrial and Applied Mathematics, 2005.

[28] P. Heggernes and Y. Villanger. Efficient implementation of a minimal triangulation algorithm. In *ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 550–561. Springer Verlag, 2002.

[29] P. Heggernes and Y. Villanger. Simple and efficient modifications of elimination orderings. In *PARA*, volume 3732 of *Lecture Notes in Computer Science*, pages 788–797. Springer Verlag, 2004.

[30] T. Kloks and D. Kratsch. Listing all minimal separators of a graph. *SIAM Journal on Computing*, 27(3):605–613, 1998.

[31] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theoretical Computer Science*, 175:309–335, 1997.

[32] N. Korte and R.H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[33] D. Kratsch and J. Spinrad. Minimal fill in $o(n^{2.69})$ time. *Discrete Mathematics*, 306(3):366–371, 2006.

[34] C.G. Lekkerkerker and J.C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamentals of Mathematics*, 51:45–64, 1962.

[35] G.S. Lueker and K.S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the Association for Computing Machinery*, 26(2):183–195, 1979.

[36] T. Ohtsuki. A fast algorithm for finding an optimal ordering in the vertex elimination on a graph. *SIAM Journal on Computing*, 5:133–145, 1976.

[37] T. Ohtsuki, L.K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix. *Journal of Mathematical Analysis and Applications*, 54:622–633, 1976.

[38] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79:171–188, 1997.

[39] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3:119–130, 1961.

[40] N. Robertson and P. Seymour. Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[41] D. Rose, R.E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:146–160, 1976.

[42] 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, New York, 1972.

[43] 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 Journal on Computing*, 13:566–579, 1984.

[44] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 800–811. Springer Verlag, 2006.

[45] Y. Villanger. Lex M versus MCS-M. *Discrete Mathematics*, 306(3):393–400, 2006.

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

[47] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2:77–79, 1981.