Fast minimal triangulation algorithm using minimum degree criterion

Mauro Mezzini

April 2011

Abstract

We propose an algorithm for minimal triangulation which, using simple and efficient strategy, subdivides the input graph in different, almost non-overlapping, subgraphs. Using the technique of matrix multiplication for saturating the minimal separators, we show that the partition of the graph can be computed in time $O(n^\alpha)$ where n^α is the time required by the binary matrix multiplication. After saturating the minimal separators the same procedure is recursively applied on each subgraphs. We also present a variant of the algorithm in which the minimum degree criterion is used. In this way we obtain an algorithm that uses minimum degree criterion and at the same time produces a minimal triangulation, thus shedding new light on the effectiveness of the minimum degree heuristics.

Key words: Minimal triangulations; Chordal graphs; Minimum Degree;

1. Introduction

A graph is *chordal* if every cycle of length greater than three has a chord which is an edge of the graph joining two non adjacent vertices of the cycle. A *triangulation* of a graph G is a chordal graph obtained from G by adding to it a set of edges. Historically, the problem of finding a triangulation of a graph is related to the problem of computing the inverse of large sparse symmetric matrices [21, 12, 7]. Other important fields in which the triangulation problem applies is database management [22, 1], artificial intelligence and statistics [17, 9, 19], to cite few. In these applications one wants to find a triangulation of the input graph whose edge set is the minimum possible. But such minimum triangulation is difficult to obtain since this has been shown to be an NP-Hard problem [23]. However the problem of finding a *minimal* triangulation, that is, finding an inclusion minimal set of edges which added to the original graph makes it chordal, can be solved in polynomial time. Therefore much of the effort in the literature has been devoted in developing algorithms for finding minimal triangulations, as efficient as possible [6, 12, 21, 14, 2, 3, 15, 19, 2, 4, 18].

Another approach, the one used in the Minimum Degree Algorithm (MDA) [11] tries to obtain a triangulation of a graph, not necessarily minimal, by employing the minimum degree heuristic. Such approach has been successful since the triangulations produced by the MDA are often minimal and, while the worst case complexity bounds of the MDA are much higher than other minimal triangulation algorithms, such complexity bounds are tight only for very dense graphs and are not often observed for problems that are solved in practice [5].

Here we present an algorithm that uses the minimum degree criterion and produces at the same time minimal triangulations. The idea is to divide the graph in different, almost non-overlapping, subgraphs and compute a minimal triangulation of each subgraph. We prove that if we add to the original graph the edges needed to obtain a minimal triangulation of each subgraph, then we produce a minimal triangulation of the original graph.

Thanks to this we will consider, among others, the possibility of recursively apply the same procedure on each subgraph, since each recursive call will spent time only on a non-overlapping part of the original graph.

We will show that all the tasks needed to obtain the partition of the graph, with the exception of the task of saturating the minimal separators, can be easily carried out and require time linear in the dimension

of the triangulated graph. As for the task of computing the set of edges needed to saturate a set of minimal separators, we may use the technique of matrix multiplication (such as that used in [14]) which has time complexity of $O(n^\alpha)$. Thus we obtain a partition of the graph with simple and very fast procedures.

We will see that we can partition the graph using the criterion of the minimum degree. While this approach does not always assure a balanced partition of the graph, i.e., does not assure that the algorithm terminates in a logarithmic number of steps, it seems the most natural and simplest choice. This is interesting also from a theoretical point of view. The fact that each subgraph of the partition can be triangulated separately of the other subgraphs, sheds new light on why the local minimum degree heuristic is so effective (and fast) in producing small or minimal triangulations on sparse graphs.

We made an implementation of our algorithm and realized various experiments. We tested our algorithm both on (pseudo) randomly generated graphs and on much of the symmetric squared sparse matrices from the Harwell-Boeing collection [8]. We compared the execution time of our algorithm with the execution time of the well-known MCS-M [2] algorithm. Although in building the implementation, we put our attention much on correctness and readability of the code and we did not implement the matrix multiplication technique for computing the minimal separators, the tests made, shown that our algorithm perform dramatically better than MCS-M. Furthermore the experiments made on the matrices of the Harwell-Boeing collection confirmed that, using the minimum degree criterion in the first iteration of the algorithm, the graphs broke in many (sometimes hundreds and even thousand), smaller subgraphs.

The work is organized as follows. In Section 2 we give some definitions and preliminary results. In Section 3 we describe the procedure for partitioning the graph in what we call quasi-split graphs. In Section 4 we give two algorithms to compute a minimal triangulation of two special cases of quasi-split graph, interesting on their own and useful in the subsequent parts. In Section 5 we detail the steps of a recursive algorithm. In Section 6 we report the results of our experimentation. Finally, in Section 7, we give some closing remarks and list possible future research issues.

2. Definitions and preliminaries

We consider simple undirected loopless graphs $G = (V(G), E(G))$, where $V(G)$ is a set of vertices and $E(G)$ is a set of pairs of distinct vertices. We denote, unless explicitly specified, $|V(G)|$ by n and $|E(G)|$ by m. A pair of distinct vertices uv is called an *edge*; u and v are called the *endpoints* of uv and are *adjacent* if $uv \in E(G)$. The *neighborhood* of a vertex v is the set of its adjacent vertices, denoted by $N_G(v)$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. Given a set S of vertices the neighborhood of S is $N_G(S) = \cup_{v \in S} N_G(v) \setminus S$. The *closed neighborhood* of a vertex set S is $N_G[S] = N_G(S) \cup S$. The *common neighborhood* of two vertices u and v is $CN_G(uv) = N_G(u) \cap N_G(v)$. When it is clear from the context we omit the subscript and write $d(v)$, $N(v)$ and $CN(uv)$. A *path of length* k is a sequence of distinct vertices $(v_0, v_1, \ldots, v_k)$ such that $v_i v_{i+1} \in E(G)$ for $0 \leq i < k$. A *cycle of length* k a sequence of vertices $(v_0, v_1, \ldots, v_k)$ such that $v_i v_{i+1} \in E(G)$ for $0 \leq i < k$ and only the first and the last vertex do coincide.

An *independent set* is a set of pairwise non adjacent vertices. A set of pairwise adjacent vertices is called a *clique*. When we *saturate* a set S of vertices we add an edge between every pair of non adjacent vertices of S.

The subgraph of G *induced* by a subset S of $V(G)$, denoted as $G[S]$ or simply by S when there is no ambiguity, is a graph with vertex set S and edge set $E(G[S]) = \{uv \in E(G) : u \in S \wedge v \in S\}$. Given a set S of vertices of a graph G, by $G - S$ we denote the subgraph of G induced by $V(G) \setminus S$. If S is a singleton, i.e. $S = \{v\}$, then we write $G - v$. Given a set F of edges, by $G + F$ we denote the graph with vertex set $V(G)$ and edge set $E(G) \cup F$.

Two vertices u and v are *connected* if there exists in G a path joining them, and disconnected otherwise. A maximal set C of pairwise connected vertices is a *connected component* or simply a component of G.

A set S is a *uv-separator*, for $u, v \notin S$, if u and v are connected in G, but disconnected in $G - S$, and is a *minimal* uv-separator if no proper subset S' of S is a uv-separator. We say that S is a *minimal separator* when there exist two vertices u and v such that S is a minimal uv-separator. Given a vertex subset S, a component C of $G - S$ is *full* if $N_G(C) = S$. A vertex set S is a minimal separator if it has at least two full components [20].

A chord of a path (or a cycle) is an edge of $E(G)$ between a pair of non consecutive vertices of the path (or cycle). A graph is *chordal* or *triangulated* if every cycle of length greater than three has a chord. A *triangulation* of a graph G is a chordal graph $G + F$ obtained from G by adding a set F of edges. A triangulation $G + F$ is *minimal* if for no proper subset F' of F we have that $G + F'$ is chordal. The set F is called (minimal) *fill-in*.

Lemma 1 ([21]). *Let G be a graph ad F be a minimal fill-in. Then every edge of F is a unique chord in a four cycle.*

Lemma 2 ([19, 15]). *A triangulation $G + F$ of a graph G is minimal if and only if for every $uv \in F$, $CN_{G+F}(uv)$ is not empty and is not a clique of $G + F$.*

A vertex is *simplicial* if its neighborhood is a clique. A vertex v is *LB-simplicial* if every minimal separator included in $N_G(v)$ is a clique. Note that a vertex is LB-simplicial if and only if $N_G[v]$ is an m-convex set, that is, if it contains the vertices of every chordless path between vertices in $N_G[v]$ [10].

Lemma 3 ([10, 16]). *A graph is chordal if and only if every vertex is $LB - simplicial$.*

Consider the following procedure called LB-Triang [3]

Algorithm LB-Triang
input : A graph G
output : A minimal fill-in F
begin
  let $v_1, \ldots, v_n$ be an (arbitrary) ordering of the vertices of G;
  $F \leftarrow \emptyset$;
  for $i = 1 \ldots, n$ **do**
  begin
    let F_i be the set of edges
      needed to make v_i LB-simplicial;
    $F \leftarrow F \cup F_i$;
    delete v_i;
  end
end

Lemma 4 ([3]). *The algorithm* LB-Triang *produces a minimal fill-in F.*

The following is a consequence of Lemma 2 and of the fact that a minimal separator has at least two full components

Lemma 5. *Let G be a chordal graph and S a minimal separator of G. Then for every two distinct vertices u and v of S we have that $CN_G(uv)$ is not empty and is not a clique.*

A graph G is *bipartite* if there exists a partition of its vertex set in two non empty sets (P, Q) such that both P and Q are independent sets. A bipartite graph with bipartition (P, Q) is a *chain* graph [23] if there exists an ordering $v_1, \ldots, v_{|Q|}$ of the vertices of Q, such that $N(v_1) \subseteq N(v_2) \subseteq \cdots \subseteq N(v_{|Q|})$. Two edges xy and uv of a bipartite graph are said to be *independent* [23] if the subgraph induced by $\{x, y, u, v\}$ consists of exactly these two edges.

Lemma 6 ([23]). *A graph is a chain graph if and only if it does not contain two independent edges.*

A graph G is a *quasi-split graph* if its vertex set can be partitioned in two sets P and Q where P is a clique, $G[Q]$ is connected and each vertex v of Q (resp. of P) has at least one neighbor in P (resp. in Q).

Given a bipartite graph B (resp. quasi-split graph G) we refer to a partition (P, Q) of the vertices of B (resp. G) as a *bipartition* of B (resp. G) and denote it by (P, Q). In particular if G is a quasi-split graph then we denote by P the set of vertices of the bipartition which induces a clique, unless explicitly specified.

Lemma 7. *Let G be a chordal graph and K a maximal clique of G. Each component of $G - K$ contains at least a simplicial vertex of G.*

Proof: Suppose, by contradiction, that there exists a component C of $G - K$ such that every vertex of C is not a simplicial vertex of G. It is known (see Theorem 3.2 of [10]) that in a chordal graph, every non simplicial vertex lies on a chordless path between two simplicial vertices. Let $v \in C$. Since v is not simplicial in G then v lies on a chordless path between two simplicial vertices of $V(G) \setminus C$. Since $N_G(C) \subseteq K$ then v lies on a chordless path between two vertices of K. But this is impossible since K is a clique and it contains all the vertices of every chordless path between vertices of K (contradiction). □

3. The partitioning of the graph

Here we describe the procedures for the partitioning of the graph.

We start with a BSF from a vertex denoted as ν of G and we compute for each vertex its distance from ν. Let us denote by δ the maximum distance from ν to any other vertex of G. For $l = 0, \ldots, \delta$ denote by S_l the set of vertices at distance l from ν. We will refer to a vertex $v \in S_l$ as a vertex at *distance* l. Clearly if $0 < l < \delta$ then $G - S_l$ has at least two connected components one of which contains ν and all the vertices of G with distance $l - 1$ or less from ν; the others connected components contain vertices with distance $l + 1$ or more from ν. We will refer to a connected component of $G - S_l$ not containing ν, as a connected *component of level* l and let us denote it as C_i^l, $l = 1, \ldots, \delta - 1$ and $i = 1, \ldots, k_l$. Of course $N(C_i^l) \subseteq S_l$ and C_i^0 is a connected component of $G - \nu$. Note that if $\{\nu\}$ is not a separator then there exists only one component C_1^0 at level 0. As the next step of the algorithm we saturate $N(C_i^l)$ for all $l = 1, \ldots, \delta - 1$ and for all $i = 1, \ldots, k_l$. Denote by F the set of edges added in this step.

Definition 8. *For each $l = 0, \ldots, \delta - 1$ and for each $i = 1, \ldots, k_l$, if C_i^l is a component at level l let $P_i^l = N_G(C_i^l)$ and let $Q_i^l = C_i^l \cap S_{l+1}$ and denote by $H_{C_i^l}$ the subgraph of $G + F$ induced by $P_i^l \cup Q_i^l$.*

We have that each graph $H_{C_i^l}$ is a quasi-split graph.

Example Consider the graph of Figure 1(a). In Figure 1(b) we saturate $N(C_i^l)$ for each component C_i^l and for all $l = 1, 2$. In Figure 2 are reported all the subgraphs $H_{C_i^l}$ for all $l = 0, 1, 2$.

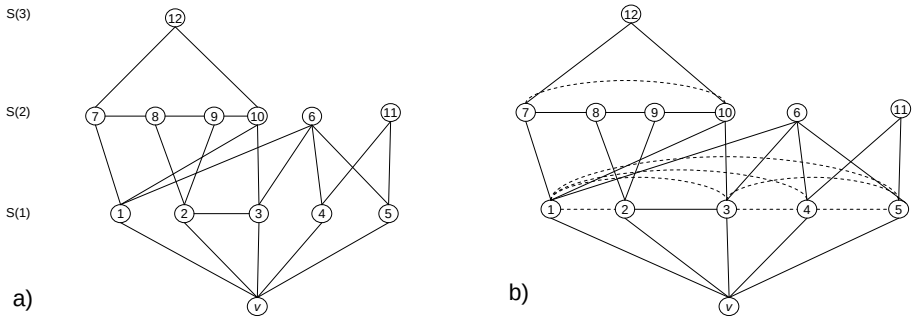

Figure 1: a) The original graph b) After saturating $N(C_i^l)$ for all $l = 1, \ldots, \delta - 1$ and $i = 1, \ldots, k_l$.

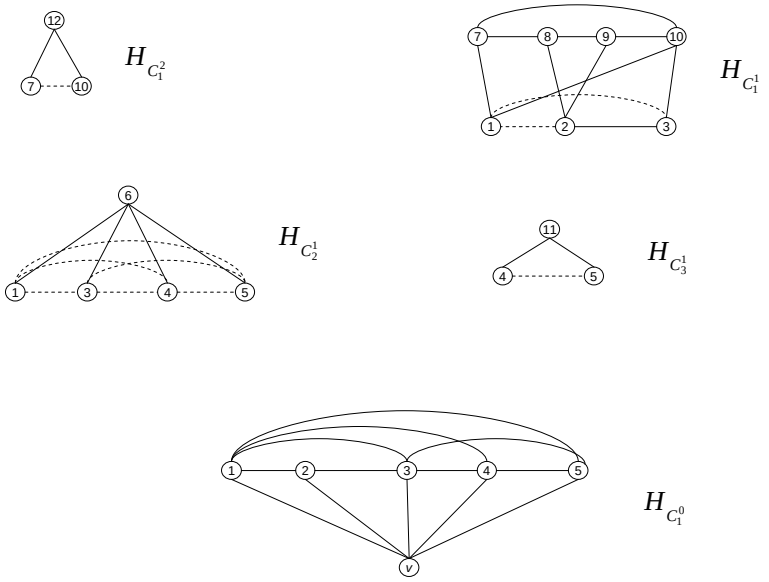

Figure 2: The detail of all $H_{C_i^l}$ of the graph of Fig 1 b)

Remark 9. *Note that $N_G(C_i^l)$ is a minimal separator of G since it has at least two full components for $l = 1, \ldots, \delta - 1$ and for $i = 1, \ldots, k_l$.*

Lemma 10. *Let D be a set of edges, $D \cap E(G) = \emptyset$, such that for all $uv \in D$ either $\{u, v\} \subseteq N_G(C_i^l)$ or $u \in S_{l+1} \cap C_i^l$ and $v \in N_G(C_i^l)$. Then*

i *the set of vertices at distance l from ν in $G + D$ do coincide with S_l for all $l = 0, \ldots, \delta$.*
ii *the component C_i^l is the same in G and $G + D$ for $l = 0, \ldots, \delta - 1$, $i = 1, \ldots, k_l$.*
iii *$N_G(C_i^l) = N_{G+D}(C_i^l)$ for $l = 0, \ldots, \delta - 1$, $i = 1, \ldots, k_l$.*

Proof: Is sufficient to show that the above statements are true when $D = \{uv\}$ is a singleton.
(i) We show that the addition of the edge uv does not change the distance of both u and v from ν. This implies also that the distances of all other vertices from ν remain unchanged in $G + D$. Suppose that both u and v are in $N_G(C_i^l)$. Any shortest path $\nu = v_0, \ldots, v_{l-1}, v_l = v$ not containing u has $v_{l-1} \in S_{l-1}$. Therefore v can decrease its distance from ν only for the existence of a shortest path $\nu = v_0, \ldots, v_k, u, v$ containing uv as the last edge which is impossible since then there exist in G a shortest path from ν to u of distance less than $l - 1$ contradicting the hypothesis that $u \in S_l$. The case when $u \in S_{l+1} \cap C_i^l$ and $v \in N_G(C_i^l)$ is similar and is omitted.
(ii) Clearly the statement is true for $l = \delta - 1$. Suppose $l < \delta - 1$ and consider first the case when $u \in S_{l+1} \cap C_i^l$ and $v \in N_G(C_i^l)$. All the components $C_i^{l^*}$ for all $l^* > l$ are unaffected by uv and therefore are the same in G and $G + D$. We have that u and v are together in the same component C_j^{l-1} since v is adjacent to a vertex of C_i^l, say x, and x and u are in C_i^l. Therefore the edge uv joins two vertices of the same component. Since u and v are contained also in each component $C_k^{l^*}$ containing C_j^{l-1} for all $l^* < l - 1$ the statements follows. The case when $\{u, v\} \subseteq N_G(C_i^l)$ is similar to the first case and is omitted.
(iii) It can be demonstrated using argument similar to the cases (i) and (ii) and is omitted. □

Lemma 11. *Let C_i^l be a component at level l, $l = 0, \ldots, \delta - 1$. Then the subgraph of $G + F$ induced by $C_i^l \cap S_{l+1}$ is connected.*

Proof: Let $Q = C_i^l \cap S_{l+1}$. This must be true if $l = \delta - 1$. Suppose $0 \leq l < \delta - 1$ and suppose by contradiction that there exist two connected components Q_1 and Q_2, of $(G + F)[Q]$ and let $x \in Q_1$ and

$y \in Q_2$. Since Q_1 and Q_2 are not connected in $(G+F)[Q]$, then they are not connected in $G[Q]$. Since x and y belong to C_i^l there is a path $p = x, a_1, \ldots, a_h, y$ in $G[C_i^l]$ connecting them. Let a_r be the first vertex of p belonging to S_{l+2} and let a_s be the first vertex of p after a_r belonging to S_{l+1}. Clearly a_{r-1} and a_s belong to $N_G(C_j^{l+1})$ for some C_j^{l+1} included in C_i^l and then the path $p' = x, a_1, \ldots, a_{r-1}, a_s, \ldots, a_h, y$ is a path of $G+F$ connecting x to y. Furthermore a_s must belong to Q_1. If we repeat the above argument substituting x with a_s and substituting p with the subpath of p starting at a_s and ending in y, we eventually will find a path of $G+F$ with vertices all in $C_i^l \cap S_{l+1}$ connecting x to y (contradiction). □

Lemma 12. *If $G+F$ is not triangulated then any chordless cycle C is entirely contained in $H_{C_i^l}$ for some component C_i^l.*

Proof: Suppose that $G+F$ is not chordal and let $C = a_0, a_1, \ldots, a_z, a_0$ be a chordless cycle not contained in some $H_{C_i^l}$. First we show that C must be contained in the subgraph of $G+F$ induced by S_l and S_{l+1} for some $0 \leq l < \delta$. In fact suppose, by contradiction, that C contains (at least) three vertices with distances respectively l, $l+1$ and $l+2$.

Suppose, w.l.o.g., that a_0 has distance $l+2$. Let a_p be the first vertex of C after a_0 having distance $l+1$ and let a_q be the last vertex of C having distance $l+1$. Note that a_p and a_q are not consecutive in C since, by hypothesis, there exists a vertex at distance l in C. Therefore a_{p-1} and a_{q+1} are connected with a path containing only vertices with distance $l+2$ or more. By (ii) of Lemma 10 we have that a_{p-1} and a_{q+1} are contained in a component C_i^{l+1} of $G+F$. It follows that a_p and a_q are in $N_{G+F}(C_i^{l+1})$ and since a_p and a_q are not consecutive, $a_p a_q$ is a chord of C (contradiction).

Now if C is entirely contained in S_{l+1} for $l = 1, \ldots, \delta$, then, by (ii) of Lemma 10 and Lemma 11 we have that C is contained in some $H_{C_i^l}$ (in particular C is contained in Q_i^l).

It remains to show that if C has a vertex in S_l and a vertex in S_{l+1} then it is entirely contained in $H_{C_k^l}$ for some C_k^l. Suppose not and suppose, w.l.o.g., that $a_0 \in S_{l+1} \cap C_k^l$. Let a_r be the first vertex of C after a_0 belonging to S_l and let a_s be the last vertex of C belonging to S_l. By (ii) of Lemma 10, we have that a_{r-1} and a_{s+1} belong together to the component C_k^l of $G+F$. It follows that a_r and a_s are in $N_{G+F}(C_k^l)$. If a_r and a_s are consecutive in C or do coincide, then C is entirely contained in $H_{C_k^l}$, otherwise $a_r a_s$ is a chord of C (contradiction). □

Theorem 13. *The graph $G+F$ is triangulated if and only if $H_{C_i^l}$ is triangulated for every C_i^l and every $l = 0, \ldots, \delta-1$ and $i = 1, \ldots, k_l$.*

Proof: Since $H_{C_i^l}$ is an induced subgraph then we need only to show that if $H_{C_i^l}$ is triangulated for every C_i^l, $l = 0 \ldots, k-1$ and $i = 1, \ldots, k_l$, then $G+F$ is chordal. Suppose not and let C be a chordless cycle of $G+F$. By Lemma 12, C is entirely contained in the subgraph $H_{C_i^l}$ of $G+F$ contradicting the hypothesis that $H_{C_i^l}$ is chordal. □

As a corollary of previous Theorem we have

Corollary 14. *Given $G+F$ as above let F_i^l the set of edges such that $H_{C_i^l} + F_i^l$ is triangulated and let*

$$F' = \bigcup_{\substack{l=0,\ldots,\delta-1 \\ i=1,\ldots,k_l}} F_i^l$$

Then $G+F+F'$ is a triangulation of G.

Then we have the following, stronger result

Theorem 15. *Given $G+F$ as above let F_i^l the set of edges such that $H_{C_i^l} + F_i^l$ is a minimal triangulation of $H_{C_i^l}$ and let*

$$F' = \bigcup_{\substack{l=0,\ldots,\delta-1 \\ i=1,\ldots,k_l}} F_i^l$$

Then $G+F+F'$ is a minimal triangulation of G.

Proof: By Corollary 14 we need only to show that $G^+ = G + F + F'$ is minimal. By Lemma 2, we shall show that, for each edge uv of $F \cup F'$, $CN_{G^+}(uv)$ is not empty and is not a clique.

By (ii) and (iii) of Lemma 10, $N_{G^+}(C_i^l) = N_G(C_i^l)$ and the vertex set of each connected component of $G^+ - S_l$ do coincide with the vertex set of each connected component of $G - S_l$ for all $l = 1, \ldots, \delta - 1$ and $i = 1 \ldots, k_l$. By Remark 9, we have that $N_{G^+}(C_i^l)$ is a minimal separator of G^+. Since G^+ is chordal then, by Lemma 5, $CN_{G^+}(uv)$ is not empty nor is a clique for every pair of vertices u, v of $N_{G^+}(C_i^l)$ and therefore for every $uv \in F$.

Now let uv be an edge of F' and let $H = H_{C_i^l} + F_i^l$ be the subgraph of $G + F + F'$ containing uv. Since H is a minimal triangulation of $H_{C_i^l}$ then $CN_H(uv)$ is not empty and is not a clique. But since H is an induced subgraph of G' then $CN_H(uv) \subseteq CN_{G^+}(uv)$ and $CN_{G^+}(uv)$ is not empty and is not a clique. □

In order to obtain the graphs $H_{C_i^l}$ we have to saturate first $N_G(C_i^l)$ for all $l = 0, \ldots, \delta - 1$ and for all $i = 1, \ldots, k_l$ which can be done with the technique of matrix multiplication as explained, for example, in [14]. Note that only separators at level l can overlap each other, while separators at different level obviously do not overlap. We may saturate $N_G(C_i^l)$ in reverse order starting from the top level down to the first level of the BFS. We compute all the quasi-split graphs at level l and saturate $N_G(C_i^l)$ for $i = 1, \ldots, k_l$. This can be done by simply visiting all the connected components of $G + F$ induced by S_{l+1} since, by Lemma 11, Q_i^l is connected for $i = 1, \ldots, k_l$. The set P_i^l can be computed by determining, for each vertex v of Q_i^l all the neighbors of v at the lower level.

4. Minimal triangulation of quasi-split graph

By Theorem 15 we have seen that we can obtain a minimal triangulation of a graph by finding a minimal triangulation of each $H_{C_i^l}$, $l = 0, \ldots, \delta - 1$, $i = 1, \ldots, k_l$, which is a quasi-split graph. In the following we concentrate on the task of computing a minimal triangulation of a quasi-split graph.

4.1. Two special cases

Let G be a quasi-split graph with bipartition (P, Q). First we consider two special cases, which are interesting on their own and are useful in the subsequent discussion. The first case is when Q induces a clique. The second case is when we first compute a minimal triangulation of $G[Q]$ (using any minimal triangulation algorithm) or when $G[Q]$ is already chordal.

In the first case, let B_G be the bipartite graph obtained from G by deleting all the edges with both endpoints in Q and deleting all the edges with both endpoints in P. It was shown in [23] that any minimal chain completion of B_G is a minimal triangulation of G. In this case the problem of finding a minimal triangulation of G reduces to the problem of finding a *minimal chain completion* of a bipartite graph B, that is finding a minimal set of edges which added to B make it a chain graph. An algorithm for computing a minimal chain completion of B is as follows. We select at each step a vertex $v \in Q$ such that $d_B(v)$ is minimum. For each vertex $w \in N_B(v)$ we add to B the edge wz for all $z \in Q \setminus v$. Then we delete v from Q and B. The algorithm[1] MinChainCompletion is reported in Figure 3.

Lemma 16. *The algorithm* MinChainCompletion *produces a minimal chain completion D of a bipartite graph B.*

Proof: First we show that the algorithm produces a chain completion of D. Let v_i and D_i be, respectively the vertex examined and the set of edges added at the end of step i, $i = 1, \ldots |Q|$, and let $D_0 = \emptyset$. Clearly at the end of step i we have that $N_{B+D_i}(v_i) \subseteq N_{B+D_i}(v_j)$ for all v_j, $j > i$. Since after the step i the vertex

[1]The algorithm given in [13] for finding a minimal chain completion of a bipartite graph is quite different from the one given here. However since our algorithm is based on the choice of a minimum degree vertex we think it is safer to use ours instead of theirs.

Algorithm MinChainCompletion
input : A bipartite graph B with bipartition (P, Q)
output : A minimal chain completion D of B
begin
  repeat
    let $v \in Q$, such that $d_B(v)$ is minimum;
    for all $z \in Q \setminus v$ and **for all** $w \in N_B(v)$ **do**
      if $wz \notin E(B) \cup D$ **then** add wz to B and D ;
    delete v from Q and B;
  until Q is not empty;
end

Figure 3: The MinChainCompletion algorithm

v_i is deleted, then its neighborhood remains unchanged in all subsequent steps of the algorithm. Therefore at the end of the algorithm we have that $N_{B+D}(v_1) \subseteq N_{B+D}(v_2) \cdots \subseteq N_{B+D}(v_{|Q|})$, i.e., the graph $B + D$ is a chain graph.

Now we show the minimality. By Lemma 6, it is sufficient to show that for every edge $xy \in D$ there exist two independent edges of B in $B + (D \setminus \{xy\})$. Suppose that the edge xy is added when the vertex v_i is being considered. Suppose, w.l.o.g., that $x \in N_{B+D_{i-1}}(v_i)$ and that $y = v_j$ for some $j > i$. We have that $v_i x \in E(B)$ for otherwise if $v_i x \in D$ then $v_i x$ would have been added at step $i^* < i$, $x \in N_{B+D_{i^*-1}}(v_{i^*})$ and then at the step i^* also the edge xv_j would have been added contradicting the hypothesis that xv_j was added at step i. The vertex v_i is selected at step i because $d_{B+D_{i-1}}(v_i)$ is minimum and since $x \notin N_{B+D_{i-1}}(v_j)$ there must exist a vertex $z \in N_{B+D_{i-1}}(v_j) \setminus N_{B+D_{i-1}}(v_i)$. It follows that $v_j z \in E(B)$. We have that $v_i x$ and $v_j z$ are independent in $B + (D \setminus \{xy\})$, since at the end of step i, v_i is deleted and the edge zv_i will never be added to D. □

In order to achieve a time complexity of $O(n + m + |D|)$ we propose the following implementation. The neighborhood of the vertex v_i need not to be reconsidered in the subsequent steps since it is already included in the neighborhood of all the remaining vertices. Furthermore at step i we can detect all the vertices v_j, $j > i$ such that $N_{B+D_{i-1}}(v_j) = N_{B+D_{i-1}}(v_i)$ without increasing the complexity. In fact in order to detect them we note that these vertices have the same degree of v_i in $B + D_{i-1}$ and for all $w \in N_{B+D_{i-1}}(v_i)$ we have that wv_j is an edge of $B + D_{i-1}$. Therefore at each step of the algorithm we delete all the neighborhood of v_i and all vertices having the same neighborhood of v_i in $B + D_{i-1}$. By the above discussion we can state the following

Lemma 17. *The algorithm* MinChainCompletion *has time complexity of* $O(n + m + |D|)$.

By Lemma 16 and Lemma 17 we are able to compute a minimal triangulation of a quasi-split graph G in time linear in the dimension of the triangulated graph when Q is a clique. If Q does not induces a clique and $|P| = 1$, in order to obtain a minimal triangulation of G we have to compute a minimal triangulation of $G[Q]$.
Now we deal with the case in which either $G[Q]$ is triangulated or we compute first a minimal triangulation of it. Let $|P| > 1$ and let F_1 be the fill-in of a minimal triangulation of $G[Q]$. We may obtain a minimal triangulation of G with the following algorithm. We examine each vertex of Q and at each step we add some edges (to be explained later) to the set variable D_i. At the beginning $D_0 \leftarrow F_1$. We select, at each step, a vertex v_i which is simplicial in $G[Q] + F_1$ and such that $|N_{G+D_{i-1}}(v_i) \cap P|$ is minimum. Then if E_i is the set of edges needed to saturate $N_{G+D_{i-1}}(v_i)$ we set $D_i \leftarrow D_{i-1} \cup E_i$ and delete v_i from $G + D_i$. The algorithm MinQSCompletion is reported in Figure 4. We now prove the correctness of the algorithm. First we have the following

Lemma 18. *Let G be a quasi-split graph with bipartition (P, Q). Let F_1 be a set of edges such that $G[Q]+F_1$ is a minimal triangulation of $G[Q]$ and let F_2 be a set of edges such that $G+F_1+F_2$ is a minimal triangulation of $G+F_1$ and such that each edge of F_2 has one endpoint in Q and the other endpoint in P. Then $G+F_1+F_2$ is a minimal triangulation of G.*

Proof: Let $G^+ = G+F_1+F_2$. Since each edge of F_2 has one endpoint in P we have that $G^+[Q] = G[Q]+F_1$. Since G^+ is a minimal triangulation of $G + F_1$, clearly each edge of F_2 is the unique chord of a four cycle of G^+. Suppose, by contradiction, that an edge $uv \in F_1$ is not a unique chord in a four cycle of G^+, that is, by Lemma 2, $CN_{G^+}(uv)$ is either empty or is a clique of G^+. It follows that $CN_{G^+}(uv) \cap Q$ is either empty or is a clique. But since $G^+[Q] = G + F_1$ then $CN_{G^+}(uv) \cap Q = CN_{G[Q]+F_1}(uv)$ is either empty or is a clique contradicting the hypothesis that $G[Q] + F_1$ is a minimal triangulation of $G[Q]$. □

Algorithm MinQSCompletion
input : A quasi-split graph G
output : A minimal triangulation of G
begin
  let $G' = G[Q] + F_1$ be a minimal triangulation of $G[Q]$;
  $D \leftarrow F_1$;
  repeat
    let v be a simplicial vertex of G' such that $|N_{G+D}(v) \cap P|$ is minimum;
    add to D the edges needed to saturate $N_{G+D}(v)$;
    delete v from $G + D$ and G';
  until G' is not empty;
end

Figure 4: The MinQSCompletion algorithm

Theorem 19. *Let G be a quasi-split graph with bipartition (P, Q). The algorithm* MinQSCompletion *computes a set of edges F_2 such that $G + F_1 + F_2$ is minimal triangulation of G.*

Proof: Let $G' = G[Q] + F_1$ and let v_i be the vertex considered at the step i. Denote by D the union of the edges added in all the steps before i. We shall show that the set of edges needed to saturate $N_{G+D}(v_i)$ are exactly those needed to make v_i LB-simplicial.

Since v_i is simplicial in G' then $K_i = N_{G+D}(v_i) \cap Q$ is a clique. Let $P_i = N_{G+D}(v_i) \cap P$, S_i be the set of vertices of K_i which are simplicial in G' and let $S'_i = \{w \in S_i : N_{G+D}(w) \cap P = P_i\}$.

Since v_i is chosen because it is simplicial in G' and $|N_{G+D}(v) \cap P|$ is minimum we have that each vertex in $P_i \cup (S_i \setminus S'_i)$ is adjacent to at least one vertex of $P \setminus P_i$. We also have that each vertex of $K_i \setminus S_i$ is adjacent to at least one vertex of $Q \setminus K_i$. Let $C_1, \ldots, C_h$ be the connected components of the subgraph of $G + D$ induced by $V(G) \setminus N_{G+D}[v_i]$. For each C_j we show that $P_i \subseteq N_{G+D}(C_j)$. In fact this is clearly true if C_j contains $P \setminus P_i$. Otherwise if C_i does not contain $P \setminus P_i$ then, by Lemma 7, it contains at least one simplicial vertex u of G'. By hypothesis each simplicial vertex u of G' has $|N_{G+D}(u) \cap P| \geq |P_i|$ and since u is not adjacent to any vertex of $P \setminus P_i$ then we have that $N_{G+D}(u) \cap P = P_i$.

Now note that by the above discussion the component C_j containing $P \setminus P_i$ has $S_i \setminus S'_i \subseteq N_{G+D}(C_j)$. Since each vertex of $K_i \setminus S_i$ has at least one vertex adjacent to $Q \setminus K_i$ then we have $\bigcup_{j=1\ldots,h} N_{G+D}(C_j) \cap K_i = K_i \setminus S'_i$. Therefore in order to saturate each $N_{G+D}(C_j)$ we have to add an edge between each vertex of P_i to each vertex of $K_i \setminus S'_i$. This is also sufficient since both K_i and P_i are cliques. Finally since each vertex of S'_i is connected to every other vertex of $P_i \cup K_i$ in order to make v_i LB-simplicial it is sufficient to saturate $P_i \cup K_i = N_{G+D}(v_i)$.

By Lemma 4, $D \setminus F_1$ is the set of edges needed to obtain a minimal triangulation of $G+F_1$. Since v_i is simplicial in G', no edge being added for saturating $N_{G+D}(v_i)$ has both endpoints in Q. Finally note that when

Algorithm MinQS
input : A quasi-split graph G
output : A triangulation of G
begin
  $D \leftarrow \emptyset$;
  repeat
    let $v \in Q$ such that $|N_{G+D}(v) \cap P|$ is minimum;
    add to D the edges needed to saturate $N_{G+D}(v)$;
    delete v from G;
  until Q is not empty;
end

Figure 5: The MinQS algorithm

we remove v_i from $G+D$ the graph remains a quasi-split graph. The Theorem then follows by Lemma 18. □

Given a quasi-split graph G with bipartition (P, Q) it is interesting to note that we may obtain a triangulation of G (not necessarily minimal) with the following simple algorithm. Repeatedly select a vertex of Q such that $|N_G(v) \cap P|$ is minimum. Add to G the edges needed to saturate $N_G(v)$ and delete v (see Figure. 5). We have the following

Lemma 20. *The algorithm MinQS produces a minimal triangulation if, when the vertex v is selected, then $N(v) \cap Q$ does not contain any vertex w such that $N(w) \cap P = N(v) \cap P$.*

Proof: Let $w \in N(v) \cap Q$ and let $P' = N(v) \cap P$. Since $N(w) \cap P \neq P'$ and since v is chosen because $|P'|$ is minimum we have that there exist a $z \in N(w) \cap P$ such that $z \notin P'$. Since this is valid for all $w \in N(v) \cap Q$ we have that $N(P \setminus P')$ contains $N(v) \cap Q$ and since P is a clique then $N(P \setminus P')$ contains $N(v)$. Therefore in order to make v LB-simplicial we have to saturate $N(v)$. The Lemma then follows from Lemma 4. □

Note that in all the above algorithms choosing a vertex of minimum degree is not an option but a necessity if we want to find minimal triangulations. We think this is interesting from the theoretical point of view since this sheds a new light (together with [4]) on why the local minimum degree heuristic is so effective in finding minimal triangulations.

5. Minimal triangulation by recursion

In the previous section we have seen a simple algorithm for finding a minimal triangulation of a quasi-split graph, the MinQSCompletion algorithm. However in order to obtain a more efficient algorithm we may use the strategy of triangulating the graph $H_{C_i^l}$ by recursively apply to it the same partitioning procedures. That is, we execute again a BFS on each $H_{C_i^l}$ and recursively saturate the minimal separators. When Q_i^l is a single vertex or a clique, we may stop the recursive calls or solve the problem using the algorithm MinChainCompletion.

However we have to be careful in order to do not revisit in each of the recursive call again those parts of the graph which overlap with the other quasi-split graphs. To this end, a quasi-split graph G with bipartition (P, Q), will be represented by the following data structure. We use adjacency lists $Adj(v)$ for each vertex v. If $v \in Q$ then the adjacency list $Adj(v)$ is equal to $N_G(v)$. If $v \in P$ then we set $Adj(v)$ equal to $N_G(v) \cap Q$.

Now the recursive call can be done as follows. We have to chose a vertex $\nu \in P$. We visit all the vertices in $N_G(\nu) \cap Q$. Then we note that in a quasi-split graph, each vertex of P is at distance at most two from any other vertex of G. Hence $\delta \leq 2$. If $\delta = 1$, that is, $N_G(\nu) \cap Q = Q$, by Lemma 4, we simply delete ν

since it is already LB-simplicial. If every vertex of P is adjacent to every vertex of Q, by Lemma 12, we can continue the recursive call by finding a minimal triangulation of the subgraph of G induced by Q.

Otherwise $\delta = 2$. In this case we find the connected components of $G \setminus N_G(\nu)$ by starting a visit from any vertex of $Q \setminus N_G(\nu)$. Let $\mathcal{C} = \{C_1 \ldots, C_h\}$ be the connected components of $G \setminus N_G(\nu)$. Then for each C_i we obtain a quasi-split graph by taking the subgraph of G induced by $C_i \cup N_G(C_i)$.

Let F be the set of edges needed to saturate $N_G(C_i)$ for all $i = 1, \ldots, h$. Since in a quasi-split graph no single vertex is a separator then there is only one component at level 0. Therefore it remains to triangulate the quasi-split graph induced by $N_{G+F}[\nu]$. This means that we have to triangulate the subgraph H of $G+F$ induced by $N_{G+F}(\nu)$ which is, by Lemma 11, connected. Note that there can be a vertex w of $N_G(\nu) \cap Q$ that is not adjacent to any vertex of $P \setminus \nu$ nor adjacent to any vertex of C_i for any $i = 1, \ldots, h$. It follows that H needs not to be a quasi-split graph.

We may visit H without revisiting the edges with both endpoints in P, by using a slightly modified BFS. Recall that in the representation of G the adjacency list of each vertex w of P contains only $N_G(w) \cap Q$. Taking this into consideration, we execute a BFS by initializing the FIFO queue[2] with all the vertices of P. In this way we visit the graph H without revisiting the edges with both endpoints in P.

By the discussion above we have that each recursive call spent time visiting only edges of a subgraph of G which are non-overlapping with the edges of other subgraphs. The algorithm QSTriang is reported in Figure 8.

Example. Consider the quasi-split graph of Figure 6(a) with bipartition $P = \{a, b, c\}$ and $Q = \{d, e, f, g\}$. Its internal representation is shown in Figure 6 (b). We chose to start the BFS from vertex a. After this we have that there is only one component of $G - N_G(a)$ which is $C = \{g\}$ (see Figure 7(a)). We saturate $N_G(g)$ (without reinserting the edge bc) by adding to G the set of edges $F = \{fb, fc\}$ (see Figure 7(a)). After this, we have that the subgraph of $G + F$ induced by $\{b, c, f, g\}$ is a quasi-split graph with bipartition (P', Q') where $Q' = \{g\}$ is a singleton and therefore is triangulated. So it remains to triangulate the subgraph of $G + F$ induced by $N_G(a)$, which is depicted in Figure 7(b). When we execute the recursive call we insert in the FIFO queue, at the beginning of the BFS, the vertices b and c. After the BFS we have two quasi-split graphs depicted in Figure 7(c) and (d) where the edge fd has been added. These two quasi-split graphs are already triangulated and the algorithm terminates. Note that the edges ab, ac and bc are never revisited during all steps of the algorithm.

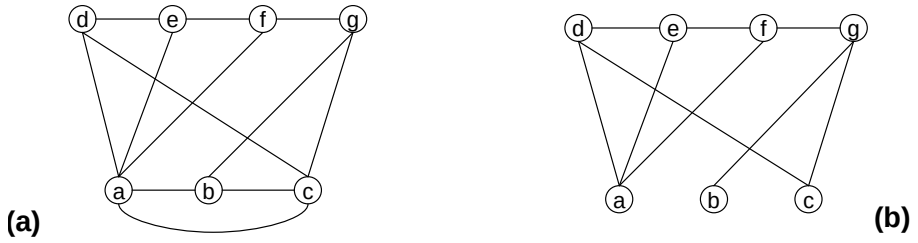

Figure 6: (a) A quasi-split graph G with bipartition $P = \{a, b, c, \}$ and $Q = \{d, e, f, g\}$ (b) Its data representation.

6. Complexity and experimental results

In order to lower the complexity of the algorithm we have to partition the graph in a way that the largest graph $H_{C_i'}$ has a dimension which is a fraction of the input graph at each recursive call. The dimension

[2]Recall that the BFS algorithm uses a FIFO queue to select at each step the next vertex to visit. Normally in the BFS algorithm this queue is initialized by inserting in it only one vertex, that is the starting vertex of the visit.

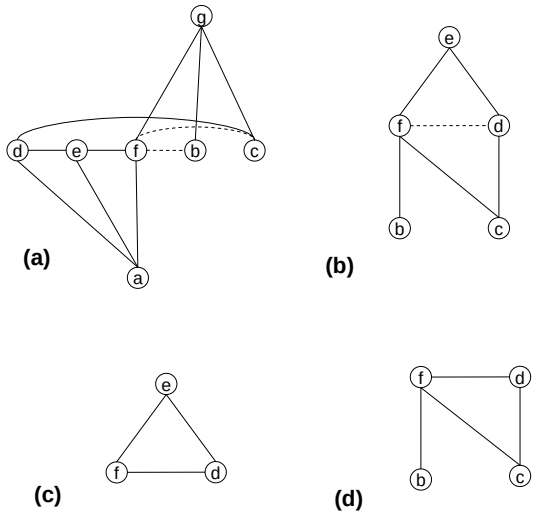

Figure 7: (a) After visiting $N_G(a)$ and after the set F of dotted edges has been added (b) The subgraph H of $G + F$ induced by $N_G(a)$. (c) and (d) the quasi-split graphs of H.

of the subgraphs heavily depend on the choice of the vertex from which start the BFS. In fact if, in every recursive call, the degree $d_G(\nu)$ of the vertex from which start the BFS satisfies

$$\frac{n}{k} \leq d_G(\nu) \leq \frac{k-1}{k}n$$

for some (small) constant $1 < k < n$ we have, in the worst case, at least two quasi-split graphs: one in which Q has $d_G(\nu)$ vertices and the other in which Q has $|V(G)| - d_G(\nu)$ vertices. Since the visit of these graphs will be done without revisiting two or more time the overlapping edges, the algorithm terminates after a logarithmic number of steps.

The simplest choice is that of selecting, at the beginning of the algorithm and at each recursive calls, a vertex of minimum degree. We may hope that the selection of a vertex of minimum degree breaks the graph in many smaller subgraphs.

Such a choice, however, does not always assure a balanced partition as we can see in the following example. Suppose, in the worst case, that after the BFS we have only two subgraphs: $H_{C_1^0}$ and $H_{C_1^1}$ such that P_1^1 has k vertices, with k much smaller than n and Q_1^1 has $n-k-1$ vertices. Then suppose again that in P_1^1 there is a vertex such that $|N_G(v) \cap Q_1^1| = k$ so that in the successive recursive call, this vertex is selected as the starting vertex of the BFS. And so on. The algorithm therefore will terminates after $O(n)$ recursive calls.
A better choice is that of finding at each recursive call a vertex v whose degree is such that the absolute value $abs(d_G(v) - |Q|/2)$ is as minimum as possible. However such a vertex could not exist at all.

In fact it can happen that at any of the recursive calls, the graph has minimum degree of $n - k$ with k much smaller than n. If k is very small, that is, the graph is very dense, we may even change completely the strategy and could be convenient to find a minimal triangulation of the graph using recent minimal triangulation technique (such as those proposed in [19, 18]) which are very fast when the graph is dense.

Anyway, the choices discussed above do not seem, at first, to assure a balanced partition of the graph. For this reason we made an extensive experimentation, by building an implementation of the recursive algorithm. We executed a time test both on a set of random graphs and on most of the sparse symmetric matrices from the Harwell-Boeing collection [8] (HBC), comparing the execution time of our algorithm with the execution time of MCS-M. In both cases we found that, even without implementing the matrix multiplication technique for saturating the minimal separators and focusing exclusively on the correctness of the code rather than speed optimization, our algorithm performs dramatically better than MCS-M.

```
Algorithm QSTriang(G, L)
input     : A connected graph G and a list L of vertices
            (in the initial call L contains only one vertex)
output    : The fill F a minimal triangulation of G
begin
   F ← ∅;
   BFS(G, L);
   for l = δ − 1 down to 0 do
   begin
      for each quasi-split graph H_{C_i^l} at level l do
      begin
         F' =RecursiveQS(H_{C_i^l});
         let F_i be the set of edges needed to saturate P_i^l;
         add F_i to G;
         F ← F ∪ F' ∪ F_i;
      end
   end
   return F;
end

RecursiveQS(G)
input     : A quasi-split graph G with bipartition (P, Q)
output    : The fill F a minimal triangulation of G
begin
   F ← ∅;
   if |Q| = 1 return F;
   let w be a vertex of P;
   for each quasi-split graph H_i determined by N_G(w) do
   begin
      F' =RecursiveQS(H_i);
      let F_i be the set of edges needed to saturate P_i;
      add F_i to G;
      F ← F ∪ F' ∪ F_i;
   end
   let H be the subgraph of G induced by N_G(w);
   F' =QSTriang(H, P);
   return F ∪ F';
end
```

Figure 8: The QSTriang algorithm

We generated a number of pseudo random graphs of various dimension. In Table 1 we reported for each of the generated graphs the number of its vertices and edges. We also reported the fill produced by both algorithms and the execution time.

n	m	Fill QS	QS	Fill MCS-M	MCS-M
800	7990	228189	2.372	234598	4.228
800	9588	246730	2.324	243498	4.648
800	11185	257483	2.309	250640	5.164
800	12784	261826	2.886	257466	5.320
800	14381	265641	3.292	262488	5.351
800	15980	265133	4.103	263835	5.632
900	10113	303993	2.793	299887	6.927
900	12136	327265	3.386	315491	7.707
900	14159	333731	3.510	319030	7.816
900	16182	332288	4.072	328661	7.738
900	18204	343652	5.148	331788	7.831
900	20227	343206	5.538	338643	8.736
1000	12487	389026	3.542	386072	10.405
1000	14985	395903	3.805	400220	10.764
1000	17482	412777	4.165	413503	11.498
1000	19980	418592	5.429	415421	11.763
1000	22477	432771	7.192	418214	12.168
1000	24975	422861	8.829	424977	12.168
1100	15111	485277	4.884	474137	14.150
1100	18133	499121	5.116	493601	15.678
1100	21155	516757	7.816	504854	16.380
1100	24178	514179	8.534	515573	17.083
1100	27200	518418	9.906	517819	17.800
1100	30222	519751	13.182	518798	19.344
1200	17985	598528	6.833	578007	21.497
1200	21582	605929	8.517	589538	22.667
1200	25178	609721	9.782	604516	23.837
1200	28776	617493	12.479	614348	23.681
1200	32372	621880	14.211	614822	25.568
1200	35970	620969	18.284	622795	26.535
1300	21108	697885	8.533	694875	28.642
1300	25330	715600	9.235	716900	32.402
1300	29552	728187	11.871	705216	32.792
1300	33774	730373	16.802	727819	36.582
1300	37995	740234	19.594	724023	37.580
1300	42217	739860	22.511	731216	39.218

Table 1: Data of the test on random generated graphs. The column n and m report, respectively, the number of vertices and of edges of the graph. Columns *Fill QS*, *Fill MCS-M*, *QS* and *MCS-M* report, respectively, the fill and the execution time (in seconds) of both algorithms.

We computed for the matrices of the HBC the number of quasi-split graphs obtained after the first iteration of the recursive algorithm. We tested two cases. In the first one we execute the first BFS, starting from a vertex of minimum degree. In the second case we execute the first BFS, starting from a vertex whose degree is *median*, that is, was the closest to $n/2$. In Table 2 we report the number of quasi-split graphs produced for each of the two cases. Note that, selecting a vertex of minimum degree is in almost all cases, slightly better than selecting a vertex of median degree. We report also the execution time of QSTriang where, in the first BFS, is selected a vertex of minimum degree, while in the recursive calls a vertex of median degree is selected. Moreover the execution time of MCS-M is reported.

We also observed that, on average, at the end of the first BFS, the 44% of the fill edges are already added to G.

7. Conclusions

We presented an algorithm for computing a minimal triangulation of a graph which exploits a recursive partitioning of the graph on the basis of a BFS visit. Its computational complexity is shown to be strictly related to the complexity of boolean matrix multiplication. We also presented variants of this algorithm in which the minimum degree criterion is used thus shedding new light on the effectiveness of the minimum degree heuristic. Experimental results are provided showing that the computation time of the proposed algorithm is dramatically better than well known existing minimal triangulation techniques on sparse matrices.
There are several future research issues. The partitioning scheme based on the choice of the minimum degree vertex does not guarantee a balanced partition of the graph. We could investigate if there exists a different partitioning scheme that guarantees a balanced partition in order to obtain an algorithm whose complexity is $O(n^{\alpha} \log n)$. By Theorem 15 the problem of computing a minimal triangulation of a graph reduces to the problem of computing a minimal triangulation of a quasi-split graph. In the special case when, in a quasi-split graph with bipartition (P, Q), the set Q induces a clique, we are able to compute a minimal triangulation in time linear in the dimension of the triangulated graph (Lemma 17). We could investigate which is the complexity of existing minimal triangulation algorithms (e.g. MCS-M) when in a quasi-split graph the subgraph induced by Q is already chordal. We could investigate both experimentally and theoretically the behavior of the MinQS algorithm: how is effective in computing minimal triangulations and which is its time complexity? We could investigate both experimentally and theoretically the possibility and the effectiveness of mixing the techniques used in [18, 19] with the QST algorithm for computing minimal triangulations of dense graphs.

Acknowledgment

I wish to thank the anonymous referee for his comments.

Matrix name	n	m	QS min	QS med	QS (sec.)	MCS-M (sec.)
bcspwr05.mtx	443	590	215	217	0.016	0.078
bcspwr06.mtx	1454	1923	662	661	0.078	1.669
bcspwr07.mtx	1612	2106	798	793	0.093	2.028
bcspwr08.mtx	1624	2213	744	735	0.093	2.293
bcspwr09.mtx	1723	2394	762	764	0.109	2.793
bcspwr10.mtx	5300	8271	1165	1165	2.059	86.160
bcsstk06.mtx	420	3720	49	47	0.109	0.172
bcsstk07.mtx	420	3720	49	47	0.110	0.140
bcsstk08.mtx	1074	5943	53	52	3.792	5.210
bcsstk09.mtx	1083	8677	18	19	0.982	2.168
bcsstk10.mtx	1086	10492	71	69	0.156	0.873
bcsstk11.mtx	1473	16384	69	68	0.593	3.682
bcsstk12.mtx	1473	16384	69	68	0.640	3.650
bcsstk13.mtx	2003	40940	15	20	8.767	85.754
bcsstk14.mtx	1806	30824	24	21	3.573	19.453
bcsstk15.mtx	3948	56934	18	19	32.387	323.889
bcsstk20.mtx	485	1325	198	192	0.015	0.063
bcsstk21.mtx	3600	11500	107	105	2.527	25.991
bcsstk23.mtx	3134	21022	30	43	22.932	66.346
bcsstk24.mtx	3562	78174	46	44	5.912	90.247
bcsstk25.mtx	15439	118401	380	166	58.890	5564.509
bcsstk26.mtx	1922	14207	88	76	1.607	10.000
bcsstk27.mtx	1224	27451	36	34	0.359	3.166
bcsstk28.mtx	4410	107307	39	34	10.343	527.440
bcsstm07.mtx	420	3416	48	46	0.109	0.109
bcsstm10.mtx	1086	10503	71	69	0.156	0.952
bcsstm12.mtx	1473	9093	103	102	0.343	2.449
bcsstm13.mtx	2003	9970	6	7	0.827	5.991
bcsstm27.mtx	1224	27451	36	34	0.359	3.401
bfw782b.mtx	782	5200	77	76	0.047	0.406
blckhole.mtx	2132	6370	36	34	4.836	8.752
dwt__419.mtx	419	1572	19	22	0.125	0.125
dwt__492.mtx	492	1332	102	83	0.047	0.063
dwt__503.mtx	503	2762	27	16	0.171	0.375
dwt__512.mtx	512	1495	101	95	0.063	0.078
dwt__592.mtx	592	2256	45	41	0.140	0.156
dwt__607.mtx	607	2262	53	51	0.109	0.219
dwt__758.mtx	758	2618	111	105	0.079	0.281
dwt__869.mtx	869	3208	78	73	0.109	0.514
dwt__878.mtx	878	3285	47	30	0.141	0.593
dwt__918.mtx	918	3233	49	45	0.250	0.687
dwt__992.mtx	992	7876	30	29	0.343	1.404
dwt_1005.mtx	1005	3808	43	41	0.453	1.014
dwt_1007.mtx	1007	3784	54	48	0.296	0.843
dwt_1242.mtx	1242	4592	36	42	1.139	1.513
dwt_2680.mtx	2680	11173	104	104	1.342	12.418
s1rmq4m1.mtx	5489	137811	30	29	20.093	700.257
s1rmt3m1.mtx	5489	107016	60	32	8.518	551.761
s2rmq4m1.mtx	5489	137811	30	29	20.593	678.574
s2rmt3m1.mtx	5489	107016	60	32	8.893	548.937
s3rmq4m1.mtx	5489	137811	30	29	20.764	695.577
s3rmt3m1.mtx	5489	107016	60	32	8.923	548.141
s3rmt3m3.mtx	5357	101169	58	38	7.442	274.312
sstmodel.mtx	3345	9702	338	323	0.562	19.312
young1c.mtx	841	3248	56	57	0.281	0.453
young2c.mtx	841	3248	56	57	0.188	0.484
young3c.mtx	841	3147	49	48	0.344	0.608
young4c.mtx	841	3248	56	57	0.234	0.514
zenios.mtx	2873	12159	253	241	0.172	11.029

Table 2: The column n and m contain, respectively, the number of vertices and the number of edges of the graph. The columns "*QS min*" and "*QS med*" contain the number of quasi-split graphs after the first BFS, when the BFS is started from a vertex of minimum degree in the first case and when the BFS is started from a vertex of median degree in the second case. The columns "*QS*" and "*MCS-M*" contain the execution time of the algorithm QSTRiang and MCS-M, respectively. Only the result of the matrices with more than 400 vertices are reported

References

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

[2] 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.

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

[4] A. Berry, E. Dahlhaus, P. Heggernes, and G. Simonet. Sequential and parallel triangulating algorithms for elimination game and new insights on minimum degree. *Theor. Comput. Sci.*, 409(3):601–616, 2008.

[5] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. In H. L. Bodlaender, editor, *WG*, volume 2880 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2003.

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

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

[8] R. Boisvert, R. Pozo, K. Remington, B. Miller, and R. Lipman. *NIST MatrixMarket*, http://math.nist.gov/MatrixMarket/index.html.

[9] A. Deshpande, M. N. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In *UAI*, pages 128–135, 2001.

[10] M. Farber and R. E. Jamison. Convexity in graphs and hypergraphs. *SIAM J. Algebraic Discrete Methods*, 7(3):433–444, 1986.

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

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

[13] P. Heggernes and C. Papadopoulos. Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. *Theor. Comput. Sci.*, 410(1):1–15, 2009.

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

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

[16] 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.

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

[18] M. Mezzini. Fully dynamich algorithm for chordal graph with $O(1)$ query time and $O(n^2)$ update-time. *submitted*, 2011.

[19] M. Mezzini and M. Moscarini. Simple algorithms for minimal triangulation of a graph and backward selection of a decomposable markov network. *Theor. Comput. Sci.*, 411(7-9):958–966, 2010.

[20] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3):171–188, 1997.

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

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

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