REPORTS
IN
INFORMATICS

ISSN 0333-3590

Minimal Triangulations of Graphs

Pinar Heggernes

REPORT NO 264 January 2004

Department of Informatics
UNIVERSITY OF BERGEN
Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/ps/2004-264.ps

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Minimal Triangulations of Graphs

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

Abstract

Any given graph can be embedded in a chordal graph by adding edges, and the resulting chordal graph is called a triangulation of the input graph. In this paper we study minimal triangulations, which are the results of adding an inclusion minimal set of edges to produce a triangulation. This topic was first studied from the standpoint of sparse matrices and vertex elimination in graphs. Today we know that minimal triangulations are closely related to minimal separators of the input graph. Since the first papers presenting minimal triangulation algorithms appeared in 1976, several characterizations of minimal triangulations have been proved, and a variety of algorithms exist for computing minimal triangulations of both general and restricted graph classes. This survey presents and relates to each other these results in a unified modern notation, keeping an emphasis on the algorithms.

1 Introduction and history

Several important and widely studied problems on graphs are concerned with computing an embedding of an arbitrary graph into a chordal graph with various properties. Edges can be added to any given graph so that the resulting graph, called a *triangulation* of the input graph, is chordal, i.e., contains no induced chordless cycle. Many different triangulations exist for a given graph in general. Most of the related graph problems that arise from practical applications seek to minimize various graph parameters of a triangulation. For example, the *minimum triangulation* problem, also referred to as the *minimum fill-in* problem, asks to find a triangulation with the fewest number of edges, and it has applications in sparse matrix computations [75], database management [4, 76], knowledge based systems [58], and computer vision [25]. The *treewidth* problem asks to find a triangulation where the size of the largest clique is minimized, and many NP-complete problems are solvable in polynomial time when they are restricted to graphs with bounded treewidth [15, 73]. Unfortunately, both the minimum triangulation and the treewidth problems are NP-hard [2, 79]. However, for both problems, the polynomially computable alternative of finding a *minimal triangulation* is interesting, and this survey is devoted to minimal triangulations. A triangulation H of a given graph G is *minimal* if no triangulation of G is a proper subgraph of H.

Although we know today that minimal triangulations are closely related to minimal separation, sparse matrix computations was the first field to study different triangulations of a given graph [43, 71, 75]. Large sparse symmetric systems of equations arise in many areas of engineering, like the structural analysis of a car body, or the modeling of air flow around an airplane wing. The function to be computed can be often discretized as a mesh that covers the physical structure, where each point is connected to a few other points, and the related sparse matrix can simply be regarded as an adjacency matrix of this graph. Such systems are solved through standard methods of linear algebra, like Gaussian elimination with

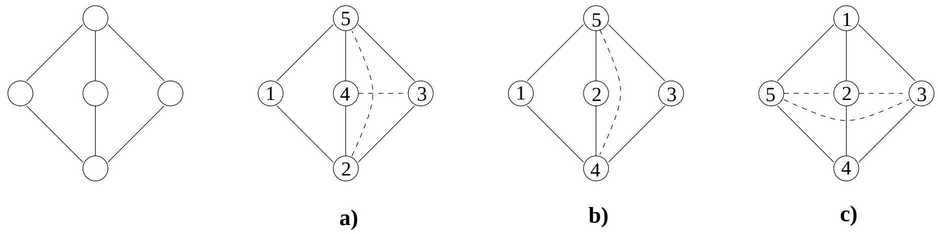

Figure 1: Three different orderings on the same graph are given, along with the resulting triangulations through Elimination Game: a) a non-minimal triangulation, b) a minimum triangulation, c) a minimal triangulation. The dashed edges represent fill edges of each triangulation.

proceeding backward and forward substitution. However, during the elimination process non-zero entries, called *fill*, are inserted into cells of the matrix that originally held zeros, which increases the time needed to perform the elimination, the storage requirements, and the time needed to solve the system after the elimination.

A graph algorithm, known as *Elimination Game* [71], was given in 1961, introducing the connection between sparse matrix computations and graphs. This algorithm simulates Gaussian elimination on graphs by repeatedly choosing a vertex v, adding edges to make the neighborhood of v into a clique, and then removing v from the graph. The edges that are added during Elimination Game correspond to the fill of Gaussian elimination, and they are called *fill edges*. The number of fill edges is heavily dependent on the order in which the vertices are processed. This ordering of the graph corresponds to the pivotal ordering of the rows and columns in Gaussian elimination[1]. An interesting connection is that the class of graphs produced by adding the fill edges of Elimination Game to the input graph is exactly the class of chordal graphs [40]. Thus Gaussian elimination and consequently Elimination Game correspond to computing triangulations of the given graph.

As mentioned above, computing a triangulation with few edges is important for efficient solution of sparse systems. Since the problem of computing minimum triangulations is NP-hard, the related polynomially solvable problem of computing minimal triangulations became interesting, and the first algorithms for it appeared in 1976 [68, 74]. The number of edges in a minimal triangulation can be far from minimum, as illustrated in Figure 1. Because of this there is a general belief that minimal triangulations are not interesting in practice for sparse matrix computations. However, on the contrary, minimal triangulations that contain low fill are indeed highly desirable for convenient storage of filled matrices. If the computed triangulation is minimal, then proceeding reorderings of it, which might be necessary for example for parallel computations, always result in the same triangulation of the input graph, and thus the allocated storage is not disturbed, as we will explain in more detail later. Although the first introduced algorithms for minimal triangulations do not consider the number of fill edges, more recent minimal triangulation algorithms are able to produce minimal triangulations with low fill [10, 11, 13, 29, 72]. Such algorithms are useful also for the treewidth problem [17].

First results and characterizations of minimal triangulations were given simultaneously by Ohtsuki, Cheung, and Fujisawa [68, 69], and Rose, Tarjan, and Lueker [74] in 1976. The algorithms of [68] and [74] both have time bound $O(nm)$, where n is the number of vertices and m is the number of edges of the input graph. Although

[1]For many real applications, the sparse matrix at hand has nice properties, like being symmetric and positive-definite, so that numerical stability is ensured regardless of the pivotal order, and pivoting can be performed with the sole goal of reducing fill.

several new algorithms have been introduced for computing minimal triangulations since then, this remained as the best known time bound for the problem until very recently [56]. The results presented in [68], [69], and [74] are strongly connected to vertex elimination. After these results, apart from a parallel minimal triangulation algorithm presented by Dahlhaus and Karpinski in 1989 [34], there was a time gap of almost 20 years until minimal triangulations began to be restudied. When new results started to appear in mid 1990's, the approaches to compute and characterize minimal triangulations were two-fold: those through vertex elimination, and those through minimal separators of the input graph.

In 1993, Kloks et al. indicated a strong connection between the minimal vertex separators of a graph and the solutions to the problems of minimum triangulation and treewidth [52, 53]. Proceeding research, motivated from this connection, gave new characterizations of minimal triangulations through minimal separators of the input graph. Parra and Scheffler [70], concluding from simultaneous results of Kloks, Kratsch, and Spinrad [55], showed that minimal triangulations can be computed by making into cliques a maximal set of non-crossing minimal separators of the input graph, thus giving the first algorithmic approaches to computing minimal triangulations that were completely independent of vertex elimination. The connection between minimum triangulations and minimal separators was concluded by Bouchitté and Todinca [19, 20] who showed that minimum triangulations can be computed in polynomial time for graphs having a polynomial number of minimal separators. At the same time, they gave a characterization of minimal triangulations through potential maximal cliques [19].

During this period, several new minimal triangulation algorithms, both on general graphs and on special graph classes, were introduced. In 1996, Blair, Heggernes, and Telle [13] posed and solved the problem of making a given triangulation minimal by removing edges. Their algorithm has time bound $O(f(m+f))$, where f is the number of fill edges in the given initial triangulation, and motivated from real applications, this algorithm performs faster than $O(nm)$ time algorithms when the initial fill is small. A year later, the same problem was also solved by Dahlhaus [29] in time $O(nm)$. In 2000, Berry [6, 10] introduced an algorithm that also solves this problem in $O(nm)$ time, and that can furthermore create any desired minimal triangulation of a given graph. In 2001 and 2003, two iterative refinement algorithms were given to solve the same problem by Peyton [72], and by Berry, Heggernes, and Simonet [11]. Although the theoretical time bounds of these algorithms are not analyzed, Peyton's algorithm is documented to run fast in practice. Finally the most recent minimal triangulation algorithms on general graphs of $O(nm)$ time with differing properties are given by Berry, Heggernes, and coauthors in [7] and [12]. For 27 years, $O(nm) = O(n^3)$ remained to be the best known time bound for minimal triangulations, and it was an open question until 2003 whether or not minimal triangulations could be computed in time $o(n^3)$. A breakthrough was made then by Kratsch and Spinrad [56], who gave a new implementation of LEX M which runs in time $O(n^{2.69})$.

Apart from the above mentioned results, algorithms were introduced with better time bounds for computing minimal triangulations either sequentially on special graph classes [21, 30, 33, 63], or in parallel on general graphs [34]. In addition to new algorithms, characterizations of some graph classes have been given through their minimal triangulations [21, 64, 70] during the same period.

The purpose of this paper is to present the mentioned results in a unified and modern terminology, and to relate various results to each other. It is not our intention to list *all* results on minimal triangulations, but the most important and characterizing ones, and our focus will be on algorithms. Our hope is that this survey will serve as an introduction to newcomers to the field, and as an inspiration for new results and algorithms to researchers who are already familiar with the field. As most of the algorithms we will describe are involved and long papers are devoted to them, we are of course not able to explain every algorithm in full detail. However, our goal is to give enough elements of each algorithm so that the interested reader will be able to decide which algorithms to study further through the given references. We will group the results on minimal triangulation of general graphs into two categories: those based on vertex elimination and those based on minimal separation. In order to give a good understanding of these issues, we will start with giving a solid background on chordal graphs, and giving characterizations of chordal graphs, both based on vertex elimination and based on minimal separators.

This paper is organized as follows. Preliminaries and notation are given in Section 2. Section 3 studies chordal graphs and their various characterizations. Minimal triangulations, their characterizations, and minimal triangulation algorithms for general graphs are studied in Chapters 4 and 5. Chapter 6 is devoted to the problem of making a given triangulation minimal, whereas Chapter 7 is about minimal triangulations of restricted graph classes. Several other related problems are mentioned together with concluding remarks in Chapter 8. In order to highlight various characterizations of minimal triangulations, we will use environment indicator **Characterization** only for this purpose.

2 Preliminaries and first characterizations of minimal triangulations

We consider simple and connected input graphs. A graph is denoted by $G = (V, E)$, with $n = |V|$, and $m = |E|$. For a set $A \subseteq V$, $G(A)$ denotes the subgraph of G induced by the vertices in A. A is called a *clique* if $G(A)$ is complete. The process of adding edges to G between the vertices of $A \subseteq V$ so that A becomes a clique in the resulting graph is called *saturating* A. The *neighborhood* of a vertex v in G is $N_G(v) = \{u \mid uv \in E\}$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. Similarly, for a set $A \subseteq V$, $N_G(A) = \cup_{v \in A} N_G(v) \setminus A$, and $N_G[A] = N_G(A) \cup A$. A vertex v is called *simplicial* in G if $N_G(v)$ is a clique. The *deficiency* of vertex v in G is $D_G(v) = \{ux \mid u, x \in N_G(v) \text{ and } ux \notin E\}$, i.e., the set of edges that must be added in order to saturate $N_G(v)$; hence $D_G(v)$ is empty if v is simplicial. A vertex v is called *universal* in G if $N_G(v) = V \setminus \{v\}$. When graph G is clear from the context, we will omit subscript G. In this paper, we distinguish between subgraphs and induced subgraphs. To denote that G is a subgraph of H on the same vertex set, we will use $G \subseteq H$.

An *ordering* of G is a function $\alpha : V \leftrightarrow \{1, 2, ..., n\}$. We will sometimes call $\alpha(v)$ the *number* of v according to α. We will also use the informal notation $\alpha = \{v_1, v_2, ..., v_n\}$ to denote that $\alpha(v_i) = i, 1 \leq i \leq n$.

Vertex separators are central in minimal triangulations. A vertex set $S \subset V$ is a *separator* if $G(V \setminus S)$ is disconnected. Given two vertices u and v, S is a u,v-*separator* if u and v belong to different connected components of $G(V \setminus S)$, and S is then said to *separate* u and v. Two separators S and T 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 $x, y \in S$ [55, 70]. A u,v-separator S is *minimal* if no proper subset of S separates u and v. In general, S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u,v-separator. It can be easily verified that S is a minimal separator if and only if $G(V \setminus S)$ has two distinct connected components C_1 and C_2 such that $N_G(C_1) = N_G(C_2) = S$. In this case, C_1 and C_2 are called *full components*, and S is a minimal u,v-separator for *every* pair of vertices $u \in C_1$ and $v \in C_2$. When a separator, or a minimal separator, is a clique, we will call it a *clique separator*, or a *clique minimal separator*, respectively.

A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. A graph is *chordal*, or equivalently *triangulated*, if it contains no induced chordless cycle of length ≥ 4. Consequently, all induced subgraphs of a chordal graph are also chordal. A graph $H = (V, E \cup F)$ is called a *triangulation* of $G = (V, E)$ if H is chordal. In order to distinguish the edges that are added to G to obtain H, we will require that $E \cap F = \emptyset$. The edges in F are called *fill edges*. Thus every edge in a triangulation is either an edge of the underlying original graph, or a fill edge. H is a *minimal triangulation* if $(V, E \cup F')$ is non-chordal for every proper subset F' of F.

The following characterization gives a valuable and important property of minimal triangulations, which is central to several minimal triangulation algorithms.

Characterization 2.1 (Rose, Tarjan, and Lueker [74]) *A triangulation H is minimal if and only if the removal of any single fill edge from H results in a non-chordal graph.*

It is important to note that this is a special virtue of minimal triangulations; the same kind of result does not necessarily apply to every minimal completion of a graph into another graph class. For a given class $\mathcal{C}$ of graphs, we can say that H is a $\mathcal{C}$-completion of G if $G \subseteq H$ and H belongs to $\mathcal{C}$, and that H is a *minimal $\mathcal{C}$-completion* if no proper subgraph of H is a $\mathcal{C}$-completion of G. In this general setting, it might happen that removing any single fill edge from H results in a graph outside of $\mathcal{C}$, whereas removing several fill edges gives again a subgraph of H that is a $\mathcal{C}$-completion of G. The case when $\mathcal{C}$ is the class of interval graphs is an example to this. The following characterization is a useful consequence of Characterization 2.1, and the proofs of correctness of several minimal triangulation algorithms are based on it.

Characterization 2.2 (Rose, Tarjan, and Lueker [74]) *A triangulation H is minimal if and only if every fill edge is the unique chord of a 4-cycle in H.*

3 Characterizations of chordal graphs

Chordal graphs were introduced several years[2] before graph theory results related to sparse matrix computations began to appear. The definition of chordal graphs is independent of vertex elimination, and graphs of this class can be characterized by their minimal separators, as shown by Dirac in 1961 [37]. On the other hand, they coincide with the class of graphs resulting from Elimination Game, and they can also be characterized through vertex orderings, as shown by Fulkerson and Gross in 1965 [40]. We will keep these two characterizations of chordal graphs central to this paper. As a result, we will group the presented results on chordal graphs around these two characterizations in two subsections concerning, respectively, vertex elimination and minimal separators. Our proceeding discussion on minimal triangulations will then follow these two tracks.

In general, the class of chordal graphs can be thought of as an extension of trees, as they are the intersection graphs of subtrees of a tree [22, 42, 77]. The relation between chordal graphs and trees is closely connected to minimal separators, as the minimal separators of a chordal graph serve as articulation points, analogous to the internal vertices of a tree. Therefore, this relation will be a part of the discussion in the minimal separators track.

As it is the case for many early results of a field, the first characterizations of chordal graphs in each following subsection are not hard to deduce from the given

[2]For example, in 1958 Hajnal and Surányi proved in [47] one direction of Dirac's later characterization of chordal graphs (Theorem 3.3).

definitions and other related results. More information on chordal graphs can be found in [45].

3.1 Chordal graphs and their relation to vertex elimination

Already before the connection between chordal graphs and vertex elimination was known, a graph analogy of Gaussian elimination was first given by Parter in 1961 [71] through an algorithm called *Elimination Game*, which is shown in Figure 2.

Algorithm Elimination Game
Input: A graph $G = (V, E)$ and an ordering $\alpha = \{v_1, ..., v_n\}$ of V.
Output: The filled graph G^+_α.
begin
  $G^0 = G$;
  for $i = 1$ **to** n **do**
    Let $F^i = D_{G^{i-1}}(v_i)$;
    Obtain G^i by adding the edges in F^i to G^{i-1} and removing v_i;
  $G^+_\alpha = (V, E \cup \bigcup_{i=1}^n F^i)$;
end

Figure 2: Elimination Game algorithm.

An *elimination ordering* α is simply an ordering that is used as input to Elimination Game to create G^+_α. If no fill is created during Elimination Game with input G and α, i.e., a simplicial vertex of the remaining graph is removed at each step, then $G^+_\alpha = G$, and α is called a *perfect elimination ordering (peo)* of G. In 1965 the following characterization of chordal graphs based on vertex elimination was given by Fulkerson and Gross [40].

Theorem 3.1 (Fulkerson and Gross [40]) *A graph is chordal if and only if it has a peo.*

Observe that α is a peo of G^+_α, and thus G^+_α is a triangulation of G by Theorem 3.1. As a consequence, deciding whether a graph is chordal can be done by repeatedly removing a simplicial vertex until no simplicial vertices are left in the remaining non-empty graph (in which case the graph is *not* chordal), or the graph becomes empty (concluding that the graph is chordal). In the latter case, the order in which the vertices are removed is a peo of the input graph. This first algorithm for chordal graph recognition by Fulkerson and Gross [40] is often referred to as *the simplicial elimination scheme*. Later, more efficient algorithms for recognizing chordal graphs and computing peos were introduced. Given a chordal graph G, a peo of G can be computed in $O(n + m)$ time by Algorithms MCS (Maximum Cardinality Search) [76] and Lex-BFS (Lexicographic Breadth First Search) [74]. These algorithms rely on the fact that any chordal graph that is not complete, has at least two non-adjacent simplicial vertices [37], thus, since chordality is a hereditary property, a peo of a chordal graph can choose to order any vertex, or any maximal clique, last. Algorithm MCS starts by assigning an arbitrary vertex of G the last position in α. Every vertex keeps a weight equal to the number of its already processed neighbors, and a vertex of largest such number is chosen at each step i to be placed in position $n-i$ in α. Lex-BFS appeared before MCS, and has a similar description but uses labels that are lists of the already processed neighbors, instead of using weights. In order to decide whether a given graph G is chordal, Elimination Game is run with input G and α, where α is an ordering returned by MCS or Lex-BFS on G. Then G is chordal if and only if no fill is introduced. Given G and α, G^+_α can be computed in time $O(n + m')$ by a clever implementation of

Elimination Game due to Tarjan and Yannakakis [76], where m' is the number of edges in G_α^+.

Thus chordal graphs can be recognized in linear time, which is important for practical applications, since no fill is generated during the elimination process if a peo of the graph is used as the pivotal order. When the input graph is not chordal one seeks to compute an ordering that results in few fill edges. A *minimum elimination ordering* is defined to be an ordering α that gives the smallest number of fill edges in G_α^+. Computing a minimum elimination ordering is equivalent to computing a minimum triangulation, and thus NP-hard [79]. An ordering α is called a *minimal elimination ordering (meo)* if G_α^+ is a minimal triangulation of G.

Elimination Game can also be implemented so that α is not a part of input, but is generated during the course of the algorithm. In this case, we can at each step i choose a vertex v of G^{i-1} according to any desired criteria, and set $\alpha(v) = i$, to define an elimination ordering α. One famous and widely used heuristic for the minimum triangulation problem is called *Minimum Degree*, and it chooses a vertex v of minimum degree in G^{i-1} at each step i. Implementations of this type cannot use the ideas of [76], and they usually have a theoretic $O(n^3)$ time bound, although very fast practical implementations of Minimum Degree and other heuristics exist.

The following characterization of fill edges of G_α^+ follows directly from the description of Elimination Game.

Theorem 3.2 (Rose, Tarjan, and Lueker [74]) *Given a graph $G = (V, E)$ and an elimination ordering α of G, uv is an edge in G_α^+ if and only if $uv \in E$ or there exists a path $u, x_1, x_2, ..., x_k, v$ in G where $\alpha(x_i) < \min\{\alpha(u), \alpha(v)\}$, for $1 \le i \le k$.*

An important consequence of Theorem 3.2 is the following. Let v_i be the vertex eliminated at step i of Elimination Game, and let G^i be the resulting elimination graph after this step. The vertices that are eliminated before v_i can be reordered in any way without affecting the fill edges that appear in G^i. Thus, given $\alpha = \{v_1, v_2, ..., v_n\}$ and $1 < i < n$, the relative local ordering of vertices $\{v_1, v_2, ..., v_i\}$ has no effect on G^i. In particular, for two non-adjacent vertices u and v of G, where $u, v \in \{v_{i+1}, v_{i+2}, ..., v_n\}$, uv is a fill edge in G^i if and only if u and v are in the neighborhood of a common connected component of $G(\{v_1, v_2, ..., v_i\})$. This observation is central in several minimal triangulation algorithms.

As a final remark of this subsection, note that there might exist triangulations of a given graph $G = (V, E)$ that cannot be generated by Elimination Game. For example, the complete graph on vertex set V is a triangulation of G, but no elimination ordering can generate it unless G has a universal vertex.

3.2 Chordal graphs and their relation to minimal separation

The following famous characterization of chordal graphs by their minimal separators was given by Dirac in 1961, and it appeared simultaneously with the first connections between graphs and sparse matrices, although it is completely independent of vertex elimination.

Theorem 3.3 (Dirac [37]) *A graph G is chordal if and only if every minimal separator of G is a clique.*

A straight forward algorithm for computing minimal triangulations can thus be deduced directly from Dirac's characterization, as an alternative to Elimination Game. Algorithm SMS (Saturate Minimal Separators), given in Figure 3, computes a triangulation H of a non-chordal input graph G by Theorem 3.3. Constructing an example to show that Algorithm SMS, like Elimination Game, is not able to compute every triangulation of a given graph, is an easy exercise. In fact, as we will see in the next session, SMS actually computes exactly the minimal triangulations of the input graph!

Algorithm Saturate Minimal Separators (SMS)
Input: A graph $G = (V, E)$.
Output: A triangulation H of G.
$H = G$;
repeat
  Let S be a minimal separator of H that is not a clique;
  Add fill edges to H to saturate S;
until all minimal separators of H are cliques

Figure 3: Algorithm SMS.

Using Theorem 3.3 and the definitions of minimal separators and chordal graphs, it can be shown that every minimal separator of a chordal graph is contained in the neighborhood of a vertex. Hence the following contemporary characterization of chordal graphs by Lekkerkerker and Boland is equivalent to the previous one by Dirac, and furthermore it provides a convenient way of finding the minimal separators to be saturated.

Theorem 3.4 (Lekkerkerker and Boland [59]) *A graph G is chordal if and only if every vertex v in G has the following property: each minimal separator $S \subseteq N(v)$ of G a clique.*

Note that for any graph G, the minimal separators that are subsets of $N_G(v)$ for a given vertex v are exactly the sets $N_G(C)$ for each connected component C of $G(V \setminus N[v])$.

Theorem 3.4 gives a property that each vertex of a chordal graph must satisfy, and that can be checked for all vertices simultaneously or one by one. In a similar way, the following is a much more recent characterization of chordal graphs by a property that every edge must satisfy, also related to minimal separators.

Theorem 3.5 (Berry, Heggernes, and Villanger [12]) *A graph $G = (V, E)$ is chordal if and only if every edge uv in G has the following property: $N(u) \cap N(v)$ is a minimal u, v-separator in $(V, E \setminus \{uv\})$.*

As a consequence, there cannot exist chordless paths with 3 or more edges between u and v in $(V, E \setminus \{uv\})$. A pair of vertices with this property in any graph is called a *2-pair* [1].

We conclude this section with an important connection between chordal graphs and trees, which is also closely related to minimal separators. Chordal graphs are exactly the intersection graphs of subtrees of a tree [22, 42, 77], and the following result gives a very useful tool which is used in several minimal triangulation algorithms.

Theorem 3.6 (Buneman [22], Gavril [42], Walter [77]) *A graph G is chordal if and only if there exists a tree T, whose vertex set is the set of maximal cliques of G, that satisfies the following property: for every vertex v in G, the set of maximal cliques containing v induces a connected subtree of T.*

Such a tree is called a *clique tree*, and it can be computed in linear time [14]. A chordal graph has at most n maximal cliques [37], and thus a clique tree has $O(n)$ vertices and edges. We refer to the vertices of T as *tree nodes* to distinguish them from the vertices of G. Each tree node of T is thus a maximal clique of G. In addition, it is customary to let each edge K_iK_j of T hold the vertices of $K_i \cap K_j$, where K_i and K_j are maximal cliques in G. Thus edges of T are also vertex sets. Although a chordal graph can have many different clique trees, these all share the following important property regarding minimal separators.

Theorem 3.7 (Buneman [22], Ho and Lee [49]) *Let T be any clique tree of a chordal graph G. Every edge of T is a minimal separator of G, and for every minimal separator S in G, there is an edge $K_iK_j = K_i \cap K_j = S$ in T.*

Now that we have given the necessary background on chordal graphs and their characterizations, we move on to our essential topic: minimal triangulations.

4 Minimal triangulations through minimal elimination orderings

In this section we will focus on the relationship between minimal triangulations and minimal elimination orderings, and mention minimal triangulation algorithms that are based on Elimination Game and the characterization of chordal graphs by Fulkerson and Gross.

Minimal elimination orderings were first defined in [69] and [74]. An elimination ordering α of G is *minimal* if G^+_α is a minimal triangulation of G. Since minimal triangulations are defined independently of vertex elimination, one can wonder whether there are minimal triangulations that cannot be generated by Elimination Game. The following result answers this question.

Characterization 4.1 (Ohtsuki, Cheung, and Fujisawa [69]) *H is a minimal triangulation of G if and only if there exists a minimal elimination ordering α on G such that $H = G^+_\alpha$.*

Thus any minimal triangulation of an input graph G can be generated by Elimination Game, or equivalently, by a minimal elimination ordering. Indeed, if H is a minimal triangulation of G, then $H^+_\alpha = G^+_\alpha = H$ for *every* peo α of H [13]. Consequently, the problems of computing a minimal triangulation and computing a minimal elimination ordering are equivalent.

The first algorithms for computing minimal triangulations were given in 1976 in independent works of Rose, Tarjan, and Lueker [74], Ohtsuki, Cheung, and Fujisawa [69], and Ohtsuki [68]. These algorithms all compute a minimal elimination ordering of the input graph.

A minimal elimination ordering cannot start with an arbitrary vertex. In [69] the authors characterize the vertices that can be the first in a meo through what we today call an *OCF-vertex* in the following definition (OCF stands for the initials of the authors of [69]).

Definition 4.2 *A vertex v in G is an* OCF-vertex *if, for each pair of non-adjacent vertices $x, y \in N_G(v)$, there is a component C in $G(V \setminus N_G[v])$ such that $x, y \in N_G(C)$.*

Note that, in the above definition, $N_G(C)$ is a minimal separator of G, and thus if an OCF-vertex is chosen at each step of Elimination Game, only fill edges within non-crossing minimal separators are introduced. With the knowledge that we have today about the role that minimal separators play in minimal triangulations, we can conclude that the described procedure will lead to a minimal triangulation. The authors of [69] were able to prove this without the use of minimal separators already in 1976:

Theorem 4.3 (Ohtsuki, Cheung, and Fujisawa [69]) *A minimal elimination ordering α is computed by choosing, at each step i of Elimination Game, an OCF-vertex v in G^{i-1}, and assigning $\alpha(v) = i$.*

Every graph has an OCF vertex, and such a vertex can be found by Lex-BFS [8] or MCS [7]. Thus Theorem 4.3 gives a straight forward algorithm for computing a minimal triangulation, however its running time is not optimal. In [68] Ohtsuki gave an $O(nm)$ time algorithm for the same purpose. Simultaneously, an $O(nm)$ algorithm that is easier to understand and implement was given in [74] by Rose, Tarjan and Lueker. This algorithm is called LEX M, and it is an extension of Lex-BFS that uses Theorem 3.2 to compute a minimal triangulation. Both algorithms use an observation by Rose [75] that for any graph G and any clique K in G, there exists a minimal elimination ordering of G where vertices of K are numbered last, i.e. with numbers $n-|K|+1, n-|K|+2, ..., n$. Thus, as opposed to first vertex of a meo, last vertex of a meo can be chosen arbitrarily. For both LEX M and Ohtsuki's algorithm, the input is simply $G = (V, E)$, and the output is an ordering α such that G^+_α is a minimal triangulation of G. Neither of these algorithms consider the number of resulting fill edges, and thus the produced triangulations are often far from minimum.

LEX M [74]. In this algorithm every vertex has a label list, and each list consists of distinct integers between 2 and n. In the beginning all label lists are empty. LEX M starts by assigning number n to an arbitrary vertex w and adding integer n to the label lists of all neighbors of w. Then, at each step $n-i+1$, an unnumbered vertex v of lexicographically highest label is chosen, v receives number i, and LEX M adds integer i to the end of the label lists of all unnumbered vertices u such that there is a path in G between v and u consisting only of unnumbered vertices with lexicographically smaller labels than those of v and u. We will call such a path a *fill path*. Thus vertex v appends its number $\alpha(v)$ to the label of every vertex u which is connected to v through a fill path. Edge vu is then an edge of the resulting minimal triangulation, hence this algorithm can add the resulting fill edges and compute G^+_α at the same time as α. However, the algorithm uses only the information from input graph G and the vertex labels during the whole process, so the added fill edges have no effect on the execution.

The correctness of LEX M can be proved through observing that once a fill path p is established between two non-adjacent vertices u and v, p will stay a fill path throughout the algorithm. Thus the vertices on p will receive smaller numbers than those of u and v, and this will result in fill edge uv no matter when u and v are processed or which numbers they receive. On the other hand, at the time p is established, since u and v already have non-empty labels, there must exist yet another path q in G between u and v, where all vertices of q are already processed and have received their numbers. Consequently, since an unnumbered vertex of highest label is chosen at each step, with some argumentation we can reach the conclusion that there exists a vertex z on q and a vertex x on p such that u, x, v, z, u is a 4-cycle in G^+_α of which uv is the unique chord.

We have already mentioned that only the original graph edges are used in an execution of LEX M. Still, the $O(nm)$ time bound does not follow immediately. Since the algorithm has n main steps, and $O(n + m)$ time is needed at each step to follow all fill paths from the processed vertex, the only obstacle in achieving the $O(nm)$ time bound is the maintenance of the labels. Fortunately, the label lists can be avoided. To see this note that a LEX M ordering is a breadth first search ordering of the input graph, since every vertex u gets its label changed from empty to non-empty when a neighbor $v \in N_G(u)$ receives its number. Accordingly, vertices of G are partitioned into levels depending on their distances to vertex v_n in the resulting ordering. In addition, each level is further partitioned into groups such that vertices belonging to the same group have the same LEX M labels. At each step the partition is refined by creating a new level, and subdividing each existing group into two new groups: those vertices that have direct edges or fill paths to v and those that do not. As a consequence, maintaining and comparing lexicographic labels can be avoided at the cost of an extra $O(n)$ sort at each step, which fits well within the overall $O(nm)$ time bound.

Through the years, LEX M has given inspiration to other minimal triangulation algorithms that have either used it or enhanced it, and its running time of $O(nm)$ remained unbeaten until 2003. It is probably the most famous and widely referenced minimal triangulation algorithm. Recently a simplification of LEX M with the same time bound but that avoids the extra sorting step, called MCS-M, was given by Berry, Blair, and Heggernes in [7]. Even more recently, Kratsch and Spinrad [56] were able to give an $O(n^{2.69})$ time implementation of LEX M. We will now describe these enhancements to LEX M before we continue to Ohtsuki's minimal triangulation algorithm.

MCS-M [7]. This algorithm proceeds in the same way as LEX M, where each processed vertex increases the labels of unnumbered vertices to which it is connected through fill paths. However, instead of using lexicographic label lists, MCS-M uses simply cardinality labels, and a vertex of highest label is chosen at each step. Thus, MCS-M is an extension of MCS in the same way as LEX M is an extension of Lex-BFS, and MCS-M is a simplification of LEX M in the same way as MCS is a simplification of Lex-BFS. The proof of correctness described above for LEX M is actually the proof of correctness for MCS-M given in [7]. Note however that the two algorithms are not equivalent, as each of them can generate minimal elimination orderings that cannot be computed by the other.

Minimal triangulation in $O(n^{2.69})$ time [56]. This implementation of LEX M relies heavily on the idea of partitioning the vertex sets into groups that contain vertices with the same labels, which was described above. Kratsch and Spinrad [56] use matrix multiplication to compute the fill edges that result from this partition whenever a new group is created. Let $V_0, V_1, ..., V_k$ be a partition of the vertices of the input graph $G = (V, E)$ at some step of LEX M, such that V_0 are the vertices that have not yet been labeled, V_1 are the vertices with the smallest labels, and V_k are the vertices with the largest labels. By the results of [74], we know that vertices of V_i will appear earlier than the vertices of V_{i+1} in the resulting elimination ordering, for $0 \leq i < k$. For any V_i with $0 \leq i \leq k$, in order to compute the fill edges that appear between pairs of vertices in $V^+ = \bigcup_{j=i}^{k} V_j$ due to the elimination of vertices in $V^- = \bigcup_{j=0}^{i-1} V_j$, one can create the following matrix[3] M. For each connected component C of $G(V^-)$ there is a row in M, and for each vertex v of V^+ there is a column in M, such that $M[C, v]$ is nonzero if and only if v has a neighbor in C. Now, by Theorem 3.2 and the discussion that followed it, for $u, v \in V^+$, uv is a fill edge if $M^T M[u, v]$ is nonzero (in which case u and v both have neighbors in a common component C), and $M^T M$ gives all fill edges appearing between pairs of vertices in V^+ that are due to elimination of vertices in V^-.

In the implementation of [56], the above process is repeated every time a new group V_i is created, and the fill edges that result between vertices of V_i and vertices of V^+ are added to the filled graph. However, the computation of M is done using only the edges of the input graph G. The added fill edges are used to split groups and create new groups at later steps of the algorithm in the same way as original

[3]Since a complete ordering is not established yet, the ideas from [76] cannot be used for efficient computation of this partial fill.

LEX M. By carefully selecting a function of n to define a bound for when a group is considered "large", and using this information in the splitting process, the authors of [56] are able to bound the number of groups created, so that the resulting total running time is $O(n^{2.69})$ if the matrix multiplication algorithm of [26] is used.

Ohtsuki's algorithm [68]. Ohtsuki's algorithm from 1976 uses similar principles as LEX M, but orders vertex subsets rather than single vertices. Two vertex subset properties are central for this algorithm.

A set $P \subset V$ satisfies *Property A* if the following holds for each connected component C of $G(V \setminus P)$: For each pair of vertices $u, v \in Adj_G(C)$, u and v are connected in $G(V \setminus C \setminus Adj_G(C) \cup \{u, v\})$.

Two disjoint vertex sets $P \subset V$ and $Q \subset V$ satisfy *Property B* if Q is a clique and every vertex of Q is adjacent to every vertex of $Adj_G(Q) \cap P$.

Ohtsuki first proves that if P satisfies Property A, then the vertices of P can be numbered last in a minimal elimination ordering. His algorithm computes a partition of V into $V_1, V_2, ..., V_k$ such that each V_k is a clique and the following two properties are satisfied:

1. $\bigcup_{j=1}^{i} V_j$ satisfies property A for $i = 1, 2, ..., k-1$.
2. $\bigcup_{j=1}^{i-1} V_j$ and V_i satisfy Property B for $i = 2, ..., k$.

Then, the vertices of V_k are numbered last, the vertices of V_{k-1} are numbered next, and so on, until the vertices of V_1 receive the smallest numbers in the computed ordering α. It can be verified easily that the local numbering within each V_i is irrelevant to the produced minimal triangulation. The correctness of Ohtsuki's algorithm follows from the observations about Properties A and B mentioned above. The obstacle is to compute the vertex partition $V_1, V_2, ..., V_k$, and a quite involved search procedure is described in [68] for this reason.

Parallel minimal triangulation [34]. The only parallel algorithm for minimal triangulation of general graphs was presented by Dahlhaus and Karpinski [34], and appeared first in 1989, more than a decade after the presentation of LEX M and Ohtsuki's algorithm, and several years before the next results on new characterizations of and new algorithms on minimal triangulations. Also this parallel algorithm is based on minimal elimination orderings, and it generates a meo of a given graph in $O(\log^3 n)$ parallel time and $O(nm)$ processors on a CREW PRAM. Given a general graph G, assume that one has been able to decide a set V^+ of vertices that can be eliminated last in a meo. The fill that appears between vertices of V^+ is partly dependent on the connected components of $G(V \setminus V^+)$ and the neighborhoods of these components contained in V^+, as we have discussed earlier. The algorithm of [34] computes a meo of each connected component of $G(V \setminus V^+)$ in parallel, and uses this information to compute a meo for the whole input graph. The authors also show how to find a good set V^+. The running time of this parallel algorithm is dependent on another result by Dahlhaus, Karpinski, and Novick who showed that the clique separator decomposition problem belongs to NC [35]. The reader might also be interested to know that there exist parallel algorithms to recognize chordal graphs and to compute perfect elimination orderings of chordal graphs [24, 38, 28, 65].

We end this section with a short discussion on why minimal triangulations are desirable in general for sparse matrix computations in practice. Similar discussions are given in [13] and [72]. The first step in these computations is to find a good ordering α of G and compute G^+_α, before continuing with the numerical calculations in which the data structure of G^+_α is used to store the numerical values. Since these systems are often very large and one is usually interested in solving the same system with many different right hand side vectors, it is important that the data structure used for G_α^+ allows fast access. Therefore it is customary to use *static* data structures to store G_α^+ in practice. Sometimes another peo β of G_α^+ is computed then before the numerical computations in order to achieve some other properties, like better parallelism, through G_β^+ without increasing the size of fill [60, 61]. Certainly, $G_\beta^+ \subseteq G_\alpha^+$. However, if α is not a meo, or equivalently if G_α^+ is not a minimal triangulation of G, then G_β^+ might be a proper subgraph of G_α^+, which makes it difficult to operate on the static data structure that was initially set up for G_α^+. For this reason α is preferred to be a minimal elimination ordering.

5 Minimal triangulations through minimal separators

In this section we will study the relationship between minimal triangulations and minimal separation, and algorithms that generate minimal triangulations that are based on characterizations of chordal graphs by their minimal separators.

Remember Algorithm SMS from Section 3. In this section, we will mention and explain the results which lead to the conclusion that this algorithm generates a minimal triangulation. Our understanding of minimal triangulations today is tightly coupled to minimal separators. By the following result of Rose [75] it was clear already in 1970's that finding a correct set of minimal separators to saturate leads to a minimal triangulation.

Lemma 5.1 (Rose [75]) *Let H be a minimal triangulation of G. Any minimal separator of H is a minimal separator of G.*

Observe that a minimum triangulation is also a minimal triangulation. Consequently, there is a set $\mathcal{S}$ of minimal separators of G such that a minimum triangulation of G is obtained by saturating each minimal separator in $\mathcal{S}$. Unfortunately, as we have already mentioned, finding this set is NP-hard on general graphs. However, for graphs that have a polynomial number of minimal separators, it can be done in polynomial time [19]. When it comes to minimal triangulations, it was discovered in the 1990's that saturating *any* maximal set of non-crossing minimal separators results in a minimal triangulation, as we will shortly come back to.

Several years before the relationship between minimal triangulations and minimal separators was fully discovered, there appeared characterizations of minimal triangulations based on minimal separators.

Characterization 5.2 (Ohtsuki, Cheung, and Fujisawa [69]) *A triangulation H of G is minimal if and only if, for each fill edge uv, there exists no u,v-separator of G that is a clique in H.*

Hence, a triangulation is minimal if and only if no fill edges are added across an original clique separator or an already saturated separator during the minimal triangulation process. Now we know that every fill edge in a minimal separator is added within a minimal separator and no fill edges are added in two crossing separators. The full connection between minimal separators and minimal triangulations was partially discovered by several researchers simultaneously. The following property gathers the related results by Berry [5], Bouchitté and Todinca [19], Kloks, Kratsch, and Spinrad [55], and Parra and Scheffler [70].

Property 5.3 ([5], [19], [55], [70]) *Given a graph G, let $\mathcal{S}$ be an arbitrary set of pairwise non-crossing minimal separators of G. Obtain a graph G' by saturating each separator in $\mathcal{S}$.*

a) A clique minimal separator of G does not cross any minimal separator of G.

b) $\mathcal{S}$ is a set of clique minimal separators of G'.

c) Any clique minimal separator of G is a minimal separator of G'.

d) Any minimal separator of G' is a minimal separator of G.

e) $G(V \setminus S)$ and $G'(V \setminus S)$ have the same set of connected components for each minimal separator S of G'.

f) $G(V \setminus S)$ and $G'(V \setminus S)$ have the same set of full components for each minimal separator S of G'.

g) Any set of pairwise non-crossing minimal separators of G' is a set of pairwise non-crossing minimal separators of G.

h) Any minimal triangulation of G' is a minimal triangulation of G.

i) If $\mathcal{S}$ is a maximal set of pairwise non-crossing minimal separators of G then G' is a minimal triangulation of G.

From these results it is now obvious that Algorithm SMS creates a minimal triangulation. A general graph might have an exponential number of minimal separators. However, when a minimal separator S is saturated all of the minimal separators that cross S disappear, as the vertices of S cannot be separated from each other any more. A chordal graph has $O(n)$ minimal separators [75]. Thus, by the above results, Algorithm SMS stops within $O(n)$ iterations. Furthermore, at each step, instead of choosing a single minimal separator, a set of non-crossing minimal separators can be chosen that can be saturated simultaneously, or a maximal set of minimal separators can be computed at once.

At least as interesting is the following result which shows that no other kind of minimal triangulations exist.

Characterization 5.4 (Parra and Scheffler [70]) *H is a minimal triangulation of G if and only if H is the result of saturating a maximal set of pairwise non-crossing minimal separators of G.*

Thus the fill edges added by the algorithms that we have seen in the previous subsection correspond indeed exactly to the saturation of a maximal set of non-crossing minimal separators, and the correctness of these algorithms can also be proved through Characterization 5.4.

The difficulty in implementing Algorithm SMS efficiently is in finding the minimal separators. Lekkerkerker and Boland's Characterization 3.4 gives a straight forward way of computing and saturating the minimal separators, and Algorithm LB-Triang by Berry et al [6, 10] for computing a minimal triangulation is based on this characterization.

LB-Triang [10]. This algorithm processes the vertices of G in an arbitrary order which can be given as input or created during the execution of the algorithm. Fill edges that are computed are added to G and stored in a transitory graph H during the computation. At each of the n steps, a vertex v is chosen, and the minimal separators contained in $N_H(v)$ are saturated, following the characterization of chordal graphs by Lekkerkerker and Boland. It is shown in [10] that no fill edges are ever added to already processed vertices, thus each vertex can actually be removed from the graph after saturating the minimal separators in its neighborhood. Consequently, LB-Triang is similar to Elimination Game, both in the aspect that vertices can be eliminated in any order, and in the aspect that fill edges are added between the neighbors of the chosen vertex for elimination. However, the edges added by LB-Triang at each step is a subset of the edges added by Elimination Game at the same step, which is the reason why LB-Triang results in a minimal triangulation while Elimination Game does not. Also, it is important to note that the ordering α in which LB-Triang processes the vertices of G to produce a minimal triangulation H is *not* necessarily a peo of H, or equivalently, H is not necessarily G^+_α. A remarkable property of LB-Triang is that it is able to generate every minimal triangulation of the input graph, depending on the vertex ordering.

The correctness of LB-Triang follows from Theorem 3.4, Property 5.3, and by showing that the minimal separators computed at each step do not cross any of the already saturated separators of previous steps. Regarding the running time, the authors show that the search for connected components and minimal separators contained in $N_H(v)$ can be done in G rather than in H, and thus requires $O(m)$ time at each of the n steps. In order to achieve an $O(nm)$ time bound, the edges of H cannot be computed and stored explicitly as one risks adding the same edge several times. Thus computing $N_H[v]$ in $O(m)$ time is an obstacle. In fact, a careful implementation through clique trees was necessary to prove the $O(nm)$ running time [48].

Vertex Incremental Minimal Triangulation [12]. Another $O(nm)$ time algorithm for computing minimal triangulations was presented recently by Berry, Heggernes, and Villanger [12]. This algorithm is based on the edge characterization of chordal graphs given in Theorem 3.5, and solves also a more general maintenance problem for chordal graphs. Assume that one is given a chordal graph $G = (V, E)$, a new vertex $u \notin V$, and a set of new edges D between u and some of the vertices in V. The questions that are considered are: Is $G' = (V \cup \{u\}, E \cup D)$ chordal? If not, what is a minimal set of extra edges (not belonging to D) that must be added between u and vertices of V in G' so that the resulting graph is chordal, and what is a maximal subset of D that can be added to G along with vertex u so that the resulting graph is chordal? Using Theorem 3.5, it is shown that a new edge uv can be added to G if and only if every edge ux, such that x belongs to a minimal u,v-separator in G, is either present in G or is also added to G along with edge uv. This way an algorithm that simultaneously generates a minimal triangulation and a maximal chordal subgraph of G in $O(nm)$ time is presented. Starting from an empty set U, a new vertex of G is processed and added to the set U of already processed vertices at each step, so that the transitory graph of each step is a minimal triangulation (or a maximal chordal subgraph) of $G(U)$. Interesting properties of this algorithm are that the vertices of G can be processed in any desired order (that can also be supplied online), and that one can use maximal subtriangulation and minimal triangulation steps interchangeably.

We would like to end this section by mentioning that other algorithmic approaches to computing minimal triangulations through saturation of minimal separators can be deduced from other characterizations of minimal triangulations. As mentioned before, Characterization 5.4 from [70] was based on contemporary results by Kloks, Kratsch, and Spinrad [55]. In [55], the following more algorithmic characterization of minimal triangulations was already given.

Characterization 5.5 (Kloks, Kratsch, and Spinrad [55]) *Let S be a minimal separator of a general graph $G = (V, E)$, and let $G' = (V, E')$ be the graph obtained from G by saturating S. Let further $C_1, C_2, ..., C_k$ be the connected components of $G(V \setminus S)$. $H = (V, E' \cup F)$ 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 $G'(S \cup C_i)$.*

Finally, in [19] the following similar characterization of minimal triangulations was given. For this characterization, a *potential maximal clique* of a graph G is defined to be a maximal clique in some minimal triangulation of G.

Characterization 5.6 (Bouchitté and Todinca [19]) *Let K be a potential maximal clique of a general graph $G = (V, E)$, let $G' = (V, E')$ be the graph obtained from G by saturating K. Let further $C_1, C_2, ..., C_k$ be the connected components of $G(V \setminus K)$, and $S_i = N_G(C_i)$ for $1 \leq i \leq k$. $H = (V, E' \cup F)$ 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 $G'(S_i \cup C_i)$.*

These and similar ideas are partially proven also by other authors and utilized in several minimal triangulation algorithms, as we have already seen and will see in the coming sections.

6 Minimal triangulation sandwich problem

Characterization 2.1 of minimal triangulations has direct implications for chordal graphs in general. It says that if $G \subset H$ for two chordal graphs G and H on the same vertex set, then there is a sequence of edges that can be removed from H one by one, such that the resulting graph after each removal is chordal, until we reach G. In the same manner, we can add a sequence of edges one by one to G to reach H, obtaining a new chordal graph at each step. Consequently, if a triangulation H of G is not minimal, there is a sequence of fill edges that can be removed from H to result in a minimal triangulation, such that the graph obtained after each removal is a triangulation of G. By Characterizations 2.1 and 2.2, each fill edge that is not the unique chord of a 4-cycle in H is a candidate for removal. In 1999 Ibarra [50] gave dynamic algorithms to test in $O(n)$ time whether a given edge can be removed from a given chordal graph without destroying chordality and, if the answer is yes, remove it within the same time bound. Thus checking whether a given triangulation is minimal, and finding a candidate for removal if not, can be done in $O(nf)$ time, where f is the number of fill edges. Unfortunately, as some fill edges may become candidates for removal only after the removal of some other fill edges, $O(nf)$ time is necessary for each removed fill edge with this approach. Thus each fill edge might have to be checked many times during the process of constructing chordal graphs that are between H and a minimal triangulation of G.

In 1996, Blair, Heggernes, and Telle posed and solved the following problem [13]: Given a graph G and an ordering α of G, compute a minimal triangulation H of G such that $H \subseteq G^+_\alpha$, thus H is "sandwiched" between G and G^+_α. We will call this the *minimal triangulation sandwich problem.* The problem is motivated from sparse matrix computations and the desire to compute orderings that both are minimal and produce few fill edges. Thus one can first use a popular heuristic to compute a "good" ordering α and then find a minimal triangulation with at most as many fill edges. The authors of [13] gave an algorithm, called MinimalChordal, to solve this problem with running time $O(f(m + f))$, where f is the number of fill edges in G^+_α. Short time after, Dahlhaus [29] presented an $O(nm)$ time algorithm to solve the same problem. Both these algorithms first compute G^+_α, and they use LEX M in parts of their computation. Interestingly, also Algorithm LB-Triang described in the previous section solves this problem. If α is the order in which LB-Triang processes the vertices of the input graph G, then the minimal triangulation H produced by LB-Triang satisfies $H \subseteq G^+_\alpha$. A nice property of LB-Triang compared to the first two mentioned algorithms is that it does not need G^+_α in its computation at all, and it does not make use of LEX M or any other algorithm. Finally, two different algorithms with an iterative approach were given to solve the same problem, respectively by Peyton [72], and by Berry, Heggernes, and Simonet [11]. Both these algorithms compute iterative refinements from an initial Minimum Degree ordering, and their running time is dependent on the running time of Minimum Degree and the number of iterations. Although the theoretical time bound is not impressive, such algorithms may perform very fast in practice [72].

MinimalChordal [13]. The approach of this algorithm is actually quite similar to the one described in the first paragraph of this section, however the authors are able to decide an order in which the fill edges of G_α^+ should be examined so that no edge needs to be checked for removal more than once. MinimalChordal takes as input a graph G and an ordering α. First, G_α^+ is computed, and during this computation, fill edges added at each step i are stored in separate sets F^i. The main result of [13] is that if fill edges are examined in the reverse order of their introduction, that is, edges in F^n first, F^{n-1} next, etc, then one never needs to reexamine a group of fill edges that has already been examined. The algorithm then straightforwardly runs through n steps, starting from $i = n$ and continuing backward to 1, and at each step i removes all the candidate edges of F^i, passes the resulting non-chordal subgraph touching these edges on to LEX M to receive a minimal triangulation of this part where some of the removed edges are possibly reinserted by LEX M. Interestingly, if one starts with an ordering α that produces low fill, then MinimalChordal runs considerably faster than LEX M, as documented in [13]. Thus running LEX M on small parts of the graph many times is faster than running LEX M once on the whole graph. Note that the running time of LEX M is not sensitive to the number of fill edges produced.

Dahlhaus' algorithm [29]. Also this algorithm takes as input a graph G and an ordering α of G. First, G_α^+ and a clique tree T of G_α^+ are computed. Since G_α^+ is not necessarily a minimal triangulation, some minimal separators of G_α^+ are *not* minimal separators of G. Remember that the edges of T correspond to the minimal separators of G_α^+. The next step in the algorithm is to split some of the tree edges of T, adding necessary maximal cliques in between, so that each tree edge of the resulting tree T' is a minimal separator of G. Now the chordal graph G' of which T' is a clique tree is a triangulation of G and satisfies $G \subseteq G' \subseteq G_\alpha^+$. However, G' is not necessarily a minimal triangulation of G. Some of the minimal separators of G that do not cross minimal separators of G' might be missing as edges of T'. Thus some of the tree nodes need to be split into new tree nodes, and edges (representing minimal separators of G) between new tree nodes must be inserted until no such refinement is possible. This is done through the final step of the algorithm, which is basically to run a modified version of LEX M on G so that fill edges are introduced only within the maximal cliques of G'. By the correctness of LEX M, the resulting graph is a minimal triangulation, and the author proves that the modification ensures that the resulting minimal triangulation is a subgraph of G'. The step of splitting edges is described in detail and this step takes $O(nm)$ time, and thus the total time complexity is bounded by the running time $O(nm)$ of LEX M.

Peyton's algorithm [72]. This algorithm is somewhat similar to the algorithm of Dahlhaus, but uses tools from sparse matrix computations rather than general graph techniques. Again the input is G and $\alpha = \{v_1, v_2, ..., v_n\}$, and the algorithm starts by computing G_α^+. However, instead of a clique tree of G_α^+, the author uses an *elimination tree* T of G_α^+. An elimination tree is a useful tool in sparse matrix computations. It is a rooted tree whose root is vertex v_n, and the parent of each vertex $x \neq v_n$ is the first vertex (with smallest index) of $\{v_{\alpha(x)+1}, v_{\alpha(x)+2}, ..., v_{n-1}, v_n\}$ that belongs to $N_{G_\alpha^+}(x)$. In the algorithm of [72], T is first post ordered, and the vertices of T are glued together in *supernodes*, resembling the tree nodes of a clique tree. A child-parent pair of vertices c and p belong to the same supernode if they have the same set of neighbors in G^+_α among the ancestors of p in T. The algorithm processes the supernodes of T in a topological order. The basic idea is to order each supernode locally by Minimum Degree. Note that each supernode S is a clique in G^+_α. For each supernode S, call the graph G_S in which all descendants of S have been eliminated from G by Elimination Game. The main result of [72] is that for each fill edge uv of G^+_α with $u \in S$, uv is a candidate for removal if and only if uv is *not* an edge of G_S. Consequently, the vertex of S of minimum degree in G_S is the vertex of S with the largest number of candidate edges incident to it in G^+_α. Therefore by ordering the vertices of G in such a way that the topological ordering of the supernodes is preserved, and the vertices within each supernode is eliminated by choosing the vertex of minimum degree at each elimination step, we are guaranteed that some candidate edges will be removed if any candidate edges exist in G^+_α. The algorithm iterates this process until no further edges can be removed at one iteration. Although no theoretical time bound is given for the algorithm, it is documented to run fast i practice.

Minimum Degree exhibits interesting behavior regarding minimal triangulations, of which the above paragraph is an example. In practice, Minimum Degree orderings are often observed to produce minimal triangulations [13, 72]. Why Minimum Degree has this desirable effect was partially explained by Berry, Heggernes, and Simonet in [11]. Given a graph G and an ordering α, they define the set of substars of G^+_α in the following way. At each step i of Elimination Game, for each connected component C of $G^{i-1}(\{v_{i+1}, ..., v_n\} \setminus N_{G^{i-1}}(v_i))$, $N_{G^{i-1}}(C)$ is a substar of G^+_α. The authors of [11] show that the set of substars of G^+_α is a (not necessarily maximal) set of non-crossing minimal separators of G. Thus if only one substar is defined at each step of Elimination Game, then the resulting triangulation is minimal. If a vertex of minimum degree is chosen at each step of Elimination Game, then as long as the union of all substars of each step is also a substar, the produced triangulation is minimal. Thus chances of Elimination Game to produce a minimal triangulation increases substantially when a vertex of minimum degree is chosen at each step. In [11] also an algorithm, called Minimal Minimum Degree, is given that runs Minimum Degree at each step and iterates until the resulting triangulation is minimal.

Minimal Minimum Degree [11]. At each iteration i of this algorithm, the following steps are executed: remove all vertices that satisfy Theorem 3.4, compute a Minimum Degree ordering α_i, compute $G^+_{\alpha_i}$, and remove from $G^+_{\alpha_i}$ all fill edges that do not appear within a substar. Repeat the process until no edges can be removed at one iteration. It is shown that the resulting triangulation H is minimal and it is a subgraph of $G^+_{\alpha_1}$, where α_1 is the initial Minimum Degree ordering.

Another popular heuristic for the minimum triangulation problem, which is used widely by the sparse matrix society to produce good orderings is Nested Dissection, and it was introduced in early 70's, long before the connection between minimal triangulations and minimal separators was known. Nevertheless, Nested Dissection uses the separators of the input graph to produce a good ordering. Initially a separator S of input graph $G = (V, E)$ is computed, and in the resulting elimination ordering the vertices of S are ordered last. Then this process is recursively repeated on each connected component of $G(V \setminus S)$. Nested Dissection does not necessarily compute minimal elimination orderings. However, choosing a minimal separator of the input graph at each recursive step certainly defines a set of non-crossing minimal separators of G. Thus if the separators are chosen carefully, meos can be computed by Nested Dissection. A Nested Dissection algorithm given by Bornstein, Maggs, and Miller [18] in 1999 was later shown by Dahlhaus to produce meos [32]. In the same paper, Dahlhaus also presents an algorithm for converting a given Nested Dissection ordering into a meo, thus bringing yet another solution to the minimal triangulation sandwich problem.

7 Minimal triangulation of restricted graph classes

For some special graph classes minimal triangulation algorithms are given with time bound $o(n^{2.69})$, that is, strictly better than the best known time bound for the general case. For some of these classes, the generally NP-hard problem of computing a minimum triangulation can be solved in time $o(n^{2.69})$, and this gives also the best known minimal triangulation algorithm. For example, minimum triangulation and treewidth can be computed for distance hereditary graphs in linear time [21], and for trapezoid graphs in $O(n^2)$ time [16]. For other classes, computing a minimum triangulation is either NP-hard or requires more time than computing a minimal triangulation, and techniques that are especially designed for minimal triangulations are used to achieve a better bound. In this section we will look at some of the graph classes of the latter type, and mention the results that are related to their minimal triangulation.

Algorithm for graphs of bounded degree [33]. In 2002, Dahlhaus presented an algorithm that computes minimal triangulations of bounded degree graphs efficiently [33]. Actually, this algorithm is not especially designed for bounded degree graphs, but for general graphs. The running time of the algorithm is $O(n(\Delta^3 + \alpha(n)))$ on general graphs, where Δ is the maximum degree and α is the inverse Ackerman's function. Thus for graphs of bounded degree, the running time becomes $O(n\alpha(n))$ automatically. The algorithm starts by computing a spanning tree T of the given connected graph G. Then the vertices of T are post ordered to result in ordering $\{v_1, v_2, ..., v_n\}$. Next, the vertices of G are partitioned into vertex sets $V_1, V_2, ..., V_n$, where V_i is the set of vertices that have v_i as their largest numbered closed neighbor. More formally, $V_i = \{x \mid i = max\{j \mid v_j \in N_G[x]\}\}$. Note that these vertex sets are disjoint and some of them might be empty. The author shows that there exists a minimal elimination ordering β on G such that the vertices of V_i are ordered before the vertices of V_{i+1} in β, for $1 \leq i \leq n-1$. Hence, as we have seen in several other algorithms, this algorithm starts by ordering vertex subsets. In order to compute a correct minimal ordering, the vertices within each subset V_i must be ordered locally in a correct way. To achieve this, elements that we have already mentioned in connection with Peyton's algorithm [72], like elimination trees and supernodes (called elimination equivalence classes in [33]) are used. Each subset V_i is further partitioned into equivalence classes, and an elimination tree whose nodes correspond to these equivalence classes is constructed. Two vertices x and y are defined to belong to the same elimination equivalence class if they are in the same V_i and in the same connected component of $G(V_1 \cup V_2 \cup ... \cup V_i)$. The author shows that the work so far can be done in time $O(n\alpha(n))$. Using the elimination tree, the fill edges within each equivalence class that are compatible with the partial ordering $V_1, V_2, ..., V_n$ are computed. Then to achieve a complete minimal elimination ordering, each equivalence class is locally ordered by a variant of Lex-BFS on the subgraph induced by the equivalence class vertices and the added fill edges. This adds an extra time requirement of $O(\Delta^3)$ for each equivalence class.

Algorithm for planar graphs [30, 31]. In 1998, Dahlhaus presented an algorithm for computing minimal a triangulation of a planar input graph G in linear time [30], and later an enhancement of it that is also parallelizable [31]. Also this algorithm starts in the same way as the algorithm of [33] described above, and computes the sets $V_1, V_2, ..., V_n$ as defined in [33], but only the non-empty subsets, resorted as $V_1, V_2, ..., V_k$, are further considered. For each subset V_i and each vertex $v \in V_i$, define $l(v) = i$. For each face f of G, the algorithm adds fill edges between pairs of vertices x, y of f if one of the two paths between x and y of the cycle surrounding f contains intermediate vertices that belong to V_t with $t < l(x)$ and $t < l(y)$. The resulting graph is called G'. Clearly, G' is also planar, and the added fill edges agree with any minimal elimination ordering that orders the vertices of V_i before the vertices of V_{i+1} for $1 \leq i \leq k-1$. The algorithm then adds edges to achieve a triangulation G'' of G' which is not necessarily a minimal triangulation of G. Finally, perfect elimination orderings of parts of G'' are computed to remove unnecessary edges, and how to do this is explained in a series of lemmas which conclude that the resulting graph is a minimal triangulation of G and can be computed in linear time.

We have assumed that graphs of bounded degree and planar graphs are well known graph classes. For the following algorithms, we need to define the involved graph classes before each algorithm. In addition to efficient algorithms on these classes, there have also been given characterizations of some of them through their minimal triangulations. These characterizations give more insight both to minimal triangulations in general and to the corresponding graph class.

AT-free graphs are often mentioned in connection with minimal triangulations. Three non-adjacent vertices u, v, x form an *asteroidal triple (AT)* if there is a path between every two of them that does not contain a neighbor of the third. A graph is *AT-free* if it does not contain an AT. A graph G is an *interval* graph if continuous intervals can be assigned to each vertex of G such that two vertices are neighbors if and only if their intervals intersect. Lekkerkerker and Boland [59] showed that a graph is an interval graph if and only if it is chordal and AT-free. The only-if part of the following characterization of AT-free graphs through their minimal triangulations was proved by Möhring [64], and the if part was proved independently by Corneil, Olariu, and Stewart [27], and by Parra and Scheffler [70].

Theorem 7.1 ([27], [64], [70]) *A graph G is AT-free if and only if every minimal triangulation of G is an interval graph.*

Thus for AT-free graphs, computing a minimal triangulation is equivalent to computing a minimal interval completion. A similar result exists for proper interval graphs. A graph G is a *proper interval* graph if G is an interval graph and none of the intervals associated with the vertices of G contain another interval properly. Proper interval graphs are related to AT-free claw-free graphs in the same way interval graphs are related to AT-free graphs. A graph is *claw-free* if it does not contain an induced copy of $K_{1,3}$. A graph is called *AT-free claw-free* if it is both AT-free and claw-free.

Theorem 7.2 (Parra and Scheffler [70]) *A graph G is AT-free claw-free if and only if every minimal triangulation of G is a proper interval graph.*

Algorithm for AT-free claw-free graphs [63]. In 2003, Meister presented a linear time algorithm for minimal triangulation of AT-free claw-free graphs [63]. His algorithm is a variant of Lex-BFS. Analogous to the equivalence between LEX M and partitioning the vertex set into groups having the same labels, Lex-BFS can also be implemented in the same way. Thus, a vertex v belonging to the group corresponding to the largest label is chosen at each step, and each existing group is split into two subgroups according to whether or not the vertices in each group are neighbors of v. Meister's variant of Lex-BFS is called min-LexBFS [63], and at each step, only the group corresponding to the smallest label is partitioned into two in the described way. Note that Lex-BFS and min-LexBFS are not equivalent. There are orderings that can be generated by one and that cannot be generated by the other, and min-LexBFS does not necessarily produce a peo of a chordal input graph. A 2min-LexBFS ordering of a graph G is achieved by first running min-LexBFS with an arbitrary start vertex and achieving an ordering α, and then running min-LexBFS again on G but this time breaking ties at each step by choosing a vertex v with smallest $\alpha(v)$. A 2min-LexBFS ordering can certainly be computed in linear time. The author proves that an ordering α on an AT-free claw-free graph G is a minimal elimination ordering if and only if α is a 2-LexBFS ordering of G.

In [63] it is also shown that for every AT-free claw-free graph G, it is decidable in linear time whether a given triangulation of G is minimal.

The next algorithm we will describe is for co-comparability graphs. A graph is called *transitively orientable* if each edge uv can be assigned an orientation, either $u \rightarrow v$ or $v \rightarrow u$, such that for every triple of vertices u, v, w, the following holds: $u \rightarrow v \wedge v \rightarrow w \Rightarrow u \rightarrow w$. The class of *comparability graphs* are the class of transitively orientable graphs. A graph is called a *co-comparability graph* if it is the complement of a comparability graph. It is shown by Kratsch and Stewart [57] that $G = (V, E)$ is a co-comparability graph if and only if there is an ordering α on V such that for every triple of vertices u, v, w, the following holds: $\alpha(u) < \alpha(v) < \alpha(w) \wedge uw \in E \Rightarrow uv \in E$ or $vw \in E$. Such an ordering is called a *co-comparability ordering*. Co-comparability graphs are a subset of AT-free graphs [46].

Algorithm for co-comparability graphs [63]. In the above mentioned 2003 paper by Meister [63], also a linear time minimal triangulation algorithm for co-comparability graphs is given. The author first defines an extension of min-LexBFS, called min-LexBFS*, which takes as input a graph G and an ordering α on the vertices of G, and runs min-LexBFS using ordering α to break ties as explained above. It is then shown that if a co-comparability ordering α of a co-comparability graph G is already known, then a minimal elimination ordering β is computed in linear time by $\beta =$min-LexBFS*(G, α). Since co-comparability orderings can also be generated in linear time [62], this results in an overall linear time algorithm for minimal triangulation of co-comparability graphs.

The last result that we will mention in this section concerns cographs and trivially perfect graphs. Both of these graph classes have various definitions and characterizations. For our purposes, we mention that a graph is a *cograph* if and only if it contains no induced copy of a P_4 (a path on 4 vertices), and that a graph is *trivially perfect* if and only if it is a chordal cograph.

Theorem 7.3 (Parra and Scheffler [70]) *A graph G is a cograph if and only if every minimal triangulation of G is trivially perfect.*

8 Concluding remarks

We have seen that two well-known characterizations of chordal graphs, respectively by Fulkerson and Gross, and by Dirac, both give triangulation algorithms directly. However, whereas the algorithm connected to Fulkerson and Gross' characterization does not necessarily give minimal triangulations, the algorithm connected to Dirac's characterization does, and it actually characterizes minimal triangulations. Still, many different minimal triangulation algorithms exist, as we have listed, and each of these algorithms gives new insight to minimal triangulations, triangulations in general, and chordal graphs. Also for several other related problems, the insight gained through minimal triangulation results has been useful.

The problem which is most closely related to minimal triangulations is perhaps the minimum triangulation problem. A minimum triangulation can be computed by generating all minimal triangulations, and choosing one with the fewest number of edges. For graphs having a polynomial number of minimal triangulations, minimum triangulation problem can be solved in polynomial time. As we have already mentioned, even for graphs having a polynomial number of minimal separators minimum triangulations can be computed in polynomial time [19]. This is why counting the number of minimal separators, and listing all minimal separators are important related problems [9, 54]. Recently, it is shown that the number of separators in any graph is $O(n \cdot 1.7087^n)$ [39]. In the same paper, an exact algorithm for solving the minimum triangulation problem on general graphs in $O(\text{poly}(n) \cdot 1.9601^n)$ time is also given. Furthermore, the minimum triangulation problem belongs to the class of fixed parameter tractable problems [51, 23]. When fast and practical algorithms with polynomial running time are required, the usual approach is to use heuristics like Minimum Degree and Nested Dissection. In addition, polynomial time constant factor approximation algorithms exist for the minimum triangulation problem [66].

Another related problem to minimal triangulations is the problem of minimal interval completion. Minimum interval completion and minimum proper interval completion problems are NP-hard [41, 44], however the status of minimal interval completions of general graphs is open. A fast algorithm for minimal interval completion would definitely be of high interest, both theoretically and in practice, as interval graphs appear frequently in biological computations. More general, it is interesting to investigate whether there are graph classes $\mathcal{C}$ such that minimal $\mathcal{C}$-completion of arbitrary graphs is NP-hard. Also, for which interesting graph classes $\mathcal{C}$ can minimum $\mathcal{C}$-completion of general graphs be computed in polynomial time?

A problem that was already mentioned in Section 5 is the maximal subtriangulation problem: given an arbitrary graph H, find a chordal subgraph G of H on the same vertex set such that no chordal graph M satisfying $G \subset M \subset H$ exists. We should mention that this problem is solvable in time $O(\Delta m)$, where Δ is the highest degree in the input graph [3, 36, 78]. An interesting question is whether faster algorithms for it can be found. The maximum subtriangulation problem is NP-hard [67], and thus polynomial time approximation algorithms and resolving its parameterized complexity would be of interest.

Finally, the ultimate ongoing challenge regarding minimal triangulations is whether minimal triangulations of general graphs can be computed in time $o(n^{2.69})$.

References

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

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

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

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

[5] A. Berry. *Désarticulation d'un graphe.* PhD thesis, LIRMM, Montpellier, France, 1998.

[6] A. Berry. A wide-range efficient algorithm for minimal triangulation. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages S860–S861, 1999.

[7] A. Berry, J. R. S. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica.* To appear.

[8] A. Berry and J.-P. Bordat. Separability generalizes Dirac's theorem. *Disc. Appl. Math.*, 84:43–53, 1998.

[9] A. Berry, J. P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. *Int. J. Foundations Comp. Sci.*, 11(3):397–403, 2000.

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

[11] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. *Proceedings of the* 29^{th} *Workshop on Graph Theoretic Concepts in Computer Science, Springer Verlag*, pages 58 – 70, 2003. Lecture Notes in Computer Science 2880.

[12] A. Berry, P. Heggernes, and Y. Villanger. A vertex incremental approach for dynamically maintaining chordal graphs. In *Proceedings 14th International Symposium on Algorithms and Computation - ISAAC 2003*, pages 47 – 57. Springer Verlag, 2003. LNCS 2906.

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

[14] 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.

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

[16] H. L. Bodlaender, T. Kloks, D. Kratsch, and H. Müller. Treewidth and minimum fill-in on d-trapezoid graphs. *J. Graph Alg. Appl.*, 2:1–23, 1998.

[17] H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. Technical Report UU-CS-2003-027, Institute of information and computing sciences, Utrecht University, Netherlands, 2002.

[18] C. F. Bornstein, B. M Maggs, and G. L. Miller. Tradeoffs between parallelism and fill in Nested Dissection. In *SPAA 99 - Eleventh ACM Symposium on Parallel Algorithms and Architectures*, pages 191–200, 1999.

[19] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.

[20] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theor. Comput. Sci.*, 276(1-2):17–32, 2002.

[21] H. J. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Disc. Appl. Math.*, 99:367–400, 2000.

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

[23] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[24] N. Chandrasekharan and S. S. Iyengar. NC algorithms for recognizing chordal graphs and k-trees. *IEEE Trans. Comp.*, 37(10):1178–1183, 1988.

[25] F. R. K. Chung and D. Mumford. Chordal completions of planar graphs. *J. Comb. Theory*, 31:96–106, 1994.

[26] D. Coppersmith and S. Winogard. Matrix multiplication via arithmetic progressions. *J. Symb. Comp.*, 9:1–6, 1990.

[27] D. G. Corneil, S. Olariu, and L. Stewart. Asteroidal triple-free graphs. *SIAM J. Disc. Math.*, 10:399–430, 1997.

[28] E. Dahlhaus. The parallel complexity of elimination ordering procedures. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 225–236, 1993.

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

[30] E. Dahlhaus. Minimal elimination of planar graphs. In S. Arnborg and L. Ivansson, editors, *Algorithm Theory - SWAT 1998*, pages 210–221. Springer Verlag, 1998. Lecture Notes on Computer Science 1432.

[31] E. Dahlhaus. An improved linear time algorithm for minimal elimination ordering in planar graphs that is parallelizable. 1999. Submitted.

[32] E. Dahlhaus. Converting a Nested Dissection into a minimal elimination ordering efficiently. 2000. Unpublished manuscript.

[33] E. Dahlhaus. Minimal elimination ordering for graphs of bounded degree. *Disc. Appl. Math.*, 116:127–143, 2002.

[34] E. Dahlhaus and M. Karpinski. An efficient parallel algorithm for the minimal elimination ordering of an arbitrary graph. *Theor. Comput. Sci.*, 134(2):493–528, 1994.

[35] E. Dahlhaus, M. Karpinski, and M. Novick. Fast parallel algorithms for the clique separator decomposition. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1990.

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

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

[38] A. Edenbrandt. Chordal graph recognition is in NC. *Information Processing Letters*, 24:239–241, 1987.

[39] F. V. Fomin, D. Kratsch, and I. Todinca. Exact algorithms for treewidth and minimum fill-in. 2004. Unpublished manuscript.

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

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1978.

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

[43] J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.

[44] P. W. Goldberg, M. C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of dna. *J. Comput. Bio.*, 2(1):139–152, 1995.

[45] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[46] M. C. Golumbic, C. L. Monma, and Jr. W. T. Trotter. Tolerance graphs. *Disc. Appl. Math.*, 9(2):157–170, 1984.

[47] A. Hajnal and J. Surányi. Über die Ausflösung von Graphen in vollständige Teilgraphen. *Ann. Univ. Sci. Budapest*, pages 113–121, 1958.

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

[49] C. W. Ho and R. C. T. Lee. Counting clique trees and computing perfect elimination schemes in parallel. *Information Processing Letters*, 31:61–68, 1989.

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

[51] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999.

[52] T. Kloks, H. L. Bodlaender, H. Müller, and D. Kratsch. Computing treewidth and minimum fill-in: All you need are the minimal separators. In T. Lengauer, editor, *Algorithms - ESA 1993*, pages 260–271. Springer Verlag, 1993. Lecture Notes on Computer Science 726.

[53] T. Kloks, H. L. Bodlaender, H. Müller, and D. Kratsch. Erratum to ESA 1993 proceedings. In *Algorithms - ESA 1994*, page 508. Springer Verlag, 1994. Lecture Notes on Computer Science 855.

[54] T. Kloks and D. Kratsch. Listing all minimal separators of a graph. *SIAM J. Comput.*, 27(3):605–613, 1998.

[55] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theor. Comput. Sci.*, 175:309–335, 1997.

[56] D. Kratsch and J. Sprinrad. Minimal fill in $o(n^3)$ time. 2004. Submitted.

[57] D. Kratsch and L. Stewart. Domination on cocomparability graphs. *SIAM J. Disc. Math.*, 6(3):400–417, 1993.

[58] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *J. Royal Statist. Soc., ser B*, 50:157–224, 1988.

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

[60] J. W. H. Liu. Equivalent sparse matrix reorderings by elimination tree rotations. *SIAM J. Sci. Stat. Comput.*, 9(3):424–444, 1988.

[61] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[62] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Disc. Appl. Math.*, 201(1-3):189–241, 1999.

[63] D. Meister. Recognizing and computing minimal triangulations efficiently. Technical Report 302, Julius Maximilians Universität Würzburg, 2002.

[64] R. H. Möhring. Triangulating graphs without asteroidal triples. *Disc. Appl. Math.*, 64:281–287, 1996.

[65] J. Naor, M. Naor, and A. Schäffer. Fast parallel algorithms for chordal graphs. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 355–364. ACM Press, 1987.

[66] A. Natanzon, R. Shamir, and R. Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Computing*, 30:1067–1079, 2000.

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

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

[69] T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix. *J. Math. Anal. Appl.*, 54:622–633, 1976.

[70] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Disc. Appl. Math.*, 79:171–188, 1997.

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

[72] B. W. Peyton. Minimal orderings revisited. *SIAM J. Matrix Anal. Appl.*, 23(1):271–294, 2001.

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

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

[75] 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.

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

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

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

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