Computing Minimal Triangulations in Time $O(n^{\alpha} \log n) = o(n^{2.376})$

Pinar Heggernes* Jan Arne Telle* Yngve Villanger*

Abstract

The problem of computing minimal triangulations of graphs, also called minimal fill, was introduced and solved in 1976 by Rose, Tarjan, and Lueker [17] in time $O(nm)$, thus $O(n^3)$ for dense graphs. Although the topic has received increasing attention since then, and several new results on characterizing and computing minimal triangulations have been presented, this first time bound has remained the best. In this paper we introduce an $O(n^{\alpha} \log n)$ time algorithm for computing minimal triangulations, where $O(n^{\alpha})$ is the time required to multiply two $n \times n$ matrices. The current best known α is less than 2.376, and thus our result breaks the long standing asymptotic time complexity bound for this problem. To achieve this result, we introduce and combine several techniques that are new to minimal triangulation algorithms, like working on the complement of the input graph, graph search for a vertex set A that bounds the size of the connected components when A is removed, and matrix multiplication.

1 Introduction and motivation

Any graph can be embedded in a chordal graph by adding a set of edges called fill, and the resulting graph is called a triangulation of the input graph. When the added set of fill edges is inclusion minimal, the resulting triangulation is called a minimal triangulation. The first algorithms for computing minimal triangulations were given in independent works of Rose, Tarjan, and Lueker [17], and Ohtsuki, Cheung, and Fujisawa [13, 14] already in 1976. Among these, the algorithms of [13] and [17] have a time bound of $O(nm)$, where n is the number of vertices and m is the number of edges of the input graph. These first algorithms were motivated by the need to find good pivotal orderings for Gaussian elimination, and the mentioned papers gave characterizations of minimal triangulations through minimal elimination orderings. Since then, the problem has received increasing attention, and several new characterizations of minimal triangulations connected to minimal separators of the input graph have been given [5, 10, 15], totally independent of the connection to Gaussian elimination. The connection to minimal separators has increased the importance of minimal triangulations from a graph theoretical point of view, and minimal triangulations have proved useful in reconstructing evolutionary history through phylogenetic trees [9]. As a result, algorithms based on the new characterizations have been given [3, 8], while at the same time new algorithms based on elimination orderings also appeared [4, 7, 16]. However, the best time bound remained unchanged, and trying to break the asymptotic $O(n^3)$ bound of computing minimal triangulations, in particular for dense graphs, became a major theoretical challenge concerning this topic.

In this paper, we introduce an $O(n^{\alpha} \log n)$ time algorithm to compute minimal triangulations of arbitrary graphs, where $O(n^{\alpha})$ is the time bound of multiplying two $n \times n$ matrices. Currently the lowest value of α is $2.375 < \alpha < 2.376$ by the algorithm of Coppersmith and Winograd [6]. Hence the current time bound for our algorithm is $o(n^{2.376})$, since $\log n = o(n^{\epsilon})$ for all $\epsilon > 0$. In order to achieve this time bound, we use several different techniques, one of which is matrix multiplication to make parts of the input graph into cliques. Our algorithm runs for $O(\log n)$ iterations, and at each iteration the total work is bounded by the time needed for matrix multiplication. In order to achieve $O(\log n)$ iterations, we show how to recursively divide the problem into independent subproblems of a constant factor smaller size using a specialized search technique. In order to bound the amount of work at each iteration by $O(n^{\alpha})$, we store and work on the complement graphs for each subproblem, in which case the subproblems do not overlap in any (non)edges. In addition, we use both the minimal separators and the potential maximal cliques of the input graph, combining the results of [5], [10], and [15].

Independent of our work, a very recent and thus yet unpublished result of Kratsch and Spinrad [12] uses matrix multiplication to give a new implementation of the minimal triangulation algorithm Lex M from 1976 [17]. Based on the matrix multiplication algorithm of [6] their presented time complexity is $O(n^{2.688})$. Other than the use of matrix multiplication, their approach

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: {pinar, telle, yngvev}@ii.uib.no.

is totally different from ours. Kratsch and Spinrad used matrix multiplication for similar problems in their SODA 2003 paper [11].

After the next section which contains some basic definitions, we give the main structure of our algorithm in Section 3, followed by the important subroutine for partitioning into balanced subproblems in Section 4, before tying these parts together in the last section.

2 Background and notation

We consider simple undirected and connected graphs $G = (V, E)$ with $n = |V|$ and $m = |E|$. When G is given, we will use $V(G)$ and $E(G)$ to denote the vertices and edges of G respectively. For a set $A \subseteq V$, $G(A)$ denotes the subgraph of G induced by the vertices in A. A is called a *clique* if $G(A)$ is complete. The process of adding edges to G between the vertices of $A \subseteq V$ so that A becomes a clique in the resulting graph is called *saturating* A. The *neighborhood* of a vertex v in G is $N_G(v) = \{u \mid uv \in E\}$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. Similarly, for a set $A \subseteq V$, $N_G(A) = \cup_{v \in A} N_G(v) \setminus A$, and $N_G[A] = N_G(A) \cup A$. $|N_G(v)|$ is the *degree* of v. When graph G is clear from the context, we will omit subscript G.

A vertex set $S \subset V$ is a *separator* if $G(V \setminus S)$ is disconnected. Given two vertices u and v, S is a *u, v-separator* if u and v belong to different connected components of $G(V \setminus S)$, and S is then said to *separate* u and v. Two separators S and T are said to be *crossing* if S is a u, v-separator for a pair of vertices $u, v \in T$, in which case T is an x, y-separator for a pair of vertices $x, y \in S$ [10, 15]. A u, v-separator S is *minimal* if no proper subset of S separates u and v. In general, S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u, v-separator. It can be easily verified that S is a minimal separator if and only if $G(V \setminus S)$ has two distinct connected components C_1 and C_2 such that $N_G(C_1) = N_G(C_2) = S$. In this case, C_1 and C_2 are called *full components*, and S is a minimal u, v-separator for *every* pair of vertices $u \in C_1$ and $v \in C_2$.

A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. A graph is *chordal*, or equivalently *triangulated*, if it contains no induced chordless cycle of length ≥ 4. A graph $G' = (V, E \cup F)$ is called a *triangulation* of $G = (V, E)$ if G' is chordal. The edges in F are called *fill edges*. G' is a *minimal triangulation* if $(V, E \cup F')$ is non-chordal for every proper subset F' of F. It was shown in [17] that a triangulation G' is minimal if and only if every fill edge is the unique chord of a 4-cycle in G'. Another characterization of minimal triangulations which is central to our results is that G' is a minimal triangulation of G if and only if G' is the result of saturating a maximal set of pairwise non-crossing minimal separators of G [15].

By the results of Kloks, Kratsch, and Spinrad [10], and Parra and Scheffler [15], it can be shown that the following recursive procedure creates a minimal triangulation of G: Take any connected vertex subset K and let $A = N[K]$, compute the connected components $C_1, ..., C_k$ of $G(V \setminus A)$, saturate each set $N(C_i)$ for $1 \leq i \leq k$ and call the resulting graph G', then compute a minimal triangulation of each subgraph $G'(N[C_i]), 1 \leq i \leq k$, and of $G'(A)$ independently. The key to understand this is to note that the saturated sets $N(C_i)$ are non-crossing minimal separators of G and G'. Thus the problem decomposes into independent subproblems overlapping only at the saturated minimal separators, and we can continue recursively on each subproblem that is not complete. This procedure basically defines the main structure of our algorithm, and we will prove its correctness formally in the next section.

An extension of these results, which we also use in our algorithm, was presented by Bouchitté and Todinca in [5]. There, a *potential maximal clique (pmc)* of G is defined to be a maximal clique in some minimal triangulation of G. If A is a pmc, then it is shown in [5] that whole A will automatically be saturated in the above recursive procedure instead of appearing as a subproblem, and that this modified procedure indeed characterizes minimal triangulations. In this case A is not necessarily $N[K]$ for a connected set K.

3 The new algorithm and the data structures

Observe that the total work for saturating all sets $N(C_i), 1 \leq i \leq k$, in the above recursive procedure requires $O(n^3)$ time if it is done straightforwardly, as these sets might overlap heavily and contain $O(n)$ vertices each. With help of matrix multiplication, this total time can be reduced to $O(n^\alpha)$. We construct the following matrix $M = M_{G,A}$: for each vertex v there is a row in M, for each connected component C of $G(V \setminus A)$ there is a column in M, and entry $M(v, C) = 1$ if $v \in N(C)$. All other entries are zero. Now we perform the multiplication MM^T, and in the resulting symmetric matrix, entry $(u, v) = (v, u)$ is nonzero if and only if u and v both belong to a common set $N(C)$ for some C. Thus MM^T is the adjacency matrix of a graph in which each $N(C)$ is a clique. The use of matrix multiplication for this purpose was first mentioned in [11].

Once MM^T is computed, the edges indicated by its nonzero entries can be added to G, resulting in the partially filled graph G', and the subproblems $G'(N[C_i]), 1 \leq i \leq k$, and $G'(A)$ can be extracted. Now for each subproblem this process can be repeated recursively. However, it is important that we do not perform a matrix multiplication for each subproblem in the further process, but create only *one* matrix and perform a single matrix multiplication for all subproblems of each level in the recursion tree. Thus in the resulting multiplied matrix, (u, v) is nonzero if and only if there is a connected component C of one of the subproblems of this level such that $u, v \in N_{G'}(C)$. For this reason, we cannot actually use recursion, and we have to keep track of all subproblems belonging to the same level. We do this by two queues Q_1 and Q_2 which will hold all subproblems for the current and next level respectively. Only those new subproblems that are not cliques in the partially filled graph should survive to the next iteration. For a new subproblem on vertex set $N[C_i]$ appearing from a connected component C_i after removing A we check this before the saturation, as we already know that the saturation will make $N(C_i)$ into a clique and not add any other edges to the graph induced by $N[C_i]$. However, for the subproblem on vertex set A itself we must wait until after the saturation before checking whether A now induces a clique, and for that reason we store the vertex sets A temporarily in a third queue Q_3.

Our algorithm, which we call FMT - Fast Minimal Triangulation - is given in Figure 1. The process of computing a good vertex set A is the most complicated part of this algorithm, and this part will be explained in the next section when we give the details of Algorithm Partition that returns such a set A. For the time being, and for the correctness of Algorithm FMT it is important and sufficient to note that Algorithm Partition returns a set A, where either $A = N[K]$ for some connected vertex set K, or A is a pmc.

LEMMA 3.1. *Algorithm FMT computes a minimal triangulation of the input graph, as long as the Partition(H) subroutine returns a set $A \subset V(H)$ where either $A = N[K]$ for some connected vertex set K or A is a pmc.*

Proof. Let $G = (V, E)$ be the input graph and let K be a set of vertices such that $G(K)$ is connected. It is shown in [1] that the set of minimal separators of G that are subsets of $N(K)$ is exactly the set $\{N(C) \mid C$ is a connected component of $G(V \setminus N[K])\}$. In [5] it is shown that if P is a pmc then the set of minimal separators that are contained in P is exactly the set $\{N(C) \mid C$ is a connected component of $G(V \setminus P)\}$.

Since A is always chosen so that either $A = N[K]$ for a connected set K, or A is a pmc (this will be proved in Section 4), then it follows that all sets that are saturated at the first iteration of Algorithm FMT are minimal separators of G. We will now argue that these minimal separators are non-crossing. Assume on the contrary that two crossing separators $S = N(C_1)$ and $T = N(C_2)$ are saturated at the first iteration, where C_1 and C_2 are two distinct connected components of $G(V \setminus A)$. Thus there are two vertices $u, v \in T$ with $u, v \notin S$ such that S is a minimal u, v-separator in G. Since $u, v \in T = N(C_2)$, and S does not contain any vertex of C_2, the removal of S cannot separate u and v as there is a path between u and v through vertices of C_2. This contradicts the assumption that S is a u, v-separator, and thus we can conclude that the minimal separators saturated at the first step are all pairwise non-crossing. It is important to observe that once these separators are saturated, all minimal separators of G that cross any of these will disappear, as the saturated sets do not contain pairs of vertices that are separable. At each iteration, any minimal separator of G' is a minimal separator of G [15]. Thus the minimal separators that we discover at each iteration will not cross the minimal separators discovered and saturated at previous iterations.

At each new iteration, the above argument can be applied to each subgraph H, and thus we compute a set of non-crossing minimal separators of each subgraph H at each iteration. We have already argued that these cannot cross any of the saturated minimal separators of previous iterations. We must also argue that no minimal separator of a subgraph of an iteration crosses a minimal separator of another subgraph of the same iteration. But this is straightforward as these subgraphs only intersect at cliques, and thus their sets of minimal separators are disjoint.

So, our algorithm computes and saturates a set of non-crossing minimal separators at each iteration. Since we continue this process until all minimal separators of G' are saturated, by the results of [10] and [15], we create a minimal triangulation.

If we consider merely correctness, any set A that fulfills the requirements can be chosen arbitrarily; for example $A = N[u]$ for a single vertex u, as in [2]. In order to achieve the desired time complexity, we will devote the next section to describing how to carefully choose a vertex subset A in each subproblem so that the number of iterations of the **repeat**-loop becomes $O(\log n)$.

In this section, we will argue that each iteration of the algorithm can be carried out in $O(n^\alpha)$ time. We start with the following lemma, which will give us the desired bound for the matrix multiplication step.

LEMMA 3.2. *At each iteration of Algorithm FMT, the number of columns in matrix M is less than n.*

Algorithm FMT - Fast Minimal Triangulation
Input: An arbitrary non-complete graph $G = (V, E)$.
Output: A minimal triangulation G' of G.

Let Q_1, Q_2 and Q_3 be empty queues; Insert G into Q_1;
$G' = G$;
repeat
    Construct a zero matrix M with a row for each vertex in V (columns are added later);
    while Q_1 is nonempty **do**
        Pop a graph $H = (U, D)$ from Q_1;
        Call **Algorithm Partition**(H) which returns a vertex subset $A \subset U$;
        Push vertex set A onto Q_3;
        for each connected component C of $H(U \setminus A)$ **do**
            Add a column in M such that $M(v, C) = 1$ for all vertices $v \in N_H(C)$;
            if $\exists$ non-edge uv in $H(N_H[C])$ with $u \in C$ **then** Push $H(N_H[C])$ onto Q_2;
        end-for
    end-while
    Compute MM^T;
    Add to G' the edges indicated by the nonzero elements of MM^T;
    while Q_3 is nonempty **do**
        Pop a vertex set A from Q_3;
        if $G'(A)$ is not complete **then** Push $G'(A)$ onto Q_2;
    end-while
    Swap names of Q_1 and Q_2;
until Q_1 is empty

Figure 1: Algorithm FMT : Fast Minimal Triangulation.

Proof. The sequence of iterations of the algorithm give rise to an iterative refinement of a tree-decomposition of the graph G', a property first shown for the LB-treedec algorithm discussed in [8]. Simplifying the standard notation, we say that a *tree-decomposition* T_i of a graph G is a collection of *bags*, subsets of the vertex set of G, arranged as nodes of a tree such that the bags containing any given vertex induce a connected subtree, and such that every pair of adjacent vertices of G is contained in some bag (see *e.g.* page 549 of [19] for the standard definition.) At the first iteration we have the trivial tree-decomposition T_1 with all vertices of G' in a single bag, until the last iteration p where the tree-decomposition T_p is in fact a clique tree of the now chordal graph G', with each bag inducing a unique maximal clique. We prove this by showing the following:

Loop invariant: At the start of iteration s we have a tree-decomposition T_s of the current partially filled graph G' whose bags consist of some vertex subsets inducing cliques, which are the vertices of subproblems inducing cliques as discovered so far by our algorithm, and where remaining bags are the vertex sets of subproblems in Q_1. The intersection of two neighboring bags in T_s is a saturated minimal separator of G' and thus induces a clique. T_s is non-redundant, meaning that if A, B are bags of T_s then we do not have $A \subseteq B$.

The invariant is clearly true for the trivial tree-decomposition T_1 with a single bag. Let vertex set U be a bag of T_s appearing as subproblem $H = (U, D)$ in Q_1. The algorithm proceeds to find $A \subset U$ and produces new vertex subsets $A, N[C_1], N[C_2], ..., N[C_k]$ where each C_i is a component of $G'(U \setminus A)$. The node of bag U in T_s is in T_{s+1} split into a k-star with center-bag A and leaf-bags $N[C_1], N[C_2], ..., N[C_k]$. Since A is a pmc or $A = N[K]$, it follows that this star is a tree-decomposition of $G'(U)$ which is non-redundant. The node of a neighboring bag X of U in the tree of T_s will also be split into a star, unless X induces a clique in which case it remains a single node, i.e. a trivial star. These two stars appearing from adjacent nodes in T_s will be joined in T_{s+1} by an edge between two bags U' and X' that each contain $U \cap X$. Such a bag must exist in each star since $U \cap X$ already induced a clique.

The tree-decomposition T_{s+1} is constructed by applying the construction above to each bag, and to adjacent pairs of bags, of T_s. After newly found minimal separators in G' have been saturated then T_{s+1} will be a tree-decomposition of G', as is easily checked. It remains to show that T_{s+1} is non-redundant. We do this by showing that none of the new vertex subsets $A, N[C_1], N[C_2], ..., N[C_k]$ are contained in $U \cap X$. The crucial fact is that each vertex in $U \cap X$ has a neighbor in $U \setminus X$, since $U \setminus X$ was a component of the minimal separator $U \cap X$. If A was chosen as $A = N[K]$ then even if $K \subseteq U \cap X$ we would therefore not have $A \subseteq U \cap X$. Likewise, we could have some component C_i of $G'(U) \setminus A$ with $C_i \subseteq U \cap X$, but we would never have $N[C_i] \subseteq U \cap X$. If A instead was chosen as a pmc then we cannot have $A \subseteq U \cap X$, as $U \cap X$ was a minimal separator and a maximal clique cannot be part of a minimal separator. Thus, T_{s+1} is non-redundant. Since any bag of T_{s+1} that does not induce a clique is put back onto Q_1 before the next iteration we have established the loop invariant.

Note that each column added to matrix M in the algorithm gives rise to a unique bag of T_{s+1}. Since the number of bags in the final tree-decomposition T_p is at most n, one for each maximal clique in a chordal graph, and the number of bags in trees $T_1, ..., T_p$ is strictly increasing, we have proved the lemma.

Consequently, the matrix multiplication step requires $O(n^{\alpha})$. In order to be able to bound the time for the rest of the operations of each iteration by $O(n^{\alpha})$, we will store and work on the *non-edges*, i.e., the edges of the complement graph for each subproblem. Note that subproblems can overlap both in vertices and in edges, which makes it difficult to bound the sum of their sizes for the desired time analysis. A non-edge uv is discarded when it becomes saturated or when vertices u and v are separated, and if it is not discarded it only appears in a single subproblem in the next iteration. Hence subproblems overlap only in cliques, so if we work on the complement of these subgraphs, then they actually do not overlap in any edges at all!

For each subgraph $H = (U, D)$ in Q_1, let $\bar{E}(H) = \binom{U}{2} \setminus D$ be the set of non-edges of H. Our data structure for each subproblem H is the adjacency list of $\bar{H} = (U, \bar{E}(H))$, where we also store the degree of each vertex in $\bar{H}$. It is an easy exercise to show that all linear time operations that we need to do for H, like computing the connected components and neighborhoods, can be done using only $\bar{H}$ in time $O(|\bar{E}(H)| + |V(H)|)$.

An interesting point is also that, when complement graphs are used, matrix multiplication is not necessary to saturate $N_H(C)$ of each subproblem $N_H[C]$, however it is still necessary in order to saturate the subsets of A that become cliques. In the implementation of our algorithm, for each subproblem $H(N_H[C])$, we push the complement graph consisting of all non-edges of $H(N_H[C])$ with at least one endpoint in C onto Q_2. We do this only if such a non-edge of H exists. Since these complement graphs consist of non-edges of $H(C)$ and non-edges of $H(N_H[C])$ between C and $N_H(C)$, all such subproblems can be computed in a total time of $O(|\bar{E}(H)| + |V(H)|)$ for H. Since we omit all non-edges between vertices belonging to $N_H(C)$, this actually corresponds to saturating $N_H(C)$ automatically.

After the matrix multiplication step, we look up in MM^T every edge of the complement of G' to check whether or not this non-edge should survive or should be deleted because it has now become a fill edge of G'. Since subproblems do not overlap in any non-edges checking whether or not $G'(A)$ is now complete can be done in a total of $O(n^2)$ time for all vertex subsets A in Q_3.

Thus, for the implementation of our algorithm, we compute $\bar{G}$ at the beginning, and use the complement graphs throughout the algorithm. This way, all operations described within an iteration can be completed within $O(n^{\alpha})$ time. For clearness, we will give the algorithms on the actual graphs and not on complement graphs, expecting the reader to think in terms of complement graphs while considering time complexity issues. We will use *edges of* $\bar{H}$ and *non-edges of* H interchangeably.

With the given data structures and explanations, it should be clear that all operations during one iteration, outside of Algorithm Partition, can be performed in $O(n^{\alpha})$ time. In order to get our algorithm to terminate within $O(\log n)$ iterations, we will split each graph that we pop from Q_1 into small enough subproblems.

Let q be a sufficiently large constant. In the next section, while proving the total time complexity, we will show that $q = 5$ suffices. For each subproblem H, if we manage to compute a vertex subset A so that the subproblems that are children of H do not contain more than $\frac{q-1}{q}|\bar{E}(H)|$ non-edges, then our algorithm will clearly stop within $\log_{\frac{q}{q-1}} n^2 = O(\log n)$ steps. In the next section, through Algorithm Partition, we show how to compute such vertex subsets for all subproblems in a total of $O(n^2)$ time for each iteration. We present the algorithm parameterized by q rather than fixing q to 5, to ease the task of an implementation that can experimentally find the best value $q \geq 5$.

4 Efficient Partition into balanced subproblems

In this section we will show how to compute vertex subsets A for each subproblem in order to achieve an even partitioning into subproblems. The algorithm that we present for doing this will have running time $O(|\bar{E}(H)| + |V(H)|)$ on each input subgraph H.

The computation of vertex subset A for each subgraph $H = (U, D)$ is done by Algorithm Partition which is given in Figure 2. The main idea behind this algorithm is to examine every vertex of H, and place these vertices into connected components that result from removing some set P of vertices from H as long as these connected components do not become too large with respect to the number of non-edges. The vertices that cannot be placed into any small enough connected component constitute exactly the set P whose removal from H results in small connected components. This way, we compute a vertex set P such that all connected components C of $H(U \setminus P)$ have the nice property that $H(N_H[C])$ contains less than a constant factor of the non-edges of H. However, after P is computed, we cannot bound the number of non-edges that will belong to $G'(P)$ after the saturation, we do not know whether $P = N_H[K]$ for a connected vertex set K as required, or whether P is a potential maximal clique. Thus we cannot simply use P as our desired set A. The set A is instead obtained using information gained through the computation of P, and we prove in Theorem 4.2 that it fulfills the requirements that were used to prove the correctness of Algorithm FMT, and that the resulting subproblems all have at most $\frac{q-1}{q}|\bar{E}(H)|$ non-edges.

During Algorithm Partition, the vertices that we are able to place into small enough connected components are marked as **c**-vertices. The remaining vertices (which constitute P) are of two types: **p**-vertices have neighbors in a connected component of $H(U \setminus P)$, whereas **s**-vertices do not. For each connected component C of $H(U \setminus P)$ we want to ensure that the number $|\bar{E}(H(N_H[C]))|$, i.e. the number of non-edges with both endpoints in $N_H[C]$, is less than some fraction of $|\bar{E}(H)|$. The obstacle is that we cannot compute this number straightforwardly for all connected components of $H(U \setminus P)$ in the given time, since the non-edges between vertices in $P \cap N_H[C]$ could be contained in too many such computations. However, we are able to give upper and lower bounds on $|\bar{E}(H(N_H[C]))|$ by summing the degrees in $\bar{H}$ of vertices in each $N_H[C]$, which we compute in the following roundabout manner in order to stay within the time limits. Define $\mathcal{E}_{\bar{H}}(S)$ to be the sum of degrees in $\bar{H}$ of vertices in $S \subseteq U = V(H)$. Since sum of degrees is equal to twice the number of edges, we have $\mathcal{E}_{\bar{H}}(S) = 2|\bar{E}(H)| - \mathcal{E}_{\bar{H}}(U \setminus S)$. The quantity $\mathcal{E}_{\bar{H}}(U \setminus N_H[C])$ we indeed do have the time to compute, as we will explain in the proof of Lemma 4.1.

$$\mathcal{E}_{\bar{H}}(U \setminus N_H[C]) = \sum_{v \in U \setminus N[C]} |N_{\bar{H}}(v)|$$

When checking whether $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}]) \geq \frac{2}{q}|\bar{E}(H)|$ in Algorithm Partition we are indirectly checking whether $|\bar{E}(N_H[C_k \cup \{v\}])| \leq \frac{q-1}{q}|\bar{E}(H)|$, which is what we indeed want to know. The discussion in the proof of Lemma 4.2 explains this connection. The value $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}])$ can be computed in $O(|N_{\bar{H}}(v)|)$ time for each vertex v in U, as we show in the proof of the following lemma.

LEMMA 4.1. *Running Algorithm Partition on all subgraphs H of a single iteration of Algorithm FMT requires a total of $O(n^2)$ time.*

Proof. First we prove that the running time of Algorithm Partition on input subgraph H is $O(|\bar{E}(H)| + |V(H)|)$, and then we will argue for the overall time bound at the end. Note that, as explained in the previous section, also for Algorithm Partition we will work on the complement graph $\bar{H}$ for an efficient implementation. Observe that between a connected component C and $U \setminus N_H[C]$, we have a complete bipartite graph in $\bar{H}$. These non-edges will be used as an argument to obtain the desired time bound.

The pseudocode of Algorithm Partition is presented in two bulks. Let us call the first bulk "defining P", and the second bulk "defining A".

The first operation in the "defining P" part is to unmark every vertex in H. The value $\mathcal{E}_{\bar{H}}(U \setminus N_H[u])$ for a single vertex u is computed straightforwardly by summing the degrees in the complement graph of all vertices in $U \setminus N_H[u] = N_{\bar{H}}(u)$, which is an $O(|N_{\bar{H}}(u)|)$ operation.

When a component C_k is created from a first vertex u we label every vertex $w \in N_{\bar{H}}(u)$ with the value $nk + |C_k| = nk + 1$. By labeling the vertices in this way, we assign a unique value to every vertex set that constitutes a component during the algorithm and ensure that only vertices in $U \setminus N_H[C_k]$ can have the label $nk + |C_k|$. The value $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}])$ can now be computed in $O(|N_{\bar{H}}(v)|)$ time, since the set of vertices in $N_{\bar{H}}(v)$ which are labeled $nk + |C_k|$ is exactly the set $U \setminus N[C_k \cup \{v\}]$. If v is going to be added to C_k then this increases the size of C_k by one and may affect the set $N[C_k]$. We update the labels of the vertices in $U \setminus N[C_k \cup \{v\}]$, by adding 1 to the label of every vertex in $N_{\bar{H}}(v)$ labeled with $nk + |C_k|$, and then add v to C_k. This requires $O(|N_{\bar{H}}(v)|)$ time for each vertex and $O(|\bar{E}(H)| + |V(H)|)$ in total for the "defining P" part, since every vertex is considered once and marked as a **p**, **c** or **s**-vertex. The **s**-vertices may be reconsidered once, and changed to **p**-vertices, but this not affect the time complexity.

The "defining A" part consists of an if-else statement with 4 cases. In the first case we can do the

Algorithm Partition
Input: A graph $H = (U, D)$ (a subproblem popped from Q_1).
Output: A subset $A \subset U$ such that either $A = N[K]$ for some connected $H(K)$ or A is a pmc of H (and G').

Unmark all vertices of H;
$k = 1$;
while $\exists$ unmarked vertex u **do**
  if $\mathcal{E}_{\bar{H}}(U \setminus N_H[u]) < \frac{2}{q}|\bar{E}(H)|$ **then**
    Mark u as an **s**-vertex (stop vertex);
  else
    $C_k = \{u\}$;
    Mark u as a **c**-vertex (component vertex);
    while $\exists\ v \in N_H(C_k)$ which is unmarked or marked as an **s**-vertex **do**
      if $\mathcal{E}_{\bar{H}}(U \setminus N_H[C_k \cup \{v\}]) \geq \frac{2}{q}|\bar{E}(H)|$ **then**
        $C_k = C_k \cup \{v\}$;
        Mark v as a **c**-vertex (component vertex);
      else
        Mark v as a **p**-vertex (pmc vertex);
        Associate v with C_k;
      end-if
    end-while
    $k = k + 1$;
  end-if
end-while
$P =$ the set of all **p**-vertices and **s**-vertices;

if $H(U \setminus P)$ has a full component C **then**
  $A = N_H[C]$;
else if there exist two non-adjacent vertices u, v such that u is an **s**-vertex and v is an **s**-vertex or a **p**-vertex **then**
  $A = N_H[u]$;
else if there exist two non-adjacent **p**-vertices u and v, where u is associated with C_i and v is associated with C_j and $u \notin N_H(C_j)$ and $v \notin N_H(C_i)$ **then**
  $A = N_H[C_i \cup \{u\}]$;
else
  $A = P$;
end-if

Figure 2: Algorithm Partition.

required test by simply finding the largest neighborhood of a component and checking if its size is $|P|$. Without increasing the time complexity of the "defining P" part we can store the values $|C|$ and $|U \setminus N_H[C]|$ for each component C of $H(U \setminus P)$. Thus $|N_H(C)| = |U| - (|C| + |U \setminus N_H[C]|)$.

In the second case, we check every non-edge in $H(P)$, which is also an $O(|\bar{E}(H)| + |V(H)|)$ operation.

In the third case we will mark non-edges and components, as follows. For each **p**-vertex u, and then for each component C of $H(U \setminus P)$ where $C \subseteq N_{\bar{H}}(u)$, we mark C with the label u. Then for every **p**-vertex $v \in N_{\bar{H}}(u)$, where v is associated to a component labeled u, we add u to the label of non-edge uv. We go through vertices in $N_{\bar{H}}(u)$, check which components they belong to, add up these numbers for each component, and check if it matches the total size of the component. This takes $O(|N_{\bar{H}}(u)|)$ time for each **p**-vertex. The third case will now exist if and only if there is a non-edge uv marked by both u and v. Thus the total time for this case is $O(|\bar{E}(H)| + |V(H)|)$ for each subgraph H.

The fourth case requires constant time, and thus the total running time of Algorithm Partition on input subgraph H is $O(|\bar{E}(H)| + |V(H)|)$.

The operations that require $O(|\bar{E}(H)| + |V(H)|)$ on each subgraph H add up to $O(n^2)$ for all subgraphs of the same iteration of FMT, since they do not overlap in non-edges, and there are at most $O(n)$ such graphs by Lemma 3.2. Thus the total time complexity for all subgraphs H at the same iteration is $O(n^2)$.

We now give upper and lower bounds on the number of non-edges in various subgraphs of H related to vertex set P.

LEMMA 4.2. *Let P be as computed by Algorithm Partition(H), and let $q \geq 3$ be a constant. Then each of the following is true:*

(i) $|\bar{E}(H(N_H[C]))| \leq \frac{q-1}{q}|\bar{E}(H)|$ for each connected component C of $H(U \setminus P)$.

(ii) $|\bar{E}(H(N_H[v]))| > \frac{q-2}{q}|\bar{E}(H)|$ for each **s***-vertex v.*

(iii) $|\bar{E}(H(N_H[C \cup \{v\}]))| > \frac{q-2}{q}|\bar{E}(H)|$ for each **p***-vertex v associated to C, where C is a connected component of $H(U \setminus P)$.*

Proof. (i) From Algorithm Partition we know that $\mathcal{E}_{\bar{H}}(U \setminus N_H[C]) \geq \frac{2}{q}|\bar{E}(H)|$ for each connected component C of $H(U \setminus P)$. Each non-edge uv outside of $H(N_H[C])$ contributes to the degree-sum $\mathcal{E}_{\bar{H}}(U \setminus N_H[C])$ by 1 if one of u or v is outside $N_H[C]$, and by 2 if both are outside. Thus there are at least $\frac{1}{q}|\bar{E}(H)|$ non-edges outside $H(N_H[C])$ and consequently at most $\frac{q-1}{q}|\bar{E}(H)|$ non-edges inside $H(N_H[C])$. Hence, $|\bar{E}(H(N_H[C]))| \leq \frac{q-1}{q}|\bar{E}(H)|$ for each connected component C of $H(U \setminus P)$, which completes the proof of *(i)*.

(ii) - (iii) From Algorithm Partition we know that $\mathcal{E}_{\bar{H}}(U \setminus N_H[v]) < \frac{2}{q}|\bar{E}(H)|$ for each **s**-vertex v, and $\mathcal{E}_{\bar{H}}(U \setminus N_H[C \cup \{u\}]) < \frac{2}{q}|\bar{E}(H)|$ for each **p**-vertex u associated to C. It follows by the same argument as case *(i)* that $|\bar{E}(H(N_H[v]))| > \frac{q-2}{q}|\bar{E}(H)|$ and $|\bar{E}(H(N_H[C \cup \{u\}]))| > \frac{q-2}{q}|\bar{E}(H)|$. This completes the proof of *(ii)* and *(iii)*.

The main result of this section, namely that the vertex set A returned by Partition results in subproblems of size bounded by a constant factor of the number of non-edges, is given in Theorem 4.2. For the proof of this theorem, we use the following result by Bouchitté and Todinca.

THEOREM 4.1. (Bouchitté and Todinca [5]) *Given $H = (U, D)$, let $P \subseteq U$ be any set of vertices, and let $C_1, C_2, ..., C_k$ be the connected components of $H(U \setminus P)$. P is a pmc of H if and only if*

1. $H(U \setminus P)$ has no full component, and

2. P is a clique when every $N(C_i)$ is saturated for $1 \leq i \leq k$.

We are now ready to prove the main theorem of this section.

THEOREM 4.2. *Let A be the vertex set returned by Algorithm Partition on input $H = (U, D)$. Then both of the following are true, where G' is as defined in Algorithm FMT:*

(i) A is a proper subset of U such that either $A = N_H[K]$ where $K \subset U$ and $H(K)$ is connected, or A is a pmc of H.

(ii) Both the number of non-edges in $G'(A)$ and the number of non-edges in $G'(N_H[C])$ for each connected component C of $H(U \setminus A)$ are at most $\frac{q-1}{q}|\bar{E}(H)|$, for any constant $q \geq 5$.

Proof. We will examine each of the 4 cases of the if-else statement in the "defining A" part of Algorithm Partition. We omit the subscript H in $N_H(C)$ and $N_H[C]$ to increase readability. The reader should keep in mind that throughout this proof we regard neighborhoods in H (and not in $\bar{H}$).

Case 1. $H(U \setminus P)$ has a full component C, i.e., $P = N(C)$.

This implies in particular that no vertices could have been marked as **s**-vertices. By Lemma 4.2 we know that the number of non-edges in $H(N[C_i])$ is less than $\frac{q-1}{q}|\bar{E}(H)|$ for each connected component C_i of $H(U \setminus P)$, in particular for C. In this case, Algorithm Partition gives $A = N[C]$, and thus $P \subset A$. C is a connected set since it was computed by adding new members from its neighborhood, and so *(i)* is satisfied. Observe that the connected components C_i of $H(U \setminus A)$ are exactly the connected components C_i of $H(U \setminus P)$, except C. It follows that the number of non-edges in $H(A) = H(N[C_i])$ and in $H(N[C_j])$ for each connected component C_j of $H(U \setminus A)$ is less than $\frac{q-1}{q}|\bar{E}(H)|$, already before the minimal separators are saturated. After the saturation, this number cannot increase but only decrease. We need to choose q so that the fractions $\frac{q-1}{q}$, and $\frac{2}{q}$ as used in the algorithm, are less than 1, and note that $q \geq 3$ suffices for this so *(ii)* is satisfied.

Case 2. There exist two vertices u, v, such that $uv \notin E(H)$, u is marked as an **s**-vertex, and v is marked as an **s**-vertex or a **p**-vertex.

We give the proof in two parts: the subcase where both u, v are **s**-vertices, and the subcase where u is an **s**-vertex and v a **p**-vertex. The arguments for the two subcases are very similar, and note that they are also very similar to the next *Case 3* where both u, v are **p**-vertices.

Assume both u and v are marked as **s**-vertices. By Lemma 4.2, $|\bar{E}(H(N[u]))| > \frac{q-2}{q}|\bar{E}(H)|$ and $|\bar{E}(H(N[v]))| > \frac{q-2}{q}|\bar{E}(H)|$, and thus for their common part we have $|\bar{E}(H(N(u) \cap N(v)))| = |\bar{E}(H(N[u] \cap N[v]))| > \frac{q-4}{q}|\bar{E}(H)|$, where the first equality holds since $u \notin N[v]$. The algorithm gives $A = N[u]$ in this case, satisfying *(i)*, which means that v will belong to a component C of $H(U \setminus A)$ with $N(C) \subseteq A$ thus being a u, v-separator. Since any u, v-separator must contain $N(u) \cap N(v)$, it follows that $N(C) \subseteq A$ induces at least $\frac{q-4}{q}|\bar{E}(H)|$ non-edges. All these non-edges will be saturated and disappear from G'. Thus, there are at most $\frac{4}{q}|\bar{E}(H)|$ non-edges left that can appear in subproblems $G'(A)$ or $H(N[C_i])$ for a component C_i of $H(U \setminus A)$, thereby satisfying also *(ii)*.

Assume u is marked as an **s**-vertex, v is marked as a **p**-vertex and let j be the index such that v is associated to C_j. We know that such a C_j exists since v is marked as a **p**-vertex. An important observation now is that $u \notin N[C_j \cup \{v\}]$. Otherwise u would have been marked as a **p**-vertex or **c**-vertex during execution of the inner while-loop in Algorithm Partition during computation of C_j. By Lemma 4.2, $|\bar{E}(H(N[u]))| > \frac{q-2}{q}|\bar{E}(H)|$ and $|\bar{E}(H(N[C_j \cup \{v\}]))| > \frac{q-2}{q}|\bar{E}(H)|$, and thus for their common part we have $|\bar{E}(H(N(u) \cap N(C_j \cup \{v\})))| = |\bar{E}(H(N[u] \cap N[C_j \cup \{v\}]))| > \frac{q-4}{q}|\bar{E}(H)|$, where the first equality holds since $u \notin N[C_j \cup \{v\}]$, as we established above. The algorithm gives $A = N[u]$ in this case, satisfying *(i)*, which means that $C_j \cup \{v\}$ will be contained in a component C of $H(U \setminus A)$ with $N(C) \subseteq A$ thus separating u from $C_j \cup \{v\}$. Since any such separator must contain $N(u) \cap N(C_j \cup \{v\})$, it follows that $N(C) \subseteq A$ induces at least $\frac{q-4}{q}|\bar{E}(H)|$ non-edges. All these non-edges will be saturated and disappear from G'. Thus, there are at most $\frac{4}{q}|\bar{E}(H)|$ non-edges left that can appear in subproblems $G'(A)$ or $H(N[C_i])$ for a component C_i of $H(U \setminus A)$, thereby satisfying also *(ii)*.

Case 3. There exist two vertices u, v marked as **p**-vertices, such that $uv \notin E(H)$, u is associated to C_i, v is associated to C_j, $u \notin N(C_j)$ and $v \notin N(C_i)$.

The important observation now is that there are no edges between $C_i \cup \{u\}$ and $C_j \cup \{v\}$. By Lemma 4.2, $|\bar{E}(H(N[C_i \cup \{u\}]))| > \frac{q-2}{q}|\bar{E}(H)|$ and $|\bar{E}(H(N[C_j \cup \{v\}]))| > \frac{q-2}{q}|\bar{E}(H)|$, and thus for their common part we have $|\bar{E}(H(N(C_i \cup \{u\}) \cap N(C_j \cup \{v\})))| = |\bar{E}(H(N[C_i \cup \{u\}] \cap N[C_j \cup \{v\}]))| > \frac{q-4}{q}|\bar{E}(H)|$, where the first equality holds since there are no edges between $C_i \cup \{u\}$ and $C_j \cup \{v\}$. The algorithm gives $A = N[C_i \cup \{u\}]$ in this case, satisfying *(i)*, which means that $C_j \cup \{v\}$ will be contained in a component C of $H(U \setminus A)$ with $N(C) \subseteq A$ thus separating $C_i \cup \{u\}$ from $C_j \cup \{v\}$. Since any such separator must contain $N(C_i \cup \{u\}) \cap N(C_j \cup \{v\})$, it follows that $N(C) \subseteq A$ induces at least $\frac{q-4}{q}|\bar{E}(H)|$ non-edges. All these non-edges will be saturated and disappear from G'. Thus, there are at most $\frac{4}{q}|\bar{E}(H)|$ non-edges left that can appear in subproblems $G'(A)$ or $H(N[C_i])$ for a component C_i of $H(U \setminus A)$, thereby satisfying also *(ii)*.

Case 4. None of the above cases apply.

First we show that P is a pmc of H in this case. Due to Theorem 4.1, all we have to show is that if none of the Cases 1, 2, and 3 applies, then $H(U \setminus P)$ has no full component associated to P, and for every pair of non-adjacent vertices $u, v \in P$ there is a connected component C of $H(U \setminus P)$ such that $u, v \in N(C)$. Since Case 1 does not apply, we know that $H(U \setminus P)$ has no full components. Since Case 2 does not apply either, then the **s**-vertices altogether induce a clique and they all have edges to all **p**-vertices. So, since P consists only of **p** and **s** vertices, the only non-edges that are possible within P are those non-edges uv where both u and v are **p**-vertices. Since Case 3 does not apply either, then for any non-adjacent $u, v \in P$ if they are not associated to the same component then one of them must be in the neighborhood of the component that the other one is associated to. Thus P is a pmc of H, and *(i)* is satisfied since Algorithm Partition gives $A = P$ in this case. In this case, whole A is saturated in G', and thus $G'(A)$ has no non-edges. The remaining subproblems will each have at most $\frac{q-1}{q}|\bar{E}(H)|$ non-edges by Lemma 4.2, since the connected components of $H(U \setminus A)$ are the same as the connected components of $H(U \setminus P)$. This case depends on Cases 1, 2, and 3, and thus $q \geq 5$ is sufficient, and *(ii)* is satisfied.

5 The total $O(n^\alpha \log n)$ time complexity

THEOREM 5.1. *Algorithm FMT described in Section 3, using Algorithm Partition described in Section 4, computes a minimal triangulation of the input graph in $O(n^\alpha \log n)$ time.*

Proof. By Lemma 3.1 and Theorem 4.2*(i)*, Algorithm FMT computes a minimal triangulation. By Lemma 3.2, the matrix multiplication at each iteration of FMT requires $O(n^\alpha)$ time. By the discussion that follows Lemma 3.2 in Section 3, all other operations outside of Algorithm Partition can be performed in $O(n^2)$ time at each iteration of FMT. Using Lemma 4.1, we conclude that the total time required at each iteration of FMT is $O(n^\alpha)$, since $\alpha \geq 2$ for any matrix multiplication algorithm. By Theorem 4.2*(ii)*, the number of non-edges in each subproblem decreases by a constant factor for each iteration, and since subproblems in one iteration do not overlap in non-edges we can at most have $\log n^2 = O(\log n)$ iterations of FMT.

We have thus given the details of a new algorithm to compute minimal triangulations of arbitrary graphs in $O(n^\alpha \log n)$ time. If we use the matrix multiplication algorithm of Coppersmith and Winograd [6], then α is strictly less than 2.376, and thus the total running time of our algorithm becomes $o(n^{2.376})$. If we instead use the matrix multiplication algorithm of Strassen [18] which has a worse asymptotic time bound of $\Theta(n^{\log_2 7}) = o(n^{2.81})$ but is considered more practical due to large constants in [6], then our time bound becomes $O(n^{\log_2 7} \log n) = o(n^{2.81})$. Using Strassen's algorithm, the time bound claimed by Kratsch and Spinrad mentioned previously becomes $O(n^{2.91})$ [12]. In fact, our algorithm is asymptotically faster than theirs regardless of the matrix multiplication algorithm used.

References

[1] A. Berry, J-P. Bordat, and P. Heggernes. Recognizing weakly triangulated graphs by edge separability. *Nordic Journal of Computing*, 7:164–177, 2000.

[2] 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*. To appear.

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

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

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

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

[7] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In *Graph Theoretical Concepts in Computer Science - WG '97*, pages 132–143. Springer Verlag, 1997. LNCS 1335.

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

[9] D. Hudson, S. Nettles, and T. Warnow. Obtaining highly accurate topology estimates of evolutionary trees from very short sequences. In *Proceedings of RECOMB'99*, pages 198–207. 1999.

[10] 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.

[11] D. Kratsch and J. Spinrad. Between $O(nm)$ and $O(n^\alpha)$. In *Proceedings of the 14^{th} Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 709–716, 2003.

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

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

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

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

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

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

[18] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14:354–356, 1969.

[19] J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*. North Holland, 1990.