Sequential and Parallel Triangulating Algorithms for Elimination Game and New Insights on Minimum Degree

Anne Berry, Elias Dahlhaus, Pinar Heggernes, Geneviève Simonet

HAL Id: lirmm-00366031

https://hal-lirmm.ccsd.cnrs.fr/lirmm-00366031

Submitted on 5 Mar 2009

Sequential and parallel triangulating algorithms for Elimination Game and new insights on Minimum Degree

Anne Berry[*]   Elias Dahlhaus[†]   Pinar Heggernes[‡]

Geneviève Simonet[§]

June 23, 2008

Abstract

[1] Elimination Game is a well known algorithm that simulates Gaussian elimination of matrices on graphs, and it computes a triangulation of the input graph. The number of fill edges in the computed triangulation is highly dependent on the order in which Elimination Game processes the vertices, and in general the produced triangulations are neither minimum nor minimal. In order to obtain a triangulation which is close to minimum, the Minimum Degree heuristic is widely used in practice, but until now little was known on the theoretical mechanisms involved.

In this paper we show some interesting properties of Elimination Game; in particular that it is able to compute a partial minimal triangulation of the input graph regardless of the order in which the vertices are processed. This results in a new algorithm to compute minimal triangulations that are sandwiched between the input graph and the triangulation resulting from Elimination Game. One of the strengths of the new approach is that is is easily parallelizable, and thus we are able to present the first parallel algorithm to compute such sandwiched minimal triangulations. In addition, the insight that we gain through Elimination Game is used to partly explain the good behavior of the Minimum Degree algorithm. We also give a new algorithm for producing minimal triangulations that is able to use the minimum degree idea to a wider extent.

1   Introduction

For the past forty years, problems arising from applications have given rise to challenges for graph theorists, and thus also to a wealth of graph-theoretic

---

[*]LIMOS, bat. ISIMA, F-63173 Aubière cedex, France. `berry@isima.fr`

[†]Department of Computer Science, Technische Universit ät Darmstadt, D-64277 Darmstadt, Germany. `dahlhaus@algo.informatik.tu-darmstadt.de`

[‡]Department of Informatics, University of Bergen, PB 7800, N-5020 Bergen, Norway. `pinar@ii.uib.no`

[§]LIRMM, 161 Rue Ada, F-34392 Montpellier, France. `simonet@lirmm.fr`

[1]Extended abstracts of preliminary versions of parts of this paper have appeared in [10] and in [5].

results. One of these is computing a minimum triangulation. Although the problem originally comes from the field of sparse matrix computations [29], it has applications in various areas of computer science.

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 physical structure can often be thought of as covered by a mesh 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 mesh. Such systems are solved through standard methods of linear algebra, like Gaussian elimination, and during this process non-zero entries are inserted into cells of the matrix that originally held zeros, which increases both the storage requirement and the time needed to solve the system. It was observed early that finding a good pivotal ordering of the matrix can reduce the amount of *fill* thus introduced: in 1957, Markowitz [22] introduced the idea behind the algorithm known today as Minimum Degree, choosing a pivot row and column at each step of the Gaussian elimination to locally minimize the product of the number of corresponding off-diagonal non-zeros. Tinney and Walker [33] later applied this idea to symmetric matrices, and Rose [29] developed a graph theoretical model of it.

As early as 1961, Parter [26] presented an algorithm, known as Elimination Game (EG), which simulates Gaussian elimination on graphs by repeatedly choosing a vertex and adding edges to make its neighborhood into a clique before removing it, thus introducing the connection between sparse matrices and graphs. In view of the results of [14], the class of graphs produced by EG is exactly the class of chordal graphs. Thus when the given graph is not chordal, Gaussian elimination and EG correspond to embedding it into a chordal graph by adding edges, a process called *triangulation*. As can be observed on the example in Figure 1, the number of fill edges in the resulting triangulation is heavily dependent on the order in which EG processes the vertices. This ordering of the graph corresponds to the pivotal ordering of the rows and columns in Gaussian elimination.

As mentioned above, it is of primary importance to add as few edges as possible when running EG. The corresponding problem is that of computing a *minimum triangulation*, which is NP-hard [35]. It is possible to compute in polynomial time a triangulation which is *minimal*, meaning that an inclusion-minimal set of edges is added [24], [31]. However, such a triangulation can be far from minimum, as can be seen from the example of Figure 1(b). In fact, it is easy to see that this example can be extended to a graph with $O(n)$ edges and an $O(n^2)$ size fill, whereas a unique fill edge can be obtained by EG on this graph.

As a result, researchers have resorted to heuristics, of which one of the most universally used and studied is *Minimum Degree (MD)*: this runs EG by choosing at each step a vertex of minimum degree in the transitory elimination graph, as illustrated by Figure 1(d). This algorithm is widely used in practice, and it is known to produce low fill triangulations. In addition, MD is also observed [6] to produce triangulations which are often minimal or close to minimal.

MD has given rise to a large amount of research with respect to improving the running time of its practical implementations, and the number of papers written on this subject is in the hundreds [1, 16]. However, very little is proved about its quality. It has in fact been analyzed theoretically only to a limited
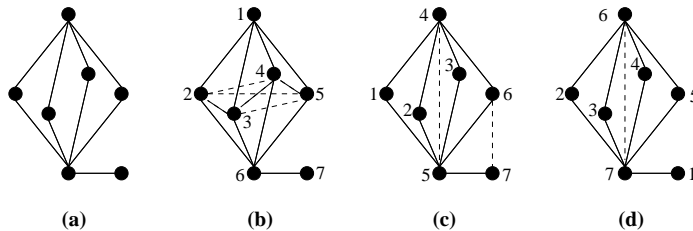
Figure 1: (a) A graph G, and various triangulations of G by EG through the given orderings: (b) A minimal triangulation with $O(n^2)$ fill edges. (c) A non-minimal triangulation of G with less fill. (d) A minimum triangulation of G.

extent, which makes it difficult to gain control over this heuristic in order to improve it yet further, although recent research has been done on algorithms for low fill minimal triangulations [6, 10, 27].

In this paper, we use recent graph theoretical results on minimal triangulation and minimal separators to explain, at least in part, why MD yields such good results. In fact, it turns out that one of the reasons why MD works so well is that the EG algorithm is remarkably *robust*, in the sense that it is resilient to error: if at some step of the process an undesirable edge with respect to minimal triangulation is added, at later steps the chances of adding only desirable edges remain intact. During EG, and in particular MD, we are able to identify the fill edges that are safe to add with respect to a minimal triangulation. Thus we show how to use EG to compute a partial minimal triangulation. We also show the implementation details of how to compute this partial minimal triangulation efficiently. An interesting property of this partial minimal triangulation is that when it is completed in any way to a minimal triangulation the resulting graph is a minimal triangulation of the input graph and a subgraph of the filled graph resulting from EG.

One of the strengths of this approach is its parallel nature, and we give implementation details for both a sequential and a parallel version of it. This results in the first efficient parallel algorithm for computing minimal triangulations sandwiched between the input graph and the filled graph resulting from EG.

Furthermore, we use the insight we have gained on the mechanisms which govern EG, and in particular MD, to propose a new algorithm that improves the results obtained by MD, giving minimal triangulations with low fill.

The remainder of this paper is organized as follows: Section 2 gives the graph theoretic background, introduces EG formally, and gives previous results on minimal separators and minimal triangulation. In Section 3, we show that EG can be used to compute a partial minimal triangulation, thereby giving some new invariants for EG. In Section 4, we explain how to implement this efficiently both sequentially and in parallel. Section 5 proposes new algorithms to compute minimal triangulations using EG. In particular we show how to extend a partial minimal triangulation to a minimal triangulation in parallel, thereby solving the sandwiched minimal triangulation problem in parallel. Section 6 applies our new results to MD, and uses our new insight to give some explanations as to the remarkably good behavior of MD.

3

2   Preliminaries

Given a graph $G = (V, E)$, we denote $n = |V|$ and $m = |E|$. For any subset S of V, $G(S)$ denotes the subgraph of G induced by S. For the sake of simplicity, we will use informal notations such as $H = G + \{e\} + \{x\}$ when H is obtained from G by adding edge e and vertex x. For any vertex v of G, $N_G(v)$ denotes the neighborhood of v in G, and $N_G[v]$ denotes the set $N_G(v) \cup \{v\}$. For a given set of vertices $X \subset V$, $N_G(X) = \cup_{v \in X} N_G(v) - X$, and $N_G[X] = \cup_{v \in X} N_G(v) \cup X$. We will omit the subscripts when there is no ambiguity.

A vertex is *simplicial* if its neighborhood is a clique. We will say that we *saturate* a set of vertices X when we add to the graph all the edges necessary to make X into a clique. A graph is *chordal*, or *triangulated*, if it contains no chordless cycle of length ≥ 4. A chordal graph $H = (V, E + F)$ is called a *triangulation* of $G = (V, E)$, where G is an arbitrary graph. The set F of edges which are added to obtain a triangulation is called a *fill*. A triangulation H is *minimal* if no strict subset of F can be added to G to obtain a triangulation.

A bijective function $\alpha : V \rightarrow \{1, 2, ..., n\}$ is called an *ordering* of the vertices of $G = (V, E)$, and (G, α) will denote a graph G, the vertices of which are ordered according to α. We will use $\alpha = (v_1, v_2, ..., v_n)$, where $\alpha(v_i) = i$.

The algorithmic description of Elimination Game (EG) given below defines the notations we will use in the rest of this paper:

Algorithm Elimination Game (EG)
Input: A graph $G = (V, E)$, and an ordering α of the vertices in G.
Output: A triangulation G_α^+ of G.
$G_\alpha^1 \leftarrow G; \quad G_\alpha^+ \leftarrow G;$
for $k = 1$ **to** n **do**
    Let F be the set of edges necessary to saturate $N_{G_\alpha^k}(v_k)$ in G_α^k;
    $G_\alpha^{k+1} \leftarrow G_\alpha^k + F - \{v_k\}; \quad G_\alpha^+ \leftarrow G_\alpha^+ + F;$

We note $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$, where $V_\alpha^k = \{v_k, ..., v_n\}$. According to the definition used in [24], we will call $G_\alpha^k(V_\alpha^k - N_{G_\alpha^k}[v_k])$ the *section graph at step k*. A characterization of the edges of G_α^+ is given in [31], which easily extends to G_α^k.

Lemma 2.1 ([31]) *Let $\alpha = (v_1, ..., v_n)$ and let i, j be distinct integers in $[1, n]$ (resp. $[k, n]$). Then $v_i v_j$ is an edge of G_α^+ (resp. G_α^k) iff there is a path in G between v_i and v_j (possibly reduced to one edge), all intermediate vertices of which have a number which is strictly smaller than $\min\{i, j\}$ (resp. k) in α.*

If no fill edges are produced during EG (i.e, if $G_\alpha^+ = G$) then α is called a *perfect elimination ordering (peo)* of G. Fulkerson and Gross showed in [14] that a graph is chordal iff it has a peo. Consequently, since α is a peo of G_α^+, EG is an algorithm for computing triangulations (not necessarily minimal). In [24] it is shown that any minimal triangulation of G can also be generated by EG. Thus for each minimal triangulation H of G, there exists an ordering α on G such that $H = G_\alpha^+$. Such an ordering α is called a *minimal elimination ordering*. If a given ordering α is not minimal, we will call a minimal triangulation H of G that is a subgraph of G_α^+, a *sandwiched* minimal triangulation.

The Minimum Degree (MD) heuristic is based on EG: it takes as input an unordered graph G, and computes an ordering α along with the corresponding

4

triangulation G_α^+, by choosing at each step a vertex of minimum degree in G_α^k and numbering it as v_k.

Minimal separators are central to chordal graphs and minimal triangulations. Given a graph $G = (V, E)$, a vertex set $S \subset V$ is a *separator* if $G(V - S)$ is disconnected. If $G(V - S)$ has a connected component C such that $N_G(C) = S$ then C is called a *full component of S in G*. A separator S is a *minimal separator* of G if S has at least two full components in G.

Characterization 2.2 (Dirac [13]) *A graph is chordal iff all its minimal separators are cliques.*

The idea behind the connection between minimal separators and minimal triangulations is that forcing a graph into respecting Dirac's characterization will result in a minimal triangulation, by repeatedly choosing a not yet processed minimal separator and saturating it [2, 19, 25]. We will need the definition of crossing separators, which characterize the separators that disappear when a saturation step of this process is executed:

Definition 2.3 ([19]) *Let S and S' be two minimal separators of G. S and S' are said to be* crossing *if there exist two connected components C_1, C_2 of $G(V - S)$, such that $S' \cap C_1 \neq \emptyset$ and $S' \cap C_2 \neq \emptyset$ (the crossing relation is symmetric).*

The saturation process described above can be generalized by choosing and simultaneously saturating a set of pairwise non-crossing minimal separators instead of a single minimal separator at each step, until a chordal graph is obtained. We will refer to this generalized process as the *Saturation Algorithm*. Given a set $\mathscr{S}$ of minimal separators of G, we will denote $G_\mathscr{S}$ the graph obtained from G by saturating all the separators belonging to $\mathscr{S}$.

The following results from the works of Kloks, Kratsch and Spinrad [19] and Parra and Scheffler [25] provide a proof of this algorithm and will be used in Sections 3 and 5.

Theorem 2.4 ([19, 25]) *A graph $H = (V, E + F)$ is a minimal triangulation of $G = (V, E)$ iff there is a maximal set $\mathscr{S}$ of pairwise non-crossing minimal separators of G such that $H = G_\mathscr{S}$.*

Corollary 2.5 *A graph $H = (V, E + F)$ is a minimal triangulation of $G = (V, E)$ iff H is chordal and there is a set $\mathscr{S}$ of pairwise non-crossing minimal separators of G such that $H = G_\mathscr{S}$.*

Lemma 2.6 ([25]) *Let $G = (V, E)$ be a graph, let $\mathscr{S}$ and $\mathscr{S}'$ be sets of pairwise non-crossing minimal separators of G and $G_\mathscr{S}$, respectively. Then $\mathscr{S} \cup \mathscr{S}'$ is a set of pairwise non-crossing minimal separators of G and $G_\mathscr{S}$.*

Lemma 2.7 ([25]) *Let $G = (V, E)$, be a graph and $\mathscr{S}$ a set of pairwise non-crossing minimal separators of G. Then any minimal triangulation of $G_\mathscr{S}$ is a minimal triangulation of G.*

We will also use the notion of *substar*, which was introduced by Lekkerkerker and Boland [20] in connection with their characterization of chordal graphs.

Definition 2.8 ([20]) *Given a graph* $G = (V, E)$ *and a connected subset* X *of* V, *the* substars *of* X *are the neighborhoods of the connected components of* $G(V - N[X])$. *Note that each substar of* X *is included in* $N(X)$.

When X is reduced to a single vertex x, we will say substar of x for substar of $\{x\}$. In fact, although Lekkerkerker and Boland seemed not to be aware of this, the set of substars of some vertex x is exactly the set of minimal separators included in the neighborhood of x. Ohtsuki et al. [24] gave the following characterization of a meo.

Characterization 2.9 *[24] An ordering α of V is a meo of a graph G if and only if for every integer k between 1 and n, every fill edge added at step k of EG on (G, α) has both endpoints in some common substar of v_k in G_α^k.*

LB-simpliciality of a vertex was defined in [4] in the following way for more convenient terminology.

Definition 2.10 *A vertex x is* LB-simplicial *if every substar of x is a clique.*

This was implicitly used by [20] to characterize chordal graphs as graphs in which every vertex is LB-simplicial, but the notion of substar is also very useful in the context of minimal triangulation, because it provides a fast and easy way of repeatedly finding sets of pairwise non-crossing minimal separators when running the Saturation Algorithm. This is fully described in [4], with in particular the following lemma:

Lemma 2.11 *([4]) The substars of a vertex x in a graph G are pairwise non-crossing minimal separators of G.*

This resulted in the following algorithm for computing minimal triangulations.

Algorithm LB-Triang
Input: A graph $G = (V, E)$, and an ordering α of the vertices in G.
Output: A minimal triangulation G_α^{LB} of G.
$G_\alpha^{LB,1} \leftarrow G$;
for $k = 1$ **to** n **do**
    Let F be the set of edges necessary to saturate the substars of v_k in $G_\alpha^{LB,k}$;
    $G_\alpha^{LB,k+1} \leftarrow G_\alpha^{LB,k} + F$;
$G_\alpha^{LB} \leftarrow G_\alpha^{LB,n+1}$;

We recall here some properties of Algorithm LB-Triang proved in [4]. Items a) to e) of Property 2.12 respectively are or immediately follow from Lemma 4.5 and its proof, Invariant 4.7, Lemma 5.2, Theorem 5.6's proof, Theorem 5.6 and Corollary 5.7 of [4].

Property 2.12 *[4] Let $G = (V, E)$ be a graph, α be an ordering of V and k be an integer between 1 and n.*
a) v_k has the same neighborhood and substars in $G_\alpha^{LB,k}$ and in G_α^{LB}.
b) For any integer i between 1 and k, v_i is LB-simplicial in $G_\alpha^{LB,k+1}$.
c) Removing LB-simplicial vertices from G does not modify the fill computed by LB-Triang on (G, α).
d) Any fill edge added at step k of LB-Triang on (G, α) is an edge of G_α^{k+1}.
e) $G_\alpha^{LB} \subseteq G_\alpha^+$, and α is a meo of G if and only if $G_\alpha^{LB} = G_\alpha^+$.

For an efficient implementation of Algorithm LB-Triang, a useful structure called *tree decomposition* was used. We will also use it here for the implementation of our algorithms.

Finally, we will mention a result which will be used to prove some of our results. Given a chordal graph G and a peo α of G, Rose [28] showed that every minimal separator of G appears as the neighborhood of a vertex to be processed at some step of EG with ordering α on G. However, there may be some steps such that the neighborhood of the processed vertex of that step is not a minimal separator of G.

Theorem 2.13 ([28]) *Let G be a chordal graph and let $\alpha = (v_1, v_2, ..., v_n)$ be a peo of G. Consider any minimal separator S of G. Then $S = N_{G_\alpha^k}(v_k)$ for some k between 1 and n.*

3 EG defines a partial minimal triangulation

We will now examine how EG behaves with respect to the minimal separators of the graph which is to be triangulated. We will first extend the definition of a substar of G given in Section 2 to that of a *substar of (G, α)*.

Definition 3.1 *Given $(G = (V, E), \alpha)$, we will say that a set $S \subset V$ of vertices is a* substar *of (G, α) if there is some step k of EG such that S is a substar of v_k in G_α^k, which will be referred to as a* substar defined at step k of EG.

Clearly, during the execution of the EG, at each step k, making the currently processed vertex v_k simplicial will saturate these substars, and may also add some extraneous edges which do not have both endpoints in some common substar, so that two kinds of edges can be added:

- Edges which have both endpoints in some common substar defined at step k. We will refer to these edges as *substar fill edges*.

- Edges which do not have both endpoints in some common substar defined at step k. We will refer to these as *extraneous edges*.

In Section 2, we mentioned that in G and for a given vertex v, the substars of v are the minimal separators included in the neighborhood of v. One of our most interesting discoveries is that, in fact, *all* the substars defined by EG are minimal separators of the input graph, whether or not extraneous edges have been added at earlier steps. This fact is stated in Theorem 3.3, and its proof is based on the following theorem, which is interesting in its own right, as it describes a strong correspondence between the structures of G and G_α^k.

Theorem 3.2 *Given $(G = (V, E), \alpha)$ and an integer $k \in [1, n]$, let $G_\alpha^k = (V_\alpha^k, E_\alpha^k)$ and $S \subseteq V_\alpha^k$. The connected components of $G_\alpha^k(V_\alpha^k - S)$ are the sets $C \cap V_\alpha^k$ where C is a connected component of $G(V - S)$ such that $C \cap V_\alpha^k \neq \emptyset$, with the same neighborhoods, i.e. $N_{G_\alpha^k}(C \cap V_\alpha^k) = N_G(C)$.*

Proof. Let $\mathscr{C}_G(S)$ denote the set of connected components of $G(V - S)$. Let $S \subseteq V_\alpha^k$. We have to prove that $\mathscr{C}_{G_\alpha^k}(S) = \{C \cap V_\alpha^k, \ C \in \mathscr{C}_G(S) \mid C \cap V_\alpha^k \neq \emptyset\}$ and $\forall C \in \mathscr{C}_G(S)$ such that $C \cap V_\alpha^k \neq \emptyset$, $N_{G_\alpha^k}(C \cap V_\alpha^k) = N_G(C)$. Let $C \in \mathscr{C}_G(S)$

such that $C \cap V_\alpha^k \neq \emptyset$ and let $C' = C \cap V_\alpha^k$. Let us show that $C' \in \mathscr{C}_{G_\alpha^k}(S)$ and $N_{G_\alpha^k}(C') = N_G(C)$. $G_\alpha^k(C')$ is connected because for any vertices x and y in C', there is a path P in $G(C)$ between x and y, and by Lemma 2.1, the subsequence of P containing only the vertices belonging to V_α^k is a path in $G_\alpha^k(C')$ between x and y. Let us show that $N_{G_\alpha^k}(C') \subseteq N_G(C)$. Let $x \in N_{G_\alpha^k}(C')$ and $y \in C'$ such that $xy \in E_\alpha^k$. By Lemma 2.1, there is a path in G between x and y, all intermediate vertices of which belong to $V - V_\alpha^k$, and therefore belong to $V - S$ and consequently belong to C, so $x \in N_G(C)$. Let us show that $N_G(C) \subseteq N_{G_\alpha^k}(C')$. Let $x \in N_G(C)$ and $y \in C$ such that $xy \in E$. As $C' \neq \emptyset$, we may choose $z \in C'$. Let P be a path in $G(C)$ between y and z and let z' be the first vertex of P from y belonging to V_α^k. Vertex $z' \in C'$, and by Lemma 2.1 $xz' \in E_\alpha^k$, so $x \in N_{G_\alpha^k}(C')$. Thus $N_{G_\alpha^k}(C') = N_G(C)$. As $C' \neq \emptyset$, $C' \subseteq V_\alpha^k - S$, $G_\alpha^k(C')$ is connected and $N_{G_\alpha^k}(C') = N_G(C) \subseteq S$, it follows that $C' \in \mathscr{C}_{G_\alpha^k}(S)$. Therefore, $\{C \cap V_\alpha^k,\ C \in \mathscr{C}_G(S) \mid C \cap V_\alpha^k \neq \emptyset\} \subseteq \mathscr{C}_{G_\alpha^k}(S)$. As $\cup_{C \in \mathscr{C}_G(S)}(C \cap V_\alpha^k) = V_\alpha^k - S$, the reverse inclusion holds too. $\blacksquare$

Theorem 3.3 *Every substar of (G, α) is a minimal separator of G.*

Proof. Let S be a substar defined at step k. S is a minimal separator of G_α^k, and by Theorem 3.2, there are at least as many full components of S in G as in G_α^k. So S is also a minimal separator of G. $\blacksquare$

Theorem 3.4 *The set of substars of (G, α) forms a set of pairwise non-crossing minimal separators of G.*

Proof. Let S and S' be two substars of (G, α) defined at steps k and k' respectively, with $k \leq k'$. By Theorem 3.3, they are both minimal separators of G. Let us show that they are non-crossing in G. If $k = k'$ then they are non-crossing in G_α^k by Lemma 2.11, so they are non-crossing in G by Theorem 3.2. We suppose now that $k < k'$. S is a clique of G_α^{k+1} and $S' \subseteq V_\alpha^{k+1}$, so there is a connected component C of $G_\alpha^{k+1}(V_\alpha^{k+1} - S')$ such that $S \subseteq S' \cup C$. By Theorem 3.2, there is a connected component C' of $G(V - S')$ containing C, so $S \subseteq S' \cup C'$. Hence S and S' are non-crossing in G. $\blacksquare$

Note that this theorem does not guarantee that the set of substars defines a set of pairwise non-crossing minimal separators which is maximal. For instance, for any non complete graph G, if v_1 is a universal vertex of G, then there is no substar of (G, α), whereas G has at least one minimal separator. A less trivial counterexample is given in Figure 2(b) of Example 3.5.

Example 3.5 *Figure 2 shows two executions of EG on graph G. A graph G and an ordering α are given in (a). The minimal separators of G are: $\{1, 3\}$, $\{3, 5\}$, $\{3, 7\}$, $\{1, 4, 6\}$, $\{1, 4, 7\}$, $\{1, 4, 8\}$, $\{3, 5, 7\}$, $\{4, 5, 7\}$, $\{4, 5, 8\}$, $\{4, 6, 8\}$, $\{3, 4, 6\}$. We now demonstrate the execution of EG on (G, α) resulting in the graph shown in (b).***Step 1:** $N(1) = \{2, 3, 5\}$, $C_1 = \{4, 6, 7, 8\}$, $N(C_1) = \{3, 5\}$; substar fill edge 35 and extraneous edge 25 are added. **Step 2:** $N(2) = \{3, 5\}$; $C_2 = \{4, 6, 7, 8\}$, $N(C_2) = \{3, 5\}$; 2 is simplicial, so no edge is added. **Step 3:** $N(3) = \{4, 5, 8\}$, $C_3 = \{6, 7\}$, $N(C_3) = \{4, 5, 8\}$; substar fill edges 48 and 58 are added. **Step 4:** $N(4) = \{5, 6, 7, 8\}$; 4 is universal, so no component is defined; extraneous edges 57 and 68 are added; the remaining graph becomes a clique;*
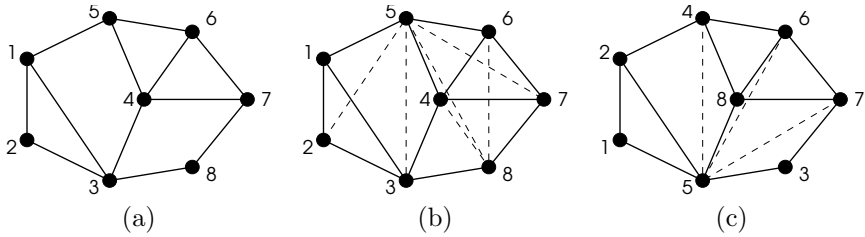
Figure 2: Two executions of EG on the same graph G with (b) an arbitrary ordering, and (c) an MD ordering. (b) is a triangulation of G which is not minimal, (c) is in this case a minimum triangulation of G.

no further edge is added. The set of substars of (G, α) is thus $\{\{3, 5\}, \{4, 5, 8\}\}$, which is a set of pairwise non-crossing minimal separators of G, but not a maximal one as $\{\{3, 5\}, \{4, 5, 8\}, \{4, 5, 7\}\}$ and $\{\{3, 5\}, \{4, 5, 8\}, \{4, 6, 8\}\}$ are also sets of pairwise non-crossing minimal separators of G. If only substar fill edges are preserved, a chordless cycle 5678 remains in the graph thus obtained. In order to saturate a maximal set of pairwise non-crossing minimal separators of G, $\{4, 5, 7\}$ or $\{4, 6, 8\}$ should also be saturated. On the same graph, MD yields a minimal (and even minimum) triangulation, as shown in (c).

Thus by Theorem 3.4 and Lemma 2.7, a minimal triangulation of a given graph G can be computed as follows. Run EG on (G, α) where α is an ordering of V, then remove from G_α^+ all fill edges that do not appear within substars (i.e., the extraneous fill edges). As Example 3.5 shows, the resulting graph is not necessarily chordal, but any minimal triangulation of it will be a minimal triangulation of G. Furthermore, this minimal triangulation will be a subgraph of G_α^+, as we will show in Section 5. In Section 4, we will show how to compute this partial minimal triangulation both in $O(nm)$ sequential time and in $O(\log^2 n)$ parallel time on $O(nm)$ processors, and in Section 5 we will show how to complete it to a minimal triangulation of G in $O(\log^3 n)$ parallel time on $O(nm)$ processors.

We would like to end this subsection with a discussion on the robustness of EG regarding the process of defining non-crossing minimal separators of G. If, during the EG process, no extraneous edge is added, then the triangulation which is computed is minimal. However, due to Theorem 3.4, even when extraneous edges have been added, all substar fill edges added later belong to a set of pairwise non-crossing minimal separators of G, and therefore to a minimal triangulation of G. Thus if only a few extraneous edges are added during EG process, they will not destroy the property that all the substar fill edges are "useful" edges and that these few extraneous edges are the only "unnecessary" edges introduced. This makes EG a fault-tolerant procedure.

3.1 Using G_α^+ to compute the substars of (G, α).

In this subsection, we will show that it is not necessary to compute the substars during the course of EG on (G, α). We can indeed compute the substars from only G and the filled graph G_α^+. This is interesting since given (G, α), the filled graph G_α^+ can be computed in linear time in the size of G_α^+ as described by

Tarjan and Yannakakis in [32], whereas EG requires $O(n^3)$ time. We will look at the minimal separators of G_α^+, and show how to split these into the desired substars of (G, α). The results of this subsection are needed for the $O(nm)$ time implementation that will be given in the next section.

In the following, we suppose that a graph $G = (V, E)$ and an ordering $\alpha = (v_1, v_2, ..., v_n)$ of V are given.

Lemma 3.6 *Let S be a minimal separator of G_α^+. Then there are an integer k between 1 and n and a full component C_0 of S in G such that $S = N_{G_\alpha^k}(v_k)$ and $C_0 \cap V_\alpha^k = \{v_k\}$.*

Proof. As α is a peo of G_α^+, by Theorem 2.13 there is an integer k between 1 and n such that $S = N_{(G_\alpha^+)_\alpha^k}(v_k)$, and $(G_\alpha^+)_\alpha^k = G_\alpha^+(V_\alpha^k)$. Since we also have $N_{G_\alpha^+(V_\alpha^k)}(v_k) = N_{G_\alpha^k}(v_k)$, we get $S = N_{G_\alpha^k}(v_k)$. As $\{v_k\}$ is a full component of S in G_α^k, by Theorem 3.2 there is a full component C_0 of S in G such that $C_0 \cap V_\alpha^k = \{v_k\}$. ∎

Definition 3.7 *A subset S of V is a split-minsep of (G, α) if there is a minimal separator S' of G_α^+ and a full component C_0 of S' in G such that S is a substar of C_0 in G (S is said to be derived from S').*

It follows from Lemma 3.6 that every minimal separator of G_α^+ can be split into split-minseps of (G, α).

Remark 3.8 *In the definition of a split-minsep of (G, α), we can moreover assume that there is an integer k between 1 and n such that $S' = N_{G_\alpha^k}(v_k)$ and $C_0 \cap V_\alpha^k = \{v_k\}$.*

Proof. Let S be a split-minsep of (G, α). Let S' be a minimal separator of G_α^+ and C_0' be a full component of S' in G such that S is a substar of C_0' in G. Let C be a component of $G(V - N_G[C_0'])$ defining S, i.e. such that $S = N_G(C)$. By Lemma 3.6 there is an integer k between 1 and n and a full component C_0 of S' in G such that $S' = N_{G_\alpha^k}(v_k)$ and $C_0 \cap V_\alpha^k = \{v_k\}$. Then S is also a substar of C_0 in G: it is evident if $C \neq C_0$, otherwise S is the substar of C_0 defined by C_0', as $S = N_G(C) = N_G(C_0) = S' = N_G(C_0')$, with $C_0 = C \neq C_0'$. ∎

Lemma 3.9 *Let C_0 be a non-empty subset of V such that $G(C_0)$ is connected. Then the substars of C_0 in G are exactly the minimal separators of G included in $N_G(C_0)$.*

Proof. Let S be a substar of C_0 in G. Then S is the neighborhood of some connected component C of $G(V - N[C_0])$. So $S \subseteq N(C_0)$, and as C and the component of $G(V - S)$ containing C_0 are two distinct full components of S in G, S is a minimal separator of G.

Conversely let S be a minimal separator of G included in $N(C_0)$. Then there is a component C_0' of $G(V - S)$ containing $N[C_0] - S$. Let C' be a full component of S in G different from C_0'. As $C' \subseteq V - N[C_0]$, C' is a subset of some component C of $G(V - N[C_0])$. But as $C \subseteq V - N[C_0] \subseteq V - S$, C is a subset of some component of $G(V - S)$. Hence $C = C'$, $S = N(C') = N(C)$ and S is a substar of C_0 in G. ∎

10

Lemma 3.10 *A subset S of V is a split-minsep of (G, α) if and only if it is a minimal separator of G included in some minimal separator of G_α^+.*

Proof. By Lemma 3.6 every minimal separator S of G_α^+ is in the form $N_G(C_0)$, where C_0 is a non-empty subset of V such that $G(C_0)$ is connected. We conclude with Lemma 3.9. ∎

Lemma 3.11 *Let k be an integer between 1 and n, let $S \subseteq V_\alpha^k$ and let C_0 be a full component of S in G such that $C_0 \cap V_\alpha^k = \{v_k\}$. Then C_0 is also a full component of S in G_α^+.*

Proof. As G is a subgraph of G_α^+, C_0 is a non-empty connected subset of vertices in G_α^+ with $S \subseteq N_{G_\alpha^+}(C_0)$. It remains to show that $N_{G_\alpha^+}(C_0) \subseteq S$. By Theorem 3.2 every fill edge having one of its endpoints in C_0 inserted before or at step k of EG has its other endpoint in $N_G[C_0]$, and no fill edge having one of its endpoints in C_0 can be inserted after step k since C_0 is eliminated. Hence $N_{G_\alpha^+}(C_0) \subseteq S$. ∎

Lemma 3.12 *Let k be an integer between 1 and n, let $S \subseteq V_\alpha^k$ and let C_0 be a full component of S in G such that $C_0 \cap V_\alpha^k = \{v_k\}$. Then any substar of C_0 in G is a split-minsep of (G, α).*

Proof. Let S_1 be a substar of C_0 in G defined by a component C of $G(V - N_G[C_0])$. By Lemma 3.11 C_0 is also a full component of S in G_α^+. Let S_1' be the substar of C_0 in G_α^+ defined by the component of $G_\alpha^+(V - N_G[C_0])$ containing C. $S_1 \subseteq S_1'$, and by Lemma 3.9 S_1 and S_1' are minimal separators of G and G_α^+ respectively. We conclude with Lemma 3.10. ∎

Theorem 3.13 *The set of all split-minseps of (G, α) is exactly the set of all substars of (G, α).*

Proof. Let S be a split-minsep of (G, α). Let S' be a minimal separator of G_α^+ and C_0 be a full component of S' in G such that S is a substar of C_0 in G. Let C be a component of $G(V - N_G[C_0])$ such that $S = N_G(C)$. By Remark 3.8 we can moreover assume that there is an integer k between 1 and n such that $S' = N_{G_\alpha^k}(v_k)$ and $C_0 \cap V_\alpha^k = \{v_k\}$.
First case : $C \cap V_\alpha^k \neq \emptyset$.
By Theorem 3.2, S is the substar of (G, α) defined at step k by $C \cap V_\alpha^k$.
Second case : $C \cap V_\alpha^k = \emptyset$.
Let $k' = max\{\alpha(v),\ v \in C\}$. $k' < k$, so $S \subseteq V_\alpha^{k'}$ and $C_0 \cap V_\alpha^{k'} \neq \emptyset$. By Theorem 3.2 $S = N_{G_\alpha^{k'}}(v_{k'})$ and S is the substar of (G, α) defined at step k' by the component of $G_\alpha^{k'}(V_\alpha^{k'} - S)$ containing $C_0 \cap V_\alpha^{k'}$.
Conversely, let S be a substar of (G, α) defined at step k, and let $S_k = N_{G_\alpha^k}(v_k)$. By Theorem 3.2 there is a full component C_0 of S_k in G such that $C_0 \cap V_\alpha^k = \{v_k\}$ and such that S is a substar of C_0 in G. By Lemma 3.12 S is a split-minsep of (G, α). ∎

Thus, by Theorem 3.13 we do not need to execute EG to compute the set of substars of (G, α). It is sufficient to compute the split-minseps derived from all minimal separators of G_α^+.

3.2 Relationships between EG and LB-Triang

LB-Triang resembles EG as it processes the vertices in a given ordering α and adds fill edges in their neighborhood. Furthermore, the processed vertices can be removed from the graph just like in EG [4]. However, whereas EG saturates the whole neighborhood of vertex v_k at step k, LB-Triang saturates only the substars of v_k in the current graph. As LB-Triang always computes a minimal triangulation, it is interesting to compare the substars computed during LB-Triang to the substars of (G, α).
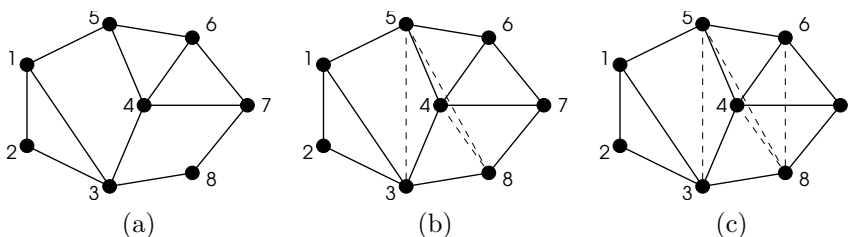

Figure 3: (a) Graph G with ordering α, (b) graph $G_{\mathscr{S}}$, where $\mathscr{S}$ is the set of substars of (G, α) and (c) graph G_α^{LB}. (b) is a partial minimal triangulation of G, (c) is a minimal triangulation of G.

Example 3.14 *Consider graph G and the ordering α of Figure 2. The graph $G_{\mathscr{S}}$ obtained from G by saturating the substars of (G, α) and the graph G_α^{LB} are shown in Figure 3. The substars of (G, α) are: $\{3, 5\}$, defined for the first time at Step 1 (and redefined at Step 2) and $\{4, 5, 8\}$, defined at Step 3. They are also substars defined by LB-Triang process: $\{3, 5\}$ is also defined for the first time at Step 1 (and redefined at Step 4) and $\{4, 5, 8\}$ is also defined for the first time at Step 3 (and redefined at step 6). Note that LB-Triang also defines the substars $\{1, 3\}$, $\{4, 6, 8\}$ (adding the fill edge 68 at Step 5) and $\{3, 4, 5\}$ that are not substars of (G, α).*

We have the following result.

Theorem 3.15 *Let S be a substar of (G, α) defined for the first time at step k. Then S is a substar of v_k defined by LB-Triang process on (G, α) for the first time at step k.*

Proof. Let $G = (V, E)$ be a graph, and let $\alpha = (v_1, ..., v_n)$ be an ordering of V. For any k between 1 and $n + 1$, let $H_k = G_\alpha^{LB,k}$. We will use the following Lemmas. Lemma 3.16 is a well-known property of chordal graphs, and Lemma 3.17 easily follows from the definition of an LB-simplicial vertex and immediately follows from Lemma 5.1 of [4].

Lemma 3.16 *Let C_0 be a non-empty subset of V. If G is chordal, $G(C_0)$ is connected and $N(C_0)$ is a clique of G then there is a vertex v of C_0 such that $N(C_0) \subseteq N(v)$.*

Lemma 3.17 *A vertex v of G is LB-simplicial in G if and only if v belongs to no chordless cycle in G of length at least 4.*

Let S be a substar of (G, α) defined for the first time at step k by a component C of $G_\alpha^k(V_\alpha^k - S_k)$, where $S_k = N_{G_\alpha^k}(v_k)$. By Theorem 3.2 there is a full component C_0 of S_k in G such that $C_0 \cap V_\alpha^k = \{v_k\}$ and there is another component C' of $G(V - S_k)$ such that $C' \cap V_\alpha^k = C$ and $N_G(C') = S$. For any i between 1 and $k + 1$, C_0 and C' are also components of $H_i(V - S_k)$ with the same neighborhoods, i.e. $N_{H_i}(C_0) = S_k$ and $N_{H_i}(C') = S$. We prove this for C_0 (the argument is similar for C'). $H_i(C_0)$ is connected and $S_k \subseteq N_{H_i}(C_0)$ because $G \subseteq H_i$, and $N_{H_i}(C_0) \subseteq S_k$ because every edge of H_i having one of its endpoints in C_0 is an edge of G_α^j for some $j \leq i$ by Property 2.12 d), and therefore has its other endpoint in $N_G[C_0] = C_0 \cup S_k$ by Theorem 3.2. In particular, for any i between 1 and k, S is a substar of C_0 in H_i and of $C_0 \cap V_\alpha^i$ in G_α^i, so by Lemma 3.9 S is a minimal separator of both H_i and G_α^i.

Let us show that S is a clique of H_{k+1}. We suppose for contradiction that it is not the case. Let y, z be non-adjacent vertices of S in H_{k+1}, let P_1 and P_2 be chordless paths in H_{k+1} between y and z whose internal vertices are in C_0 and C' respectively, and let Q be the cycle formed by P_1 and P_2. Q is a chordless cycle in H_{k+1} of length at least 4 containing some vertex v of C_0. As $C_0 \cap V_\alpha^k = \{v_k\}$, $v = v_i$ for some $i \leq k$. By Property 2.12 b), v is LB-simplicial in H_{k+1}, and by Lemma 3.17 v belongs to no chordless cycle in H_{k+1} of length at least 4, a contradiction.

Now consider the graph $H = H_{k+1}(C_0 \cup S)$. C_0 is non-empty, $H(C_0)$ is connected, $N_H(C_0) = S$ is a clique of H and H is chordal since by the preceding argument no vertex of C_0 belongs to a chordless cycle in H_{k+1}, and therefore in H, of length at least 4, and S is a clique of H. So by Lemma 3.16 there is a vertex v of C_0 such that $S \subseteq N_H(v)$, and therefore there is some $i \leq k$ such that $S \subseteq N_H(v_i)$. Let i be the smallest such integer, and let us show that $i = k$. By Property 2.12 d) no fill edge containing v_i can be added after step i of LB-Triang, so $S \subseteq N_H(v_i) \subseteq N_{H_{k+1}}(v_i) = N_{H_i}(v_i)$, and by Property 2.12 d) again $S \subseteq N_{H_i}(v_i) \cap V_\alpha^i \subseteq N_{G_\alpha^i}(v_i)$. As S is minimal separator of G_α^i included in $N_{G_\alpha^i}(v_i)$, by Lemma 3.9 S is a substar of (G, α) defined at step i, and since S is defined for the first time at step k, $i = k$. As S is minimal separator of H_k included in $N_{H_k}(v_k)$, by Lemma 3.9 S is a substar of v_k defined by LB-Triang process at step k, and it is defined for the first time at step k since for any $i < k$, $S \not\subseteq N_H(v_i)$ and therefore $S \not\subseteq N_{H_i}(v_i)$. ∎

Theorem 3.18 *Let $\mathscr{S}$ be the set of substars of (G, α). $G_{\mathscr{S}} \subseteq G_\alpha^{LB} \subseteq G_\alpha^+$ and if α is a meo of G then $G_{\mathscr{S}} = G_\alpha^{LB} = G_\alpha^+$.*

Proof. By Theorem 3.15 $G_{\mathscr{S}} \subseteq G_\alpha^{LB}$, by Property 2.12 e) $G_\alpha^{LB} \subseteq G_\alpha^+$, and if α is a meo of G then by Characterization 2.9 every fill edge in G_α^+ has both endpoints in some common substar of (G, α), so $G_\alpha^+ \subseteq G_{\mathscr{S}}$, and therefore $G_{\mathscr{S}} = G_\alpha^{LB} = G_\alpha^+$. ∎

Note that if α is not a meo of G then by Property 2.12 e) $G_\alpha^{LB} \subset G_\alpha^+$, but $G_{\mathscr{S}}$ may be equal to G_α^{LB}. In particular, if G is chordal and α is not a peo of G then $G = G_{\mathscr{S}} = G_\alpha^{LB} \subset G_\alpha^+$

4 Computing the partial minimal triangulation efficiently

In this section we will give the implementation details of how to compute and saturate the substars efficiently. More formally, given (G, α), we want to compute $G_{\mathscr{S}}$, where $\mathscr{S}$ is the set of substars of (G, α). We give both a sequential and a parallel implementation of this process. In both implementations, we will use the definition of a substar of (G, α) as a split-minsep of (G, α) described in Subsection 3.1.

4.1 Partial minimal triangulation in sequential $O(nm)$ time

In this subsection, we will show that the set $\mathscr{S}$ of all substars of (G, α) and the graph $G_{\mathscr{S}}$ can be computed in $O(nm)$ time. For this second result, we will use a data structure introduced in [4] to implement LB-Triang and we will prove more generally that for any given set $\mathscr{S}$ of pairwise non-crossing minimal separators of G, the graph $G_{\mathscr{S}}$ can be computed in $O(nm)$ time.

Theorem 4.1 *Given (G, α), the set $\mathscr{S}$ of all substars of (G, α) can be computed in $O(nm)$ time.*

Proof. By Theorem 3.13 it is sufficient to compute the set of split-minseps of (G, α). Computing G_{α}^{+} can be done in time $O(n + m')$, where m' is the number of edges of G_{α}^{+}, as described in [32]. Since it is a chordal graph, it has at most n minimal separators (by Theorem 2.13) which can be computed in a global $O(n + m')$ time [7]. For any minimal separator S' of G_{α}^{+}, the connected components of $G(V - S')$ and their neighborhoods can be computed in $O(n + m)$ time. The split-minseps derived from S' are these neighborhoods, except S' itself if it has only one full component in G. We use a Search/Insert data structure similar to that used in [4] to eliminate duplicates with no extra time cost, and we obtain a global $O(nm)$ time bound. ∎

It is less easy to prove that saturating the substars of (G, α) can also be done in $O(nm)$ time. With a straightforward implementation, saturating a substar takes $O(m')$ time, where m' is the number of edges of $G_{\mathscr{S}}$. As the number of substars of (G, α) is bounded by n (by Lemma 4.2 below) saturating all substars takes $O(nm')$ time. To achieve $O(nm)$ time, we will use the *tree decomposition* data structure used in [4] to compute G_{α}^{LB} and will extend it to the more general problem of computing $G_{\mathscr{S}}$ for any set $\mathscr{S}$ of pairwise non-crossing minimal separators of G.

Lemma 4.2 *For any set $\mathscr{S}$ of pairwise non-crossing minimal separators of G, $|\mathscr{S}| \leq n$.*

Proof. Let $\mathscr{S}'$ be a maximal set of pairwise non-crossing minimal separators of G containing $\mathscr{S}$. $G_{\mathscr{S}'}$ is chordal by Theorem 2.4, and by Lemma 2.6 $\mathscr{S}'$ is also a set of minimal separators of $G_{\mathscr{S}'}$. As by Theorem 2.13 chordal graph $G_{\mathscr{S}'}$ has at most n minimal separators, we obtain $|\mathscr{S}| \leq |\mathscr{S}'| \leq n$. ∎

Theorem 4.3 *Given a graph G and a set $\mathscr{S}$ of pairwise non-crossing minimal separators of G, the partial minimal triangulation $G_{\mathscr{S}}$ of G can be computed in $O(nm)$ time.*

14

Proof. We generalize the implementation of LB-Triang described in [4] using a data structure called *tree decomposition by minimal separators*. We recall the definition of this structure.

A tree structure on G is a structure $TS = (T, (X_u)_{u \in U_T}, (S_{uv})_{uv \in E_T})$, where $T = (U_T, E_T)$ is a tree, X_u is a subset of V for each u in U_T and S_{uv} is a subset of V for each uv in E_T.

We note:
- $\forall x \in V$, $U_x = \{u \in U_T \mid x \in X_u\}$ and $T_x = T(U_x) = (U_x, E_x)$,
- $\forall C \subseteq V$, $T_C = (\cup_{x \in C} U_x, \cup_{x \in C} E_x) = (U_C, E_C)$,
- $\forall uv \in E_T$, T_{uv} and T_{vu} are the two connected components of $T' = (U_T, E_T \setminus \{uv\})$ respectively containing u and v.

A tree decomposition of G is a tree structure TS on G such that:
a) $\cup_{u \in U_T} X_u = V$,
b) $\forall xy \in E$, $\exists u \in U_T \mid x, y \in X_u$ (i.e. $U_x \cap U_y \neq \emptyset$),
c) $\forall x \in V$, T_x is a subtree of T,
d) $\forall uv \in E_T$, $S_{uv} = X_u \cap X_v$.

If TS is a tree decomposition of G, and C is a connected subset of V then T_C is a subtree of T [4].

A tree decomposition of G by minimal separators is a tree decomposition TS of G verifying the extra property:
e) $\forall uv \in E_T$, $\exists C_1, C_2$ full components of S_{uv} in $G \mid T_{C_1} \subseteq T_{uv}$ and $T_{C_2} \subseteq T_{vu}$.

The graph $G_{\mathscr{S}}$ can be computed as follows. First a tree decomposition of G by minimal separators TS is computed, where the edges of T contain the elements of $\mathscr{S}$ (or more accurately, the elements of a subset $\mathscr{S}'$ of $\mathscr{S}$ such that every separator in $\mathscr{S}$ is a subset of a separator in $\mathscr{S}'$, so that $G_{\mathscr{S}} = G_{\mathscr{S}'}$). Then, using Algorithm Neighbors described in [4], we compute the closed neighborhood in $G_{\mathscr{S}'}$, and therefore in $G_{\mathscr{S}}$, of every vertex of G. The whole process is described in the following variant NonCrossing-Treedecomp of Algorithm LB-Treedecomp presented in [4]. We refer the reader to [4] for the implementations of Algorithms Neighbors, InitVariables and UpdateVariables. For every vertex x of G, variable $u(x)$ contains an arbitrary node of T_x.

Algorithm Neighbors
Input: A graph $G = (V, E)$, a vertex x of G, and a tree decomposition $TS = (T = (U_T, E_T), (X_u)_{u \in U_T}, (S_{uv})_{uv \in E_T})$ of G.
Output: the set $N_{G'}[x]$, where G' is the graph obtained from G by saturating the elements of the sets S_{uv} for each uv in E_T.

Algorithm NonCrossing-Treedecomp
Input: A graph $G = (V, E)$,
and a set $\mathscr{S}$ of pairwise non-crossing minimal separators of G.
Output: the graph $G_{\mathscr{S}}$.
$T \leftarrow (\{u_0\}, \emptyset)$; $\quad X_{u_0} \leftarrow V$;
InitVariables();
foreach $S \in \mathscr{S}$ **do**
$\qquad$ Compute two distinct full components C_1 and C_2 of S in G;
$\qquad$ Pick any vertex c_1 in C_1 and any vertex c_2 in C_2;
$\qquad$ Compute the path $P = (u(c_1) = u_0, u_1, ..., u_p = u(c_2))$
in T between $u(c_1)$ and $u(c_2)$;
$\qquad$ $i \leftarrow 0$:

15

```
    foreach s ∈ S do
        while s ∉ X_{u_i} do i ← i + 1;
    w ← u_i;
    if X_w ∩ C_2 ≠ ∅ then
        Split w into w_1 and w_2;
        X_{w_1} ← X_w ∩ (C_1 ∪ S);   X_{w_2} ← X_w \ C_1;
        Replace each edge wv by w_1v with S_{w_1v} = S_{wv} if S_{wv} ⊆ C_1 ∪ S
        and by w_2v with S_{w_2v} = S_{wv} otherwise;
        Add edge w_1w_2;   S_{w_1w_2} ← S;
        UpdateVariables();
G_𝒮 ← (V, ∅) ;
foreach x ∈ V do
    N_{G_𝒮}[x] ← Neighbors(G, x, TS);
```

The main difference between Algorithms LB-Treedecomp and NonCrossing-Treedecomp is the way in which the node w to be split is searched for in T. We first prove the following Lemma, which generalizes Lemmas 7.17 to 7.20 from [4].

Lemma 4.4 *Let $S \in \mathscr{S}$. We suppose that TS is a tree decomposition of G just before processing S in an execution of NonCrossing-Treedecomp. Then when processing S a node $w = u_i$ of T is computed, with $S \subseteq X_w$, $X_w \cap C_1 \neq \emptyset$ and if $X_w \cap C_2 = \emptyset$ then $S \subseteq S_{u_i u_{i+1}}$.*

Proof. (of Lemma 4.4) Just before processing S, as TS is a tree decomposition of G, T_{C_1} and T_{C_2} are subtrees of T. Moreover $|U_{C_1} \cap U_{C_2}| \leq 1$ (otherwise T_{C_1} and T_{C_2} would have a common edge uv with $S_{uv} \in \mathscr{S}$, $S_{uv} \cap C_1 \neq \emptyset$ and $S_{uv} \cap C_2 \neq \emptyset$, so S_{uv} and S would be crossing elements of $\mathscr{S}$). Let Q be the path between T_{C_1} and T_{C_2} in T. Q is in the form $(u_{i_1}, u_{i_1+1}, ..., u_{i_2})$ with $0 \leq i_1 \leq i_2 \leq p$. For any $s \in S$, $s \in N_G(C_1) \cap N_G(C_2)$, so $U_s \cap U_{C_1} \neq \emptyset$ and $U_s \cap U_{C_2} \neq \emptyset$, and therefore the set of integers j in $[0, p]$ such that $s \in X_{u_j}$ is an interval containing i_1 and i_2. It follows that the execution of the loop 'foreach $s \in S$ do' will terminate with $0 \leq i \leq i_1$ and $S \subseteq X_i$. So $S \subseteq X_w$ and as $0 \leq i \leq i_1$, u_i is a node of T_{C_1} and therefore $X_w \cap C_1 \neq \emptyset$. If $X_w \cap C_2 = \emptyset$ then $i < i_2$ and therefore $S \subseteq X_{u_i} \cap X_{u_{i+1}} = S_{u_i u_{i+1}}$. ∎

By Lemma 4.4, if TS is a tree decomposition of G just before processing S then either $X_w \cap C_2 \neq \emptyset$ and then w is split into w_1 and w_2 and the edge $w_1 w_2$ with $S_{w_1 w_2} = S$ is created, or S is a subset of an already processed minimal separator in $\mathscr{S}$. Thus the set $\mathscr{S}'$ of minimal separators contained in the edges of the final tree T is a subset of $\mathscr{S}$ such that $G_\mathscr{S} = G_{\mathscr{S}'}$, and Algorithm Neighbors correctly computes the closed neighborhood of every vertex of G in $G_\mathscr{S}$, provided that TS is a tree decomposition of G at every step. We prove that TS is a tree decomposition of G by minimal separators at every step of the algorithm in the same way as in [4] (Invariant 7.21) since this proof only uses the facts that $\mathscr{S}$ is a set of pairwise non-crossing minimal separators of G and that if the node w computed when processing S is split then $S \subseteq X_w$, $X_w \cap C_1 \neq \emptyset$ and $X_w \cap C_2 \neq \emptyset$, which holds by Lemma 4.4. This completes the proof of correctness of NonCrossing-Treedecomp.

Let us prove $O(nm)$ time bound. We know from [4] that InitVariables requires $O(n)$ time, that splitting a node w, UpdateVariables and Neighbors take $O(m)$

time (using the fact that TS is a tree decomposition of G by minimal separators), and that the data structure used to implement the sets X_u allows to test whether a vertex of G belongs to a set X_u or not in $O(1)$ time. Computing C_1 and C_2 requires $O(n + m)$ time, computing P takes $O(n)$ time (since by Lemma 4.2 $|E_T| \leq |\mathscr{S}| \leq n$), as well as computing w and testing $X_w \cap C_2 \neq \emptyset$. Thus each iteration of each one of the two main **foreach**-loops requires $O(n + m) = O(m)$ time, and since $|\mathscr{S}| \leq n$ we obtain a global $O(nm)$ time bound. $\blacksquare$

Corollary 4.5 *Given (G, α), the partial minimal triangulation $G_{\mathscr{S}}$ of G can be computed in $O(nm)$ time, where $\mathscr{S}$ is the set of all substars of (G, α).*

Proof. By Theorem 4.1 $\mathscr{S}$ can be computed in $O(nm)$ time. By Theorem 3.4 $\mathscr{S}$ is a set of pairwise non-crossing minimal separators of G, so we conclude with Theorem 4.3. $\blacksquare$

Note that the algorithm described by Dahlhaus in [10] as the Tree Splitting Procedure can also be used to compute the set $\mathscr{S}$ of substars of (G, α) and the graph $G_{\mathscr{S}}$ in $O(nm)$ time. This algorithm computes what is called in [10] a *quasi-minimal tree representation* of G. It can be proved that the computed tree is a tree decomposition of G by minimal separators whose edges exactly contain the substars of (G, α). This tree is initialized with a clique tree of G_α^+ and at each step of the algorithm, an edge containing a minimal separator S' of G_α^+ is split into edges containing some split-minseps of (G, α) derived from S'. The graph $G_{\mathscr{S}}$ can be computed from that tree, using Algorithm Neighbors in the same way as in NonCrossing-Treedecomp.

4.2 Parallel partial minimal triangulation

In this subsection, we give a parallel algorithm for computing the substars of (G, α).

Algorithm Parallel Substar Computation (PSC)
Input: A graph $G = (V, E)$ and an ordering $\alpha = (v_1, v_2, ..., v_n)$ of G.
Output: The set of substars of (G, α).
1. Compute the connected components C of $G(\{v_1, v_2, ..., v_i\})$
for all $i \in [1, n - 1]$;
2. Compute $N_G[C]$ for all connected components C computed at Step 1;
3. Compute the connected components K of $G(V - N_G[C])$ for all
   closed neighborhoods $N_G[C]$ computed at step 2;
4. Compute $N_G(K)$ for all connected components K computed at step 3;
5. Eliminate duplicates from the sets $N_G(K)$ of Step 4 by sorting them;
6. Output the remaining sets from Step 5 as the substars of (G, α);

Lemma 4.6 *Algorithm PSC computes the set of all substars of (G, α).*

Proof. By Theorem 3.13 it is sufficient to show that PSC computes the set of all split-minseps of (G, α).
Let S be a set computed by PSC. Let $i \in [1, n - 1]$, let C_0 be a component of $G(\{v_1, v_2, ..., v_i\})$ and K be a component of $G(V - N_G[C_0])$ such that $S = N_G(K)$. Let k be the maximum value of $\alpha(v)$ for $v \in C_0$ and let $S_0 = N_G(C_0)$.

17

$S_0 \subseteq V_\alpha^{i+1} \subseteq V_\alpha^k$, C_0 is a full component of S_0 in G such that $C_0 \cap V_\alpha^k = \{v_k\}$ and S is a substar of C_0 in G, so by Lemma 3.12 S is a split-minsep of (G, α). Conversely let S be a split-minsep of (G, α). By Remark 3.8 there is a minimal separator S' of G_α^+, a full component C_0 of S' in G and an integer k between 1 and n such that S is a substar of C_0 in G, $S' \subseteq V_\alpha^{k+1}$ and $C_0 \cap V_\alpha^k = \{v_k\}$. It follows that C_0 is a component of $G(\{v_1, v_2, ..., v_k\})$, so S is computed by PSC. ∎

Lemma 4.7 *Algorithm PSC runs in $O(\log^2 n)$ parallel time with $O(nm)$ processors on a CREW PRAM.*

Proof. The first step of the algorithm can be done as follows. We give any edge $v_i v_j$ the distance $d(v_i, v_j) = max(i, j)$. Let E_i be the set of edges vw of G with $d(v, w) \leq i$. We call the connected components of E_i the *i-clusters*. By [11], for all i simultaneously, the i-clusters can be determined in $O(\log^2 n)$ time with $O(n + m)$ processors. Observe that the i-clusters are just the connected components of $G(\{v_1, ..., v_i\})$. The second step can be done in $O(\log n)$ time on $O(nm)$ processors, since for each C separately it can be done in logarithmic time with a linear number of processors and the number of C is bounded by n. The third step can be done in $O(\log^2 n)$ time on $O(nm)$ processors, since it can be done in $O(\log^2 n)$ time with $O(n + m)$ processors, for each C separately, using the algorithm of Shiloach and Vishkin [30] and the number of C is bounded by n. The fourth step is analogous to the second step, it requires the same bounds for each C separately, and therefore the same global bounds. The fifth step needs $O(\log n)$ time and $O(n)$ processors if we assume that one comparison needs constant time and a linear number of processors [9]. Here one comparison needs $O(n)$ processors and $O(\log n)$ time on a CREW-PRAM. Therefore the fifth step needs $O(n^2)$ processors and $O(\log^2 n)$ time on a CREW-PRAM. ∎

Theorem 4.8 *Given (G, α), the partial minimal triangulation $G_{\mathscr{S}}$ of G can be computed in parallel $O(\log^2 n)$ time with $O(nm)$ processors on a CREW PRAM, where $\mathscr{S}$ is the set of all substars of (G, α).*

Proof. By Lemmas 4.6 and 4.7, we know how to compute the substars $\mathscr{S}$ of (G, α) in parallel $O(\log^2 n)$ on $O(nm)$ processors. The substar edges can be added to G in parallel $O(\log n)$ time on $O(nm)$ processors, and the result follows. ∎

5 Completing the partial minimal triangulation into a minimal triangulation

Theorem 5.1 *Let $\mathscr{S}$ be the set of substars of (G, α). Then any minimal triangulation H of $G_{\mathscr{S}}$ is a minimal triangulation of G which is a subgraph of G_α^+.*

Proof. Let H be a minimal triangulation of $G_{\mathscr{S}}$. By Theorem 3.4 and Lemma 2.7, H is a minimal triangulation of G. Let us show that H is a subgraph of G_α^+. By Theorem 2.4 there is a set $\mathscr{S}'$ of pairwise non-crossing minimal

separators of $G_\mathscr{S}$ such that $H = (G_\mathscr{S})_{\mathscr{S}'}$. As $G_\mathscr{S}$ is a subgraph of G_α^+, it is sufficient to show that any element of $\mathscr{S}'$ is a clique of G_α^+. Let $T \in \mathscr{S}'$, let u and v be two vertices of T with $\alpha(u) < \alpha(v)$ and let $k = \alpha(u)$ (i.e. $u = v_k$). Let us show that uv is an edge of G_α^+, i.e. $v \in N_{G_\alpha^k}(v_k)$. We assume by contradiction that $v \notin N_{G_\alpha^k}(v_k)$. Let S be the substar of (G, α) defined at step k by the component containing v. S is a minimal separator of G_α^k separating v_k and v. By Theorem 3.2, S is also a minimal separator of G separating the vertices v_k and v of T. So S and T are crossing in G. But as $S \in \mathscr{S}$ and $T \in \mathscr{S}'$, by Lemma 2.6 S and T are non crossing in G, a contradiction. ∎

As a consequence of Theorem 5.1, we can use any minimal triangulation algorithm that we like to compute a minimal triangulation of $G_\mathscr{S}$, which will also be a minimal triangulation of G that is a subgraph of G_α^+.

Since there are several minimal triangulation algorithms with an $O(nm)$ time bound, as LEX M [31], MCSM [3], and LB-Triang [4], the overall time for computing a minimal triangulation through this method is $O(nm')$ where m' is the number of edges of $G_\mathscr{S}$. This gives a new algorithm for solving the sandwiched minimal triangulation problem. Meanwhile, note that the LB-Triang algorithm of [4] already solves this problem directly in $O(nm)$ time. However, there is no efficient parallel algorithm for solving the sandwiched minimal triangulation problem, and our approach results in such a parallel algorithm.

5.1 A parallel algorithm to compute sandwiched minimal triangulations

In 1994, Dahlhaus and Karpinski [12] described a parallel algorithm that computes a minimal triangulation of a given graph in $O(\log^3 n)$ parallel time using $O(nm)$ processors on a CREW PRAM. This algorithm does not solve the sandwiched minimal triangulation problem. However, by Theorem 5.1, a parallel algorithm for solving this problem is the following: given (G, α), compute and fill the substars of (G, α) by the parallel algorithm given in Section 4. On the resulting graph, run the parallel algorithm of [12].

Now, we want to show that the total time requirement for this process is $O(\log^3 n)$ using $O(nm)$ processors on a CREW PRAM. There is one point we need to handle carefully. The input graph G has n vertices and m edges, but $G_\mathscr{S}$ has more edges. Thus, if we simply compute $G_\mathscr{S}$ and pass it on to the algorithm of [12], then the $O(nm)$ bound on the number of processors does not necessarily hold. However, if instead of using the algorithm of [12] as a black box, we do the necessary calculations on G and $\mathscr{S}$ so that we can go directly into the appropriate intermediate step of [12], we can keep the bounds related to the parameters of G instead of the parameters of $G_\mathscr{S}$.

Theorem 5.2 *Given (G, α), a minimal triangulation of G that is a subgraph of G_α^+ can be computed in parallel $O(\log^3 n)$ time with $O(nm)$ processors on a CREW PRAM.*

Proof. Let the substars $\mathscr{S}$ of (G, α) be computed by the parallel algorithm described in Section 4 in time $O(\log^2 n)$ using $O(nm)$ processors on a CREW PRAM.

We first define a partial ordering on vertex subsets of G given $\mathscr{S}$. Let S be a substar in $\mathscr{S}$. We say that a vertex $x < S$ if x is in a full component C of S in G

such that C is not the unique largest (in the number of vertices) full component of S in G. We define the *closest substar of x* to be a substar S with $x < S$ such that the connected component of $G(V - S)$ containing x is the smallest in size compared to the full components containing x of all other substars $> x$. We claim that given a vertex x there is a unique substar satisfying this condition, provided that there is at least one substar $> x$. Let S_1 be a closest substar of x; let C be the full component of S_1 containing x; then any other substar is either a subset of $C \cup S_1$ or does not intersect C. Let S_2 be another closest substar of x; first we assume that S_2 does not intersect C: if S_1 is a subset of S_2 then C is a component of $G(V - S_2)$, but not a full component of S_2. If S_1 is not a subset of S_2, then $C \cup (S_1 - S_2)$ is a subset of a component of $G(V - S_2)$. This component is larger than C, contradicting the assumption that S_2 is a closest substar. Finally we have to consider the case that S_2 is a subset of $C \cup S_1$: then S_2 cannot be a subset of S_1 because C and the vertices in $S_1 - S_2$ would belong to the full component of S_2 containing x, contradicting the assumption that S_2 is a closest substar. For symmetry reasons, S_1 cannot be a subset of S_2. Let C_2 be any full component of S_2 not containing x; since S_2 is not a subset of S_1, any vertex of C_2 can be joined by a path in G with x, avoiding S_1. Therefore C_2 is a subset of C. Note that there is another full component D of S_1 that is at least as large as C. For the same reasons that C_2 is a subset of C, D is a subset of the full component of S_2 containing x. Since C_2 is any full component of S_2 not containing x, there is no full component of S_2 not containing x that is at least as large as D, contradicting the assumption that S_2 is a substar with $x < S_2$, and therefore contradicting the assumption that S_2 is a closest separator.

Now we define a partition of V into vertex subsets called *cells* which we will arrange in an order. A cell contains all the vertices having the same closest substar S and belonging to the same connected component of $G(V - S)$. The closest substar of a cell is the common closest substar of its elements. A special cell (without closest substar) contains all vertices having no closest substar. We now order the cells as follows: a cell is smaller than any cell intersecting its closest substar. We must show that this defines a partial order among the cells, i.e. that it has no cycles. Suppose $x < S_1$, $y \in S_1$, and $y < S_2$. Let C_1 be the connected component of $G(V - S_1)$ containing x and C_2 be the connected component of $G(V - S_2)$ containing y. We claim that C_1 is a proper subset of C_2. To prove the claim, it is sufficient to show that S_2 does not intersect C_1 (in that case, C_1 and y are in one connected component of $G(V - S_2)$). Assume now that S_2 intersects C_1. Let D_1 be a largest connected component of $G(V - S_1)$ (D_1 is different from C_1) and D_2 be a largest connected component of $G(V - S_2)$ (D_2 is different from C_2). Since the minimal separators S_1 and S_2 do not cross, C_1 is the only connected component of $G(V - S_1)$ that intersects S_2. It follows that D_1 and y are in one connected component of $G(V - S_2)$, so $|D_1| < |D_2|$. In the same way, C_2 is the only connected component of $G(V - S_2)$ that intersects S_1, D_2 and $S_2 \cap C_1$ are in one connected component of $G(V - S_1)$, so $|D_2| < |D_1|$. This is a contradiction.

Now any extension of this partial ordering among the cells to a total ordering is an approximate minimal elimination ordering of $G_{\mathscr{S}}$, in the sense that there exists a minimal elimination ordering of $G_{\mathscr{S}}$ in which the vertices of the smallest cell appear first, the vertices of the next smallest cell appear thereafter, etc. Let K be a cell. The full component of the closest substar containing K is called the full component of K. To determine the cells we sort the vertices

lexicographically by their closest separators and by the full components of the separators containing them. Then one can easily observe that vertices of the same cell appear consecutively. This can be done in $O(log^2(n))$ time using $O(n^2)$ processors because the comparison of two separators and two components can be done in $O(log(n))$ time with $O(n)$ processors and sorting can be done in $O(log(n))$ time with $O(n)$ processors if we assume that comparison can be done in constant time [9]. To get a total ordering of the cells that is an extension of the partial order on the cells as mentioned before, we sort the cells by the cardinalities of their full components. This can be done in logarithmic time using $O(n)$ processors.

This approximate minimal elimination ordering can be extended to a minimal elimination ordering by deciding the local order of the vertices in each of the cells. This is done in [12] as follows: for any cell K we determine a minimal elimination ordering of $G_{\mathscr{S}}$. Secondly, for each cell K, we transform the minimal elimination ordering of K into a minimal elimination ordering with the same fill such that the second ordering concatenated with any ordering of its closest substar S is a minimal elimination ordering of $G_{\mathscr{S}}$ restricted to $K \cup S$. By [12], this can be done in $O(log^3(n))$ time using $O(nm)$ processors for all cells simultaneously.

The substar edges do not play a role regarding the limits of time and number of processors, and we can run the algorithm of [12] within the desired limits for time and number of processors. We conclude that the total time requirement of this parallel algorithm solving the sandwiched minimal triangulation problem is $O(\log^3 n)$ using $O(nm)$ processors. ∎

5.2 A new minimal triangulation algorithm: MEG

In this subsection we introduce a new algorithm that completes a given partial minimal triangulation resulting from the process described in Section 3 to a minimal triangulation directly without using another existing minimal triangulation algorithm. We will repeat the process of saturating in the partially triangulated graph G' obtained so far the substars of (G', β) for some ordering β, until a chordal graph is obtained. However, at the beginning of each iteration, we will remove all LB-simplicial vertices, according to the following result:

Lemma 5.3 *Let $G = (V, E)$ be a graph, X the set of LB-simplicial vertices of G, and $G' = G(V - X) = (V', E')$. For any minimal triangulation $H' = (V', E' + F')$ of G', the graph $H = (V, E + F')$ is a a minimal triangulation of G.*

To prove Lemma 5.3 we use the following result which generalizes Property 2.12 a) and b).

Lemma 5.4 *Let G be a graph, v be a LB-simplicial vertex of G, G' be an induced subgraph of G and $\mathscr{S}$ be a set of minimal separators of G'. Then v has the same neighborhood and substars in G and in $G_{\mathscr{S}}$, and it is LB-simplicial in $G_{\mathscr{S}}$.*

Proof. Suppose for contradiction that v has not the same neighborhood or not the same substars in G and in $G_{\mathscr{S}}$. Then there is some $S \in \mathscr{S}$, some

vertices $y, z \in S$ and some component C of $G(V - N_G[v])$ such that $y \in C$ and $z \notin N_G[C]$. Let C_1 and C_2 be distinct full components of S in G' and let P_1 and P_2 be paths in G' between y and z whose internal vertices are in C_1 and C_2 respectively. As $y \in C$ and $z \notin N_G[C]$, there is an internal vertex v_1 of P_1 (resp. v_2 of P_2) in $N_G(C)$. As v is LB-simplicial in G, $N_G(C)$ is a clique of G, so v_1 and v_2 are equal or adjacent in G', which is impossible since $v_1 \in C_1$ and $v_2 \in C_2$. So v has the same neighborhood and substars in G and in $G_{\mathscr{S}}$, and therefore it is LB-simplicial in $G_{\mathscr{S}}$ since it is LB-simplicial in G. ∎

Proof. (of Lemma 5.3) H is chordal because for any cycle C in H of length $\geq$ 4, either C is in H' and then C has at least one chord, or C contains a vertex x of X and then C has at least one chord by Lemma 3.17 since x is LB-simplicial in H by Lemma 5.4 ($H = G_{\mathscr{S}}$ for some set $\mathscr{S}$ of minimal separators of G' by Theorem 2.4). So H is a triangulation of G. It is a minimal one because for any chordal graph $H_1 = (V, E + F_1')$ with $F_1' \subseteq F'$, the graph $H_1' = (V', E' + F_1')$ is chordal too, so that $F_1' = F'$. ∎

Thus the LB-simplicial vertices can only cause EG to add extraneous edges, as well as unnecessarily increasing some vertex degrees if MD orderings are used, which justifies our systematically eliminating them from the graph at each step. Note that saturating the substars of (G', β) tends to create LB-simplicial vertices (at least $\beta(1)$ is LB-simplicial in $G'_{\mathscr{S}}$ as proved below), so removing these can make a significant difference regarding the quality of the fill obtained when MD orderings are used. We now present the new algorithm.

Algorithm Minimal Elimination Game (MEG)
Input: A graph $G = (V, E)$ and an ordering α on G.
Output: A sandwiched minimal triangulation H of G.
Compute the set of substars $\mathscr{S}$ of (G, α);
$G' \leftarrow G_{\mathscr{S}}$; $H \leftarrow G_{\mathscr{S}}$;
while G' is not chordal **do**
    Remove all LB-simplicial vertices from G';
    Choose an arbitrary ordering β and compute the set of substars $\mathscr{S}$ of (G', β);
    $G' \leftarrow G'_{\mathscr{S}}$; $H \leftarrow H_{\mathscr{S}}$;

Theorem 5.5 *Given a graph G and an ordering α on its vertices, MEG computes a minimal triangulation of G which is a subgraph of G_α^+.*

Proof. MEG terminates, because at least one vertex is removed at each step since $\beta(1)$ is LB-simplicial in $G'_{\mathscr{S}}$, where $\mathscr{S}$ is the set of substars of (G', β): $\beta(1)$ is LB-simplicial in $G'_{\mathscr{S}_1}$, where $\mathscr{S}_1$ is the set of substars of (G', β) defined at step 1, and $\beta(1)$ is still LB-simplicial in $G'_{\mathscr{S}}$ by Lemma 5.4 with G' and G equal to $G'_{\mathscr{S}_1}$ and Theorem 3.3 since $\mathscr{S}$ is also the set of substars of $(G'_{\mathscr{S}_1}, \beta)$. Let us now prove MEG correctness. Let H be the output graph, let $\mathscr{S}_0$ be the set of substars computed at the beginning of the algorithm and $\mathscr{S}'$ be the union of those computed in the **while**-loop. Thus $H = (G_{\mathscr{S}_0})_{\mathscr{S}'}$, $G_{\mathscr{S}_0}$ being the input graph of the **while**-loop. By Theorem 5.1 it is sufficient to show that H is a minimal triangulation of $G_{\mathscr{S}_0}$, or more generally that for any input graph G' of the **while**-loop, the graph $G'_{\mathscr{S}'}$, where $\mathscr{S}'$ is the union of the sets of substars computed in the **while**-loop, is a minimal triangulation of G'. Let us prove this property by induction on the number p of iterations of the

while-loop before G' becomes chordal. It trivially holds for $p = 0$, as in that case G' is chordal and $\mathscr{S}'$ is the empty set. We suppose that it holds when the number of iterations of the **while**-loop before G' gets chordal is p. Let us show that it holds when this number is $p+1$. Let G'_1 be the graph obtained from G' by removing all its LB-simplicial vertices, let $\mathscr{S}'_1$ be the set of substars computed at the first iteration of the **while**-loop and $\mathscr{S}''$ be the union of those computed at the following iterations, and let G'' be the graph obtained at the end of the first iteration. Thus $G'' = (G'_1)_{\mathscr{S}'_1}$ and $\mathscr{S}' = \mathscr{S}'_1 \cup \mathscr{S}''$. By the induction hypothesis, $G''_{\mathscr{S}''}$ is a minimal triangulation of G'', so by Theorem 3.4 and Lemma 2.7, it is also a minimal triangulation of G'_1. $G''_{\mathscr{S}''} = ((G'_1)_{\mathscr{S}'_1})_{\mathscr{S}''} = (G'_1)_{\mathscr{S}'_1 \cup \mathscr{S}''} = (G'_1)_{\mathscr{S}'}$. Thus the graphs $G'_{\mathscr{S}'}$ and $G''_{\mathscr{S}''}$ are obtained from G' and G'_1 respectively by adding the same set F' of edges, so by Lemma 5.3, $G'_{\mathscr{S}'}$ is a minimal triangulation of G', which completes the proof by induction and therefore the proof of MEG correctness. ∎

Thus MEG solves the sandwiched minimal triangulation problem directly, given (G, α).

We will remark that the orderings β used at the successive steps of MEG are not necessarily sub-orderings of α. However, when only sub-orderings of α are used, the following result is obtained.

Property 5.6 *If at each step, the ordering β is the restriction of α to the vertices of G' then MEG yields the minimal triangulation G_α^{LB} computed by LB-Triang.*

We first prove the following Lemma.

Lemma 5.7 *If $G \subseteq G' \subseteq G_\alpha^{LB}$ then $(G')_\alpha^{LB} = G_\alpha^{LB}$.*

Proof. For any k between 1 and $n+1$, let $H_k = G_\alpha^{LB,k}$ and $H'_k = (G')_\alpha^{LB,k}$. Let us show by induction on k that for any k from 1 to $n+1$, $H_k \subseteq H'_k \subseteq G_\alpha^{LB}$. It holds for $k = 1$ since $H_1 = G$ and $H'_1 = G'$. We suppose that $H_k \subseteq H'_k \subseteq G_\alpha^{LB}$. By Property 2.12 a) v_k has the same neighborhood and substars in H_k and in G_α^{LB}, so v_k also has these same neighborhood and substars in H'_k. As H_{k+1} and H'_{k+1} are obtained from H_k and H'_k respectively by saturating these substars which are cliques of G_α^{LB}, $H_{k+1} \subseteq H'_{k+1} \subseteq G_\alpha^{LB}$ which completes the proof by induction. Hence $H_{n+1} \subseteq H'_{n+1} \subseteq G_\alpha^{LB}$, i.e. $G_\alpha^{LB} \subseteq (G')_\alpha^{LB} \subseteq G_\alpha^{LB}$, so $(G')_\alpha^{LB} = G_\alpha^{LB}$. ∎

Proof. (of Property 5.6) Let us show that the following property (P) holds at every step (i.e. at the beginning of every iteration of the **while**-loop) of MEG, where V' is the vertex set of G':
(P) $G' = H(V')$, every vertex of $V - V'$ is LB-simplicial in H and $G \subseteq H \subseteq G_\alpha^{LB}$.
(P) holds at the first step since $G' = H = G_\mathscr{S}$ and by Theorem 3.18 $G_\mathscr{S} \subseteq G_\alpha^{LB}$. We suppose that (P) holds at some step of MEG. Let us show that it still holds at the next step. Let G'_1 be the graph obtained from G' by removing its LB-simplicial vertices, $\mathscr{S}$ be the set of substars of (G'_1, α') where α' is the restriction of α to the vertices of G'_1, $G'' = (G'_1)_\mathscr{S}$, $H' = H_\mathscr{S}$ and V'' be the vertex set of G''. Clearly $G'' = H'(V'')$ and $G \subseteq H'$. Let v be a vertex of $V - V''$, and let us show that v is LB-simplicial in H'. By Lemma 5.4, it is sufficient to show that v is LB-simplicial in H. We suppose by contradiction that v is not LB-simplicial in

H. By (P) $v \notin V - V'$. It follows that $v \in V' - V''$ and therefore is LB-simplicial in G'. By Lemma 3.17, as v is not LB-simplicial in H and every vertex of $V - V'$ is LB-simplicial in H, v belongs to a chordless cycle in $H(V')$, i.e. G', of length at least 4, and therefore v is not LB-simplicial in G', a contradiction. It remains to show that $H' \subseteq G_\alpha^{LB}$. By Theorem 3.18 $(G'_1)_{\mathscr{S}} \subseteq (G'_1)_{\alpha'}^{LB}$. $G'_1 = H(V'')$, every vertex of $V - V''$ is LB-simplicial in H and by Property 2.12 c), first removing the LB-simplicial vertices of $V - V''$ from H does not modify the fill computed by LB-Triang on (H, α). Thus LB-Triang computes the same fill on (G'_1, α') and on (H, α), so we also have $H_{\mathscr{S}} \subseteq H_\alpha^{LB}$. Now, as by (P) $G \subseteq H \subseteq G_\alpha^{LB}$, by Lemma 5.7 $H_\alpha^{LB} = G_\alpha^{LB}$, so $H' = H_{\mathscr{S}} \subseteq H_\alpha^{LB} = G_\alpha^{LB}$, which completes the proof that (P) still holds at the next step.

It follows that the graph H computed by MEG is chordal and satisfies $G \subseteq H \subseteq G_\alpha^{LB}$, and therefore $H = G_\alpha^{LB}$ since G_α^{LB} is a minimal triangulation of G. ∎

It follows from Property 5.6 that to do "better" than LB-Triang in the sense of producing less fill, MEG has to take advantage of the possibility given by the algorithm of choosing an appropriate ordering β at each iteration of the **while**-loop instead of the given ordering α. For instance, ordering β can be chosen as an MD ordering of the transitory graph (we call MMD this variant of MEG using MD heuristic). However, this potential gain on the quality of the fill has a cost in terms of computational time, at least theoretically, as shown by the rapid following investigation of MEG time complexity. By Corollary 4.5 computing a partial minimal triangulation of the transitory graph at each iteration of the **while**-loop requires $O(nm')$ time, where m' is the number of edges of the resulting minimal triangulation of G. If a straightforward algorithm in $O(nm')$ is used to determine whether a given vertex is LB-simplicial in the transitory graph, we obtain $O(n^2m')$ for each iteration of the **while**-loop and $O(n^3m')$ time complexity for MEG, whereas LB-Triang can be implemented in $O(nm)$ time [4]. As in practice (with a MD approach) the number of iterations of the **while**-loop is rarely more than 2 (see Section 6), one may wonder whether there is a better bound than $O(n)$ for the number of iterations in an execution of MEG. The following example shows that the answer is negative. Consider for any integer $k \geq 1$ the graph G_k with $n = 3k + 1$ vertices consisting of a chain of k chordless 4-cycles in the form $(3i+1, 3i+2, 3i+4, 3i+3)$ for i from 0 to $k-1$ (two consecutive cycles have a common vertex since $3i + 4 = 3(i+1) + 1$) such that vertex $3i + 2$ of each cycle is adjacent to every vertex of the subsequent cycles (graph G_3 is shown in Figure 4). With $\alpha = (1, 2, ..., n)$ and $\beta = (3i+1, 3i+2, ..., n)$
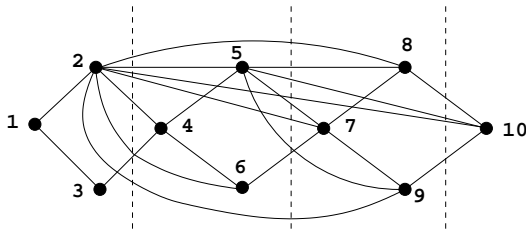

Figure 4: An example of execution of MEG with $O(n)$ iterations.

at iteration i for i from 0 to $k - 1$, the number of iterations is $k = (n-1)/3$

(the 3 iterations on G_3 are represented by dashed lines in Figure 4). Note that with MMD there is only one iteration, so the problem of the best bound of the number of iterations remains open in the case of MMD. Thus even if the time complexity of removing LB-simplicial vertices is reduced, MEG time complexity remains in at least $O(n^2 m')$, though it is much better in practice when appropriate orderings β are chosen. Beyond the fact that MEG may produce less fill than LB-Triang on some graphs (we leave it to the reader to test it on the kind of graphs used in his or her research domain), MEG presents a new approach of EG and the sandwich problem which may lead to further properties and algorithms. In particular, it can be combined with some heuristics that cannot be used in the framework of the parallel algorithm presented in Section 5.1 because they are inherently sequential, as is the case for MD heuristic leading to algorithm MMD.

6 Applications to MD and experimental results

As mentioned earlier, though EG does not necessarily compute a minimal triangulation, MD is observed in practice to often produce orderings that are minimal or close to minimal, in addition to low fill. We will now give an explanation of this good behavior of MD through Lemma 6.1.

Lemma 6.1 *Let v_k be a vertex of minimum degree in G_α^k such that the union of all substars defined at step k is equal to a substar S defined at step k. Then only substar fill edges are added at step k.*

Proof. Assume by contradiction that there is an extraneous edge yz added at step k. Then y or z, say y, is not in S. $N_{G_\alpha^k}(y) \subseteq N_{G_\alpha^k}[v_k] - \{y, z\}$, so that $|N_{G_\alpha^k}(y)| < |N_{G_\alpha^k}(v_k)|$, which contradicts the fact that v_k is a vertex of minimum degree in G_α^k. ∎

This result shows in particular that when the section graph is connected at some step of MD, then fill edges are added at that step only within the single substar defined. As is clear from the proof of this lemma, this is not the case in general for EG. Moreover, in most practical applications [23], the input graph is sparse; although no statistical result has been established on this, intuitively, a vertex x of minimum degree quite often defines only one substar, which usually corresponds to a connected section graph. MD run on sparse graphs thus stands a significantly higher chance of generating minimal triangulations than an arbitrary EG.

It should be noted that graphs can be constructed such that no execution of MD can produce a minimal triangulation. Such an example is a graph consisting of two large cliques, connected by a single path of length ≥ 2. The graph is chordal, but vertices on the path will be chosen by MD at first steps, introducing unnecessary fill. MEG run with an MD approach (introduced above as MMD) will not encounter any problem with that kind of graph, since the only minimal triangulation of a chordal graph is the graph itself.

With practical tests, we have compared MMD against MD with respect to the number of edges in the resulting triangulation. We have done a simple and straightforward implementation of MMD in Matlab, and we have run the tests

25

both on randomly generated graphs of varying density, and on graphs from Matrix Market [23]. On each graph G, we first generated an MD ordering α. Then we compared the number of fill edges in G_α^+ to the number of fill edges produced by MMD. As expected by Theorem 5.5, the number of fill edges resulting from MMD was always less than or equal to the number of fill edges resulting from MD. An interesting point is that on most graphs, MMD required only two iterations of the **while**-loop. The reduction in the number of fill edges achieved by MMD was not very large, because of MD's remarkably good performances. However, this improved algorithm may both give significant results on very large graphs and help researchers gain a better evaluation of how close MD gets to an optimal solution. Finally, we would like to mention that existing MD codes in use are very fast although the theoretical running time of these implementations is not good [17]. In addition, other iterative procedures for computing minimal triangulations have been implemented to run fast in practice[27]. Thus we believe that, with some effort, MMD can be implemented to run efficiently in practice.

It would be interesting to compare the fill produced by MMD with the fill produced by the variant of LB-Triang (called Dynamic LB-Triang in [4]), in which ordering α is defined dynamically by choosing at each step an unprocessed vertex with minimum degree in the transitory graph. Once more, we leave it to the reader to test this on the graphs used in his or her research domain, as the results may depend on the properties of these graphs (sparse or dense for example), and to imagine combinations or variants of these algorithms to solve the sandwich or other problems.

7   Conclusion

We have conducted a theoretical investigation of Elimination Game, and covered new aspects with regard to minimal triangulation.

We showed that Elimination Game is remarkably robust. Though unnecessary edges may be added at some steps, the chances remain intact at further steps to add only useful edges. We use this property to define a partial minimal triangulation of the input graph, any minimal triangulation of which solves the sandwich problem. Based on this, we present several algorithms which compute a minimal triangulation solving the sandwich problem, including an efficient parallel approach.

All these processes improve Minimum Degree heuristics. Furthermore our results explain at least partially the good behavior of this heuristic.

Acknowledgments

References

[1] P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17:886–905, 1996.

[2] A. Berry. Désarticulation d'un graphe. *PhD Dissertation*, LIRMM, Montpellier, December 1998.

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

[4] A. Berry, J.-P. Bordat, P. Heggernes, G. Simonet, and Y. Villanger. A wide-range algorithm for minimal triangulation from an arbitrary ordering. *J. Algorithms*, 57(2), 2005.

[5] A. Berry, P. Heggernes, and G. Simonet. The Minimum Degree Heuristic and the Minimal Triangulation Process. *Proceedings WG 2003 - 29th Workshop on Graph Theoretic Concepts in Computer Science, June 2003* Lecture Notes in Computer Science 2880, pages 58 - 70.

[6] J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comput. Sci.*, 250:124–141, 2001.

[7] J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In J. A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, pages 1–30. Springer Verlag, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

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

[9] R. Cole. Parallel merge sort. *SIAM J. Computing*, 17:770–785, 1988.

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

[11] E. Dahlhaus. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *J. Algorithms*, 36: 205–240, 2000.

[12] 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.

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

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

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

[16] J. A. George and J. W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[17] P. Heggernes, S. Eisenstat, G. Kumfert, and A. Pothen. The computational complexity of the Minimum Degree algorithm. *Proceedings of 14th Norwegian Computer Science Conference, NIK 2001, University of Tromsø, Norway.* Also available as ICASE Report 2001-42, NASA/CR-2001-211421, NASA Langley Research Center, USA.

[18] 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.

[19] 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.

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

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

[22] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3:255–269, 1957.

[23] Matrix Market *Web site:* http://math.nist.gov/MatrixMarket/

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

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

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

[27] B. Peyton. Minimal orderings revisited. *SIAM J. Matrix Anal. Appl.*, 23:271–294, 2001.

[28] D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970.

[29] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, 1972.

[30] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. Algorithms.* 3: 57–67, 1982.

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

[32] 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.

[33] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization *In Proceedings of the IEEE*, 55:1801–1809, 1967.

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

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