# A Practical Algorithm for Making Filled Graphs Minimal *

Jean R S Blair
U S Military Academy
West Point, NY, USA
*dj4693@trotter.usma.edu*

Pinar Heggernes    Jan Arne Telle
Department of Informatics
University of Bergen, Norway
*pinar@ii.uib.no*    *telle@ii.uib.no*

**Abstract**

For an arbitrary filled graph $G^+$ of a given original graph $G$, we consider the problem of removing fill edges from $G^+$ in order to obtain a graph $M$ that is both a minimal filled graph of $G$ and a subgraph of $G^+$. For $G^+$ with $f$ fill edges and $e$ original edges, we give a simple $O(f(e+f))$ algorithm which solves the problem and computes a corresponding minimal elimination ordering of $G$. We report on experiments with an implementation of our algorithm, where we test graphs $G$ corresponding to some real sparse matrix applications and apply well-known and widely used ordering heuristics to find $G^+$. Our findings show the amount of fill that is commonly removed by a minimalization for each of these heuristics, and also indicate that the runtime of our algorithm on these practical graphs is better than the presented worst-case bound.

## 1 Introduction

For any graph $G$ and an ordering $\alpha$ of its vertices, there is an associated set of *fill* edges that, when added to $G$, results in a chordal graph $(G;\alpha)$ called the *filled graph* (see Section 2 for all definitions.) The problem of finding orderings of the vertices that produce low fill has been studied by researchers in many areas of computer science, *e.g.* in the solution of sparse symmetric systems of linear equations [13, 16, 17, 18, 19], data-base management systems [1, 21], knowledge-based systems [9, 12], and computer vision [5]. The problem remains an important research topic.

In a central 1976 paper Rose, Tarjan, and Lueker [20] gave an algorithm which finds a *minimal* fill ordering of a graph $G$ in $O(ne)$ time, where $n$ and $e$ are respectively the number of vertices and edges in $G$. To date, this algorithm, which is called LEX-M, has the best known running time for finding a minimal filled graph of an arbitrary graph. An efficient parallel algorithm for the same problem is given by Dahlhaus and Karpinski in [7]. Yannakakis showed in [22] that finding the *minimum* fill for an arbitrary graph is $\mathcal{NP}$-hard. Several heuristics have been proposed for finding elimination orderings producing low fill. The two most famous and practically useful methods are called *minimum degree* and *nested dissection* (see [10] for a survey).

Given a graph $G$ and an arbitrary ordering $\alpha$ of its vertices, we consider the problem of finding a graph $M$ that is both a minimal chordal supergraph of $G$

*This research was supported in part by the Research Council of Norway, and was conducted while the first author was visiting the University of Bergen, Norway.

and a subgraph of the filled graph $(G;\alpha)$. We also find a related minimal ordering $\beta$ where $(G;\beta) = M$, so that $M$ is a minimal filled graph of $G$. Minimal orderings are desirable in practice since any perfect elimination ordering of the resulting filled graph, when applied to the original graph, produces the same filled graph, and hence the planned data storage scheme is not disturbed. This is the case, for example, in sparse matrix computations, where perfect elimination orderings of the filled graph, *e.g.* post orderings of the corresponding elimination tree (see [15]), are usually found to achieve better properties for further computations. In other words, if $\beta$ is a minimal ordering of a graph $G$ then for any perfect elimination ordering $\gamma$ of $(G;\beta)$ we have $(G;\gamma) = (G;\beta)$, and this property does not hold for elimination orderings in general. In particular, we show in Section 3 that if an ordering $\alpha$ is not minimal, then there always exists a perfect elimination ordering $\gamma$ of $(G;\alpha)$ such that $(G;\gamma)$ is a strict subgraph of $(G;\alpha)$.

The problem we consider is motivated by the following two facts: 1. Minimal orderings are not necessarily close to minimum in general, and lower fill is usually achieved by practical heuristic algorithms like minimum degree and nested dissection. 2. These famous heuristic algorithms usually produce non-minimal fill, and minimal fill is desirable in practice as explained above. Therefore, a suitable approach is to first apply a heuristic algorithm to find a non-minimal low fill ordering $\alpha$ and then run our algorithm to remove redundant fill until the remaining fill is minimal.

The main contributions of this paper are two-fold: First, we develop an $O(f(e+f))$ time algorithm that, given $G$ and $\alpha$, based on our Theorem 3.8 greedily considers fill edges for removal, in the reverse order to that in which they were introduced, and produces an ordering $\beta$ such that $(G;\beta)$ is minimal and is a subgraph of $(G;\alpha)$. Here, $f$ and $e$ are respectively the number of fill edges in $(G;\alpha)$ and the number of edges in $G$, *i.e.* $(G;\alpha)$ has $f+e$ edges total. Second, we have implemented our algorithm in FORTRAN90 and we report on experiments where we take our graphs $G$ from the Harwell-Boeing matrix collection and apply the minimum degree and nested dissection heuristics to find $\alpha$. Our findings indicate that minimum degree indeed finds a minimal fill in many cases, while for nested dissection we are more often able to remove a substantial number of fill edges with our algorithm. To our knowledge, this is the first time minimality properties of these famous heuristics are confirmed experimentally. Furthermore, the worst-case time complexity $O(f(e+f))$ of our algorithm depends on structural properties of the filled graph, and our tests indicate that this worst-case bound is usually not met.

A preliminary version [2] of this paper was presented at the Fifth Scandinavian Workshop on Algorithm Theory in 1996. Subsequently Dahlhaus presented in [6] an $O(ne)$ algorithm to solve the same problem. These two algorithms thus have the same worst-case asymptotic time complexity when the fill is linear in the number of vertices, *i.e.* when $f = \Theta(n)$, while $O(f(e+f))$ wins if $f = o(n)$ and $O(ne)$ wins if $f = \omega(n)$. Our tests indicate that there are many practical matrices where indeed $f = O(n)$.

The paper is organized as follows. Section 2 formally defines terms used in the paper. Section 3 characterizes redundant fill edges. The new algorithm and its proof of correctness are found in Section 4 with an analysis of the time complexity. Section 5 contains numerical results of our tests concerning the amount of redundant fill removed and the runtime of our implementation compared to that of minimum degree and nested dissection. The paper is concluded with some final remarks in Section 6.

## 2 Definitions and Notation

We start with some standard graph terminology. We consider undirected, simple graphs. For a graph $G$, the vertex and edge sets are denoted by respectively $V(G)$ and $E(G)$. $N_G(v)$ is the set of neighbors of $v$ in $G$. A *path* from $v_1$ to $v_k$ is a sequence of vertices $v_1, v_2, ..., v_k$ that are connected by the edges $v_1v_2, v_2v_3, ..., v_{k-1}v_k$. We also use $v_1 \to v_k$ to denote a path from $v_1$ to $v_k$. A *cycle* is a path whose first and last vertices are the same. An edge in $G$ is called a *chord* of a cycle if it joins two nonconsecutive vertices on the cycle. A graph is *chordal* if every cycle of length at least four has a chord. For a set $S$ of vertices in $G$, the subgraph of $G$ induced by $S$ is denoted by $G[S]$. The graph $G \setminus S$ is the graph $G[V(G) \setminus S]$. For a set $K$ of edges in $G$, the graph $G \setminus K$ is the result of removing the edges in $K$ from $G$. A *supergraph* of $G$ is a graph which contains $G$ as a subgraph.

For a graph $G$ with $|V(G)| = n$, an *elimination ordering* of $G$ is a bijection $\alpha : V(G) \leftrightarrow \{1, 2, ..., n\}$. For ease of presentation, we will also refer to $\alpha$ as a sequence, $\alpha = v_1, v_2, ..., v_n = \alpha^{-1}(1), \alpha^{-1}(2), ..., \alpha^{-1}(n)$. Thus $v_i$ denotes the vertex $v$ such that $\alpha(v) = i$.

Associated with $\alpha$ is a sequence of supergraphs of $G$, defined as follows. $G_0 = G$, and for $1 \leq i \leq n$, $G_i$ is the graph obtained by adding edges to $G_{i-1}$ so that all vertices in $N_{G_{i-1}}(v_i) \cap \{v_{i+1}, ...v_n\}$ are pairwise adjacent. This step is called the *elimination* of vertex $v_i$, and the whole process of obtaining $G_n$ from $G$ is called the *elimination process*. Note that vertices are not removed from the graph: $V(G) = V(G_i)$, $1 \leq i \leq n$. Usually, the elimination process and the graphs $G_i$ are defined such that vertex $v_i$ is removed after its elimination (thereby the term *elimination*). However, for our purposes and algorithm, the non-shrinking graphs $G_i$ defined here are more appropriate. The new edges added are called *fill edges*, and $F_i = E(G_i) \setminus E(G_{i-1})$ is the set of fill edges created by the elimination of $v_i$. We use $C_i$ to denote the resulting clique induced by $v_i$ and its higher numbered neighbors. The graph $G_n$ is the *filled graph* of $G$ for elimination ordering $\alpha$. We also use $(G; \alpha)$ to denote the filled graph $G_n$. All filled graphs are chordal.

An elimination ordering $\alpha$ on $G$ is *minimal* if the resulting $(G; \alpha)$ is a *minimal chordal* supergraph of $G$. In other words, no strict subgraph of $(G; \alpha)$ containing $G$ is chordal, equivalently, for no ordering $\gamma$ is $(G; \gamma)$ a strict subgraph of $(G; \alpha)$. Then $(G; \alpha)$ is also referred to as a *minimal filled graph* of $G$. An elimination ordering $\alpha$ on $G$ is *minimum* if no other elimination ordering $\beta$ on $G$ can produce a filled graph with fewer edges than $(G; \alpha)$. Minimum orderings are, of course, minimal.

A vertex is *simplicial* in a graph $G$ if its neighbors induce a clique. Note that $v_i$ is simplicial in $G_{i-1}[\{v_i, ..., v_n\}]$ if and only if $F_i = \emptyset$. An elimination ordering on $G$ is *perfect* if no fill edges are created, that is, if $F_i = \emptyset$ for $1 \leq i \leq n$. Chordal graphs are exactly the class of graphs that have perfect elimination orderings.

## 3 Greedy Removal of Redundant Fill Edges

In this section we develop some properties of fill edges related to the order of their introduction, with the goal of finding an algorithm to remove redundant fill edges. The following result from [20] gives another characterization of fill edges.

**Lemma 3.1** *[20] Let $G$ be a graph and $\alpha$ an elimination ordering of its vertices. Then $uv$ is an edge of $(G; \alpha)$ if and only if $uv \in E(G)$ or there exists a path*

$u = u_0, u_1, ..., u_{k+1} = v$ *in* $G$ *such that, for* $1 \leq i \leq k$*, we have* $\alpha(u_i) < \min\{\alpha(u), \alpha(v)\}$.

The same paper also gives the following alternative characterization of minimal elimination orderings.

**Theorem 3.2** *[20] Let* $G$ *be a graph and* $\alpha$ *an elimination ordering of its vertices. Then* $\alpha$ *is a minimal elimination ordering if and only if each fill edge is the unique chord of a 4-cycle in* $(G; \alpha)$.

We will need the following corollary.

**Corollary 3.3** *Let* $G$ *be a graph and* $\alpha = v_1, ..., v_n$ *be an elimination ordering of its vertices. If a fill edge* $uv \in F_i$*, created by elimination of* $v_i$*, is the unique chord of a 4-cycle in* $(G; \alpha)$ *then* $uv$ *is the unique chord of a 4-cycle* $u, v_i, v, x, u$ *in* $(G; \alpha)$ *with* $\alpha(x) > i, \alpha(u) > i$ *and* $\alpha(v) > i$.

**Proof.** Let $u, y, v, x, u$ be a 4-cycle of $(G; \alpha)$ in which $uv$ is the only chord. Since $uv$ is created by elimination of $v_i$ we must have $\alpha(u) > i$, $\alpha(v) > i$, $\alpha(y) \geq i$ and $\alpha(x) \geq i$. If either $y = v_i$ or $x = v_i$ we are done. If this is not the case, note that either $yv_i$ or $xv_i$ is not an edge of $(G; \alpha)$ since otherwise $xy$ would be a fill edge. As $u$ and $v$ are both neighbors of $v_i$, we can find a 4-cycle as stated in the corollary. □

Observe that Theorem 3.2 and Corollary 3.3 can be used to easily detect whether a given elimination ordering is minimal.

**Definition 3.4** *Let* $G^+$ *be a chordal supergraph of a graph* $G$*. A* candidate *edge is an edge* $uv \in E(G^+) \setminus E(G)$ *which is not the unique chord of any 4-cycle in* $G^+$.

Since removing a single edge from a chordal graph results in either a chordal graph or a graph with a chordless 4-cycle, it is clear that removing any candidate edge cannot destroy chordality. The following result is implicit in [20].

**Fact 3.5** *[20] A non-minimal chordal supergraph* $G^+$ *of a graph* $G$ *has at least one candidate edge. Any candidate edge can be removed without destroying chordality.*

Thus, if $(G; \alpha)$ is not minimal, then there exists another ordering $\gamma$ such that $(G; \gamma)$ has at least one edge less than $(G; \alpha)$. Furthermore, before we continue with the results that constitute the main basis of our algorithm, we show that $\gamma$ can always be chosen among perfect elimination orderings of $(G; \alpha)$. As mentioned in the introduction, this constitutes a further motivating factor for finding minimal orderings.

**Lemma 3.6** *For any non-minimal ordering* $\alpha$ *of a graph* $G$*, there is a perfect elimination ordering* $\gamma$ *of* $(G; \alpha)$ *such that* $(G; \gamma)$ *is a strict subgraph of* $(G; \alpha)$.

**Proof.** Let $uv$ be the last introduced fill edge that is a candidate edge in $(G; \alpha)$, with $\alpha = v_1, ..., v_n$. In other words, if $uv \in F_i$ then there are no candidate edges in $F_j$, where $i < j$. Let $v_i$ be the vertex whose elimination introduced the fill edge $uv$. Consider $v_i$ and its higher numbered neighbors in $(G; \alpha)$; these induce a clique $C_i$ as explained in the previous section. Since $uv$ is a candidate edge, every vertex $v_j$, with $j > i$, that is adjacent to both $u$ and $v$ must also be adjacent to $v_i$, and thus be a part of the clique $C_i$. Consequently, the only vertices whose elimination could lead to the creation of the fill edge $uv$ all belong to $C_i$. Otherwise $uv$ would have been created before the elimination of $v_i$. In [19] Rose shows that any clique can be eliminated last in a perfect elimination ordering of a chordal graph. Let $\gamma$ be a perfect elimination ordering of $(G;\alpha)$ where the vertices of $C_i$ are eliminated last. In addition let $u$ and $v$ be eliminated last among the vertices of $C_i$. Applying $\gamma$ on the original graph $G$ will not create the fill edge $uv$, and since it is a perfect elimination ordering of $(G;\alpha)$, it will not result in any fill that is not in $(G;\alpha)$. Thus, $(G;\gamma)$ has at least one edge less than $(G;\alpha)$ and is a strict subgraph. $\square$

Fact 3.5 gives an idea for finding a minimal filled graph by removing fill edges from a filled graph: repeatedly remove candidate edges until no candidate edges remain. Note that the proof of Lemma 3.6 provides an algorithm for doing this: Since at least one candidate edge is removed by the described perfect elimination ordering of the filled graph, the same procedure can be repeated for the resulting graph recursively until a minimal filled graph is reached. However, this is a time consuming approach, and Lemma 3.6 is merely meant as a theoretical motivation for finding minimal orderings.

Following Fact 3.5, we will describe a method where fill edges are examined for candidacy and removed if possible. Unfortunately, the set of candidate edges changes as edges are removed, with new candidate edges being introduced and old candidate edges ceasing to be candidates. Therefore, if we are to remove the candidate edges in an arbitrary order, each fill edge might have to be checked several times for candidacy. For a simple example, consider the path on 4 vertices $P_4$ with edges $ab, bc, cd$ and elimination order $b, c, d, a$ creating the fill $F_1 = \{ac\}$ and $F_2 = \{ad\}$. The fill edge $ac$ is not a candidate for removal but $ad$ is. However, after removing $ad$ the fill edge $ac$ becomes a candidate for removal.

This non-monotonicity property of the set of candidate edges seems to be the main obstacle in obtaining an efficient implementation of this algorithm. As we show below, we can partially avoid this by processing fill edges in the reverse fill order of that in which they were introduced. In particular, the following strategy will produce a minimal chordal supergraph:

**for** $i = n$ **downto** 1 **do**
    **while** there is a candidate edge in $F_i$ **do** remove it from the filled graph;

We first give a lemma and then a theorem that will provide the basis for the correctness of this strategy.

**Lemma 3.7** *Let $\alpha = v_1, ..., v_n$ be an elimination order of a graph $G$. Let $M_i$ be both a subgraph of $G_n$ and a minimal chordal supergraph of $G_{i-1}$. Then $M_i$ has a perfect elimination ordering $\gamma = w_1, w_2, ..., w_n$ with $(G;\gamma) = M_i$ and $v_k = w_k$ for $k = 1, ..., i-1$.*

**Proof.** Since both $G_{i-1}$ and $G_n$ are associated with $\alpha = v_1, ..., v_n$, we have for all $k = 1, ..., i-1$ the vertex $v_k$ simplicial in both the graphs $G_{i-1}[\{v_k, v_{k+1}, ..., v_n\}]$ and $G_n[\{v_k, v_{k+1}, ..., v_n\}]$. Note also that $v_k$ has the same neighbors in both these graphs.

Hence $v_k$ is also simplicial in $M_i$ since $M_i$ is a supergraph of $G_{i-1}$ and a subgraph of $G_n$. For chordal graphs it is well-known that the elimination order resulting from repeatedly eliminating a vertex which is simplicial in the graph induced by non-eliminated vertices, is a perfect elimination ordering. We can therefore find a perfect elimination ordering $\gamma$ of $M_i$ with $\gamma = w_1, ..., w_n$ and $v_k = w_k$ for $k = 1, ..., i-1$.

We now show that $(G;\gamma) = M_i$. Eliminating $v_1, ..., v_{i-1}$ in this order in $G$ or $G_{i-1}$ will in either case give us $G_{i-1}$. Since $M_i$ is a minimal chordal supergraph of $G_{i-1}$ we must have $(G_{i-1};\gamma) = M_i$. Hence, $(G;\gamma) = M_i$ since the initial sequence of $\gamma$ is $v_1, ..., v_{i-1}$. $\Box$

We can now state the main theorem of this section.

**Theorem 3.8** *Let $\alpha = v_1, ..., v_n$ be an elimination order of a graph $G$. Let $M_i$ be both a subgraph of $(G;\alpha)$ and a minimal chordal supergraph of $G_{i-1}$. For any graph $M$ which is both a subgraph of $M_i$ and a chordal supergraph of $G$, we have $E(M_i) \setminus E(G_{i-1}) \subseteq E(M)$.*

**Proof.** Observe that $G \subseteq G_{i-1} \subseteq M_i \subseteq (G;\alpha)$ and $G \subseteq M \subseteq M_i$. By Lemma 3.7, we know that $M_i$ has a perfect elimination ordering $\gamma = w_1, w_2, ..., w_n$ with $v_k = w_k$ for all $k = 1, ..., i-1$ and we also know that $(G_{i-1};\gamma) = M_i$. The statement in the theorem says that if the edge $uv$ of $M_i$ is a fill edge created, in the process giving $(G_{i-1};\gamma) = M_i$ from $G_{i-1}$, by the elimination of some $w_j$ with $i \leq j \leq n$, then $uv$ must be an edge of $M$.

We prove this by contradiction. Assume that $uv$ is the latest introduced fill edge violating the condition. In other words, $uv$ is created in the process giving $(G_{i-1};\gamma) = M_i$ by $w = w_j$, $uv$ is not an edge of $M$, and every other fill edge of this elimination process created by any $w_k$ with $k > j$ is an edge of $M$. Since $M_i$ is a minimal chordal supergraph of $G_{i-1}$, and $uv$ is a fill edge created by $w$, by Theorem 3.2 and Corollary 3.3 there must exist a vertex $x$ which in $M_i$ is not adjacent to $w$ but is adjacent to both $u$ and $v$, with $\gamma(u) > \gamma(w)$, $\gamma(v) > \gamma(w)$ and $\gamma(x) > \gamma(w)$.

We will show that there exists in $M$, which is a subset of $M_i$, a path from $w$ to $x$ whose internal vertices have $\gamma$-order less than $w$ and $x$. We will thereby arrive at a contradiction since, with $\gamma$ being a perfect elimination ordering of $M_i$, such a path would imply, by Lemma 3.1, that $wx$ is an edge of $M_i$. But $w$ and $x$ are not adjacent in $M_i$.

Since neither $uv$ nor $wx$ are edges in the chordal graph $M$, at least one of $wu, ux, xv$ and $vw$ is not an edge of $M$. But $M$ is a supergraph of $G$ so any of $wu, ux, xv$ and $vw$ which is not an edge of $M$ must be a fill edge in the elimination process $(G;\gamma)$ giving $M_i$. By assumption, $uv$ is the latest fill edge in this elimination process which is not in $M$, so any of $wu, ux, xv$ and $vw$ which is not an edge of $M$, must be a fill edge of $M_i$ created earlier than $uv$. (Note it could not be created at the same time as $uv$.) Therefore, by Lemma 3.1 there must exist paths $w \to u, u \to x, x \to v$, and $v \to w$ in $G \subseteq M$ whose internal vertices (if any) are earlier in the $\gamma$-order than $w$ and thus also earlier than $u, x, v$. Consider the combined paths $u \to w \to v$ and $u \to x \to v$. If these two paths intersect in an internal vertex $y$ then we find a path $w \to y \to x$ giving the desired contradiction. Otherwise, consider the shortest path $u \to w' \to v$ in $M$ from $u$ to $v$ using only vertices on the $u \to w \to v$ path and the shortest path $u \to x' \to v$ in $M$ from $u$ to $v$ using only vertices on the $u \to x \to v$ path. These paths must have an internal vertex since $uv$ is not an edge of $M$, and the vertices on these paths therefore induce a subgraph of $M$ containing a $k$-cycle with $k \geq 4$. Since $M$ is chordal, there is a chord on this cycle and this chord must connect an internal vertex $y'$ in $u \to x' \to v$ with an internal vertex $y''$ in $u \to w' \to v$. But then we find a path $w \to y' \to y'' \to x$ giving the desired contradiction. (Note that this argument allows for $w = y'$ or $x = y''$.) $\Box$

From Theorem 3.8, it follows that the fill edges in $F_i$ that are not removed while examining $F_i$, will never become candidates for removal at later steps while examining $F_j$ for $j = i-1, ..., 1$. We are now ready to give a full description of the algorithm in the next section.

# 4 The Algorithm

In this section we develop an algorithm which, given a graph $G$ and an elimination ordering $\alpha$ of $G$, finds a graph $M$ which is both a minimal chordal supergraph of $G$ and a subgraph of $(G;\alpha)$.

After computing $G_n \equiv (G;\alpha)$, and finding $C_i$ and $F_i$ for $i = 1,...,n$ from $G_n$ in a straightforward manner, our algorithm proceeds as follows: The algorithm has $n$ iterations. Initially, we set $M = G_n$. Starting from $i = n$ and going backwards, at each iteration $i$, redundant fill edges in $F_i$ introduced by the elimination of $v_i$ are removed. By Theorem 3.8, we know that the remaining edges of $F_i$ need not be considered for removal at later iterations. The algorithm has $n$ iterations for simplicity, but actually $n-3$ would suffice since $F_n$ and $F_{n-1}$ are empty and $F_{n-2}$ can always be removed.

The full algorithm is given in Figure 1. The subroutine that checks an edge $uv \in F_i$ for its candidacy for removal is called CandidateEdge, and is based on Corollary 3.3. The subroutine LEX-M is used to decide which of the candidate edges in $F_i$ can be removed and which must stay to preserve chordality. $W_i$ is the subgraph of $C_i$ on which we run LEX-M to find out which edges in $Candidate(i)$ are necessary. The original LEX-M was introduced by Rose, Tarjan and Lueker in [20], and finds an elimination ordering resulting in a minimal chordal supergraph of a given graph. For ease of understanding we assume that LEX-M$(W_i)$ returns the set $KeepFill(i) \subseteq Candidate(i)$ of fill edges whose addition to $W_i$ produces a minimal chordal supergraph of $W_i$. Note that $W_i \cup Candidate(i)$, if nonempty, is a clique and a subgraph of $C_i$.

Since the resulting graph $M$ is a minimal chordal supergraph of $G$, every perfect elimination ordering $\beta$ of $M$ gives $(G;\beta) = M$. A linear-time algorithm for finding a perfect elimination ordering of a chordal graph is given in [20] and is called LEX-P. Figure 2 illustrates how our algorithm processes a graph on 7 edges, with details of the calls of LEX-M in Figure 3. We first prove correctness of the algorithm and then consider its time complexity.

**Theorem 4.1** *Algorithm MinimalChordal on input $G$ and $\alpha = v_1,...,v_n$ finds a graph $M$ which is both a minimal chordal supergraph of $G$ and a subgraph of $(G;\alpha)$.*

**Proof.** Denote the graph $M$ at the beginning of iteration $i$ of the main loop of the algorithm by $M_{i+1}$, and at the end of iteration $i$ by $M_i$. Since $M_i \subseteq M_{i+1}$ and $M_{n+1}$ is initialized to $G_n \equiv (G;\alpha)$ it is clear that each $M_i$ is a subgraph of $(G;\alpha)$. In addition, the algorithm has the loop invariant: "The graph $M_i$ is a minimal chordal supergraph of $G_{i-1}$." We show this by reverse induction on $i$ from $n+1$ to 1. The loop invariant is clearly true initially for the graph $M_{n+1} = G_n$. The edges $E(M_{i+1})$ are of four types:

(1) edges belonging to the original graph G.

(2) fill edges that were introduced before the elimination of $v_i$ which will be considered for removal at later iterations.

(3) fill edges that were introduced after the elimination of $v_i$ which have not been removed at earlier iterations.

(4) fill edges belonging to $F_i$.

The graphs $G_i$ and $G_{i-1}$ both contain all the edges of types 1 and 2, so these edges must belong to any chordal supergraph of $G_{i-1}$. Since $M_{i+1}$ is a subgraph of $G_n$ and a minimal chordal supergraph of $G_i$ we know by Theorem 3.8 that no edges of Type 3 can be removed from $M_{i+1}$ and still give a chordal supergraph of $G$, where $E(G) \subseteq E(G_{i-1})$.

Hence, we need only show that (A) $M_i$ is chordal and that (B) any edge $uv$ of $F_i$ which remains in $M_i$ is not a candidate for removal. Consider Case (B)

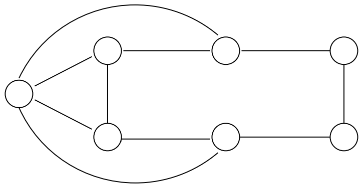
**Algorithm** MinimalChordal $(G, \alpha)$;
**Input:** A graph $G$ and an elimination ordering $\alpha = v_1, ..., v_n$ of $G$.
**Output:** 1. A chordal graph $M$ which is both a minimal chordal supergraph of $G$ and a subgraph of the filled graph $(G;\alpha)$.
2. A minimal elimination order $\beta$ of $G$ s.t. $M$ is the filled graph $(G;\beta)$.

```
begin
    Find (G; α) and C_i, F_i for i = 1, 2, ..., n;
    M = (G; α);
    for i = n downto 1 do
        Candidate(i) = ∅;
        Incident(i) = ∅;
        for all edges uv ∈ F_i do
            if CandidateEdge(uv, i, M) then
                Candidate(i) = Candidate(i) ∪ {uv};
                Incident(i) = Incident(i) ∪ {u, v};
            end-if
        end-for
        if Candidate(i) ≠ ∅ then
            W_i = C_i[Incident(i)] \ Candidate(i);
            KeepFill(i) = LEX-M(W_i);
            M = M \ (Candidate(i) \ KeepFill(i));
        end-if
    end-for
    return M and β = LEX-P(M);
end
```

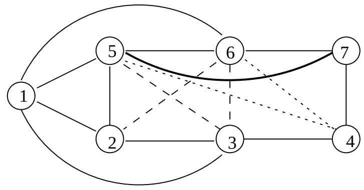
**Function** CandidateEdge$(uv, i, M)$: **boolean**;
**Input:** An edge $uv \in F_i$ and the graph $M$.
**Output:** Returns **true** if $uv$ is a candidate to be removed from $M$, **false** o.w.

```
begin
    cand = true;
    for each neighbor x of u do
        if α(x) > i and xv ∈ E(M) and xv_i ∉ E(M) then
            cand = false;
    end-for
    return cand;
end
```

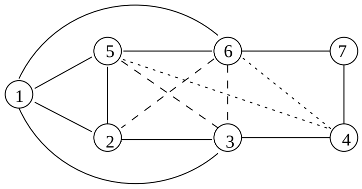
Figure 1: Algorithm MinimalChordal and Function CandidateEdge.



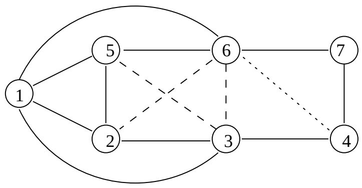
**1. The original graph G.**



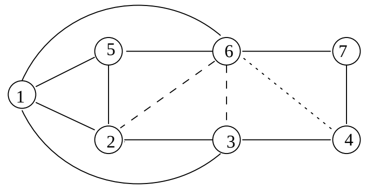

**2. The filled graph of G resulting from the shown elimination ordering.**
The fill edges introduced by vertices 1, 3 and 4 are drawn with respectively dashed, dotted and thick lines.



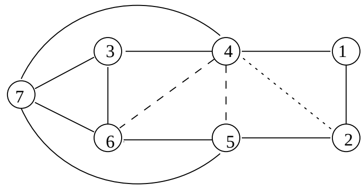

**3. After removal of fill introduced by vertex 4.**
The edge (5,7) was a candidate edge and was removed.




**4. After removal of fill introduced by vertex 3.**
The edge (4,5) was a candidate edge and was removed, whereas (4,6) was not a candidate edge since (6,7) and (4,7) are edges, and (3,7) is not an edge.




**5. After removal of fill introduced by vertex 1.**
Edges (2,6) and (3,5) were candidate edges, whereas (3,6) was not a candidate edge since (6,4) and (3,4) are edges and (2,4) is not. One of the candidate edges can be removed, and (3,5) was chosen.




**7. A perfect elimination ordering on the resulting minimal chordal supergraph of G.**

Figure 2: An example illustrating the algorithm. The calls of LEX-M in Steps 4 and 5 are shown in detail in Figure 3.

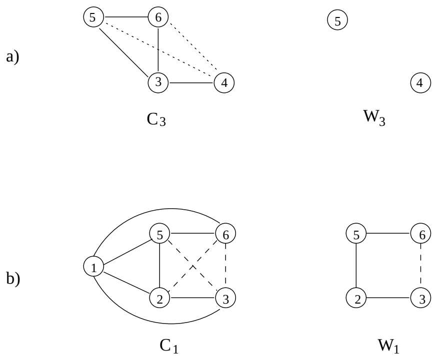



Figure 3: Steps 4 and 5 of the previous example (Figure 2) shown in detail. a) $Candidate(3) = \{(4,5)\}$, and $Incident(3) = \{4,5\}$. The subgraph $W_3$ is defined as $C_3[Incident(3)] \setminus Candidate(3)$. LEX-M($W_3$) returns no fill, hence $(4,5)$ is removed from $M$. b) $Candidate(1) = \{(2,6),(3,5)\}$, and $Incident(1) = \{2,3,5,6\}$. LEX-M($W_1$) returns $(2,6)$ as fill, thus only $(3,5)$ can be removed from $M$ as redundant fill.

first. By Theorem 3.2 it suffices to show that any such edge $uv$ is the only chord of a 4-cycle in $M_i$. There are two cases: either (B.1) $uv \in Candidate(i)$ or (B.2) not.

(B.1) At the beginning of iteration $i$, the neighbors among $\{v_{i+1},...,v_n\}$ in $M_{i+1}$ of vertex $v_i$ induce a clique $C_i$, and the fill edges $F_i$ are all contained in this clique. The edges in $F_i$ that are candidates for removal are identified as $Candidate(i)$, and the vertices which are incident to at least one candidate edge, are stored in $Incident(i)$. We set $W_i$ to be the subgraph of $C_i$ induced by the vertices in $Incident(i)$ minus the edges in $Candidate(i)$. Candidate edges of $F_i$ may be removed in the call of LEX-M($W_i$), which gives us a set $KeepFill(i) \subseteq Candidate(i)$ of fill edges that when added to $W_i$ gives a minimal chordal supergraph of $W_i$. The edges in $Candidate(i) \setminus KeepFill(i)$ are then removed from $M_{i+1}$ to give us $M_i$. An edge $uv \in KeepFill(i)$ is guaranteed, by the correctness of algorithm LEX-M and Theorem 3.2, to be a unique chord of a 4-cycle in the graph $W_i \cup KeepFill(i)$ and therefore also in $M_i$, since $W_i \cup KeepFill(i)$ is an induced subgraph of $M_i$.

(B.2) If $uv$ is found not to be a candidate in the call CandidateEdge$(uv, i, M_{i+1})$ then it is because there exists a vertex $x$ which in $M_{i+1}$ is a neighbor of both $u$ and $v$ but is not a neighbor of $v_i$. Note that the 4-cycle $v_i, u, x, v, v_i$ contains only edges of Type 1, 2 or 3 and $uv$ is therefore a unique chord of this 4-cycle also in $M_i$.

(A) It remains to show that $M_i$ is chordal, which we do by contradiction. Let $S$ be the vertices on a shortest chordless cycle of length at least 4 of $M_i$. We know that there must exist at least two vertices $a, c$ in $S$ such that $ac \in Candidate(i) \setminus KeepFill(i)$, since $M_{i+1} = M_i \cup (Candidate(i) \setminus KeepFill(i))$ is chordal. Also $S$ must contain at least one vertex $b \notin V(C_i)$ since the graph $M_i[V(C_i)]$ is chordal ($M_i[V(C_i)]$ contains the minimal chordal supergraph of $W_i$ computed by LEX-M and the vertices in $V(C_i) \setminus Incident(i)$ which are incident to every other vertex in $V(C_i)$.) Moreover, every vertex $x \in S$ must have $\alpha(x) \geq i$ since $M_i$ contains all fill edges $F_1, ..., F_{i-1}$, and the cycle induced by $S$ contains no chords. We can therefore find a path $a, b_1, ..., b_k, c$ in $M_i[S]$ with $k \geq 1$, $a$ and $c$ in $V(C_i)$, $ac \in Candidate(i) \setminus KeepFill(i)$, and $b_j, 1 \leq j \leq k$, not in

$V(C_i)$. But then we have a $(k+3)$-cycle $a, b_1, ..., b_k, c, v_i, a$ in $M_{i+1}$, which $ac$ is the only chord of since none of $b_j$ is in $V(C_i)$. If $k = 1$, then this contradicts that $ac \in Candidate(i)$. If $k \geq 2$, then $a, b_1, b_2, ..., b_k, c, a$ induce a chordless $(k+2)$-cycle in $M_{i+1}$, contradicting that $M_{i+1}$ is chordal.

The loop invariant therefore holds as claimed. Note that this establishes correctness of the algorithm as upon termination the returned graph $M_1$ is both a minimal chordal supergraph of $G_0 = G$ and a subgraph of $G_n \equiv (G; \alpha)$. □

**Theorem 4.2** *Let $\alpha = v_1, v_2, ..., v_n$ be an elimination ordering of a connected graph $G$, with $n = |V(G)|$, $e = |E(G)|$ and $f = \sum_{i=1}^{n} |F_i|$. The time complexity of Algorithm MinimalChordal($G; \alpha$) is $O(f(e+f))$.*

**Proof.** Computing $G_n$ (see [21]) and the call of LEX-P (see [20]) both take time linear in the size of the filled graph. From $G_n$, the sets $F_i$ and $C_i$ can be computed in time $O(nf)$ by simply examining every fill edge for every vertex. The algorithm has $n$ iterations. At each iteration $i$, CandidateEdge is called $|F_i|$ times, and LEX-M is called once. The time complexity of CandidateEdge($uv, i, M$) is $O(|N_G(u)|) = O(n)$. The time complexity of LEX-M($W_i$) is $O(|V(W_i)||E(W_i)|)$, (see [20]). Thus for the whole algorithm, we get:

$$O\left(\sum_{i=1}^{n}(|V(W_i)||E(W_i)| + |F_i|n)\right)$$

By the definition of $W_i$, $|V(W_i)| = |Incident(i)| \leq 2|F_i|$. Clearly, $|E(W_i)| \leq (e+f)$. Thus, the time complexity of LEX-M($W_i$) becomes $O(|F_i|(e+f))$. Since all the $F_i$ are disjoint, $\sum_{i=1}^{n} |F_i|n = O(fn)$, and $\sum_{i=1}^{n} |F_i|(e+f) = O(f(e+f))$. Since $G$ is connected, $n \leq (e+f)$, and the overall time complexity is $O(f(e+f))$ □

We would like to emphasize that the upper bound given on the time complexity is met only if the subgraphs $W_1, W_2, ..., W_n$ overlap heavily so that the calls of LEX-M involve some large part of the graph several times. However, the examples we have studied indicate that these subgraphs do not overlap very much in practical applications. In the example of Figure 3 we see that the subgraphs $W_i$ do not overlap on edges at all. Our numerical tests presented in the next section, show that if the original ordering is a low-fill ordering (like minimum degree), then there is very little overlap between the mentioned subgraphs. On the other hand, orderings resulting in heavy fill give more overlap, and we have constructed an example that matches the upper bound of the time complexity, showing that we indeed have a $\Theta(f(e+f))$ algorithm.

# 5 Numerical results

In this section, we present the numerical values of tests using a FORTRAN90 implementation of Algorithm MinimalChordal, running on graphs corresponding to sparse matrices from real applications. The matrices that form our sample are taken from the well-known Harwell-Boeing collection [8], and downloaded from Matrix Market [4, 3]. We have chosen to include in our sample all $n \times n$ symmetric matrices with $n \leq 500$ that appear in this collection, and that are available in Matrix Market format. We have omitted diagonal matrices and completely dense matrices since these correspond respectively to graphs on isolated vertices and complete graphs. This constitutes a total of 51 matrices, coming from several families, but even within a family the structural properties of the corresponding graphs can vary widely.

For each matrix, we test two initial orderings on the corresponding graph. First we find the filled graphs produced by a minimum degree (MD) ordering and by a nested dissection (ND) ordering. In Figure 4 we have plotted the number of fill edges $f$ produced by the MD ordering versus the number of vertices $n$ of the graph. As the worst-case time complexity of our MinimalChordal algorithm depends heavily on the value of $f$, it is comforting to see that in most cases we have $f \leq 10n$. Of the 51 tested matrices, 12 have MD fill below $n$.

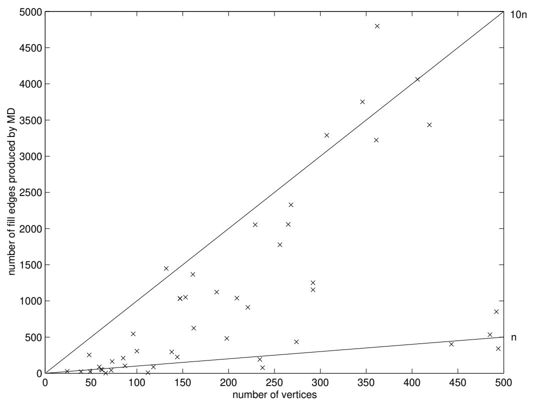



Figure 4: The number of fill edges after MD ordering (y-axis) versus number of vertices (x-axis). The linear curves $f = n$ and $f = 10n$ are shown. To improve the scale we have left out the three data points with highest fill: (420, 6461), (445,8030), (468,14929).

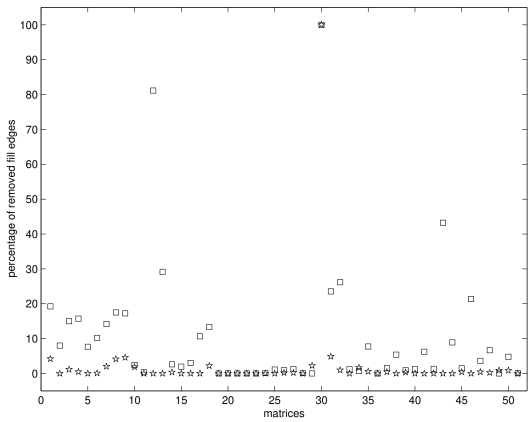



Figure 5: The percentage of fill edges removed (y-axis) by Algorithm MinimalChordal for each matrix (x-axis). Stars denote fill removed from the MD ordering, and boxes denote fill removed from the ND ordering.

Then we minimalize these filled graphs by Algorithm MinimalChordal. The number of fill edges, $f$, produced by an initial ordering is compared to the number of fill edges removed by Algorithm MinimalChordal. In Figure 5 we have plotted the percentage of removed fill edges. It is interesting to note that the fill produced by the MD ordering is in many cases already minimal (*i.e.* the star is on the 0 percent line), whereas for ND ordering a substantial amount is removed in many cases. For our test set, MD gave a minimal ordering for 51% of the matrices, whereas ND was minimal for about 16%.

The FORTRAN code used for MD is implemented by Joseph Liu [14], and it is among the fastest known implementations of this heuristic. For ND, we used the public domain FORTRAN90 code from METIS [11]. Our FORTRAN90 code is a straightforward implementation carried out by a graduate student, and with limited time for enhancements. After removal of comments our code has 600 lines, about the same as the established MD code. About a third of these lines is the actual MinimalChordal algorithm, with the rest taken up by preprocessing that finds the fill edges produced by the initial orderings, and by the LEX-M subroutine (to our knowledge, LEX-M has no public domain code.)

| matrix | $n$ | $e$ | order | $f$ | red. fill | red. % | time order | time MC | time factor |
|---|---|---|---|---|---|---|---|---|---|
| bcsstk01 | 48 | 176 | MD | 253 | 0 | 0.0 | 1.3 | 9.0 | 6.9 |
| | | | ND | 289 | 1 | 0.4 | 3.3 | 9.9 | 3.0 |
| bcsstk03 | 112 | 264 | MD | 8 | 0 | 0.0 | 1.3 | 8.0 | 6.2 |
| | | | ND | 138 | 112 | 81.2 | 1.0 | 14.0 | 14.0 |
| bcsstk04 | 132 | 1758 | MD | 1449 | 6 | 0.4 | 37.2 | 295.8 | 8.0 |
| | | | ND | 1762 | 26 | 1.5 | 32.3 | 360.2 | 11.2 |
| bcsstk05 | 153 | 1135 | MD | 1051 | 0 | 0.0 | 28.8 | 192.0 | 6.7 |
| | | | ND | 1516 | 324 | 21.4 | 21.1 | 285.2 | 13.5 |
| bcsstk06 | 420 | 3720 | MD | 6461 | 27 | 0.4 | 128.4 | 1173.3 | 9.1 |
| | | | ND | 8053 | 291 | 3.6 | 66.8 | 1529.6 | 22.9 |
| bcsstk08 | 1074 | 5943 | MD | 23111 | 49 | 0.2 | 263.4 | 1943.6 | 7.4 |
| | | | ND | 28008 | 1035 | 3.7 | 64.5 | 2642.3 | 41.0 |
| bcsstk19 | 817 | 3018 | MD | 4470 | 83 | 1.9 | 22.1 | 196.9 | 8.9 |
| | | | ND | 5894 | 1475 | 25.0 | 13.7 | 307.5 | 22.5 |
| bcsstk20 | 485 | 1325 | MD | 534 | 0 | 0.0 | 9.0 | 54.4 | 6.0 |
| | | | ND | 967 | 282 | 29.2 | 3.6 | 70.3 | 19.5 |
| bcsstk22 | 138 | 279 | MD | 295 | 1 | 0.3 | 2.6 | 17.0 | 6.5 |
| | | | ND | 306 | 8 | 2.6 | 2.0 | 16.8 | 8.4 |

Table 1: Columns 2 and 3 show the number of vertices and the number of original edges in the associated graph respectively. For each matrix, there are two rows, one for each of the initial ordering heuristics MD and ND. For each initial ordering, we compute the fill in Column 5, and the fill removed by MinimalChordal in Column 6. The percentage reduction in the number of fill edges is given in Column 7. The last three columns show respectively the runtime in milliseconds of the ordering algorithm and of Algorithm MinimalChordal, and the multiplicative factor between these.

In Table 1, we have chosen an arbitrary family of matrices from the collection, and display in addition to numerical values of vertices, edges and fill, also the exact runtimes of the ND, MD and MC (MinimalChordal) codes. The matrices in this family display high variance on all these properties, and give a good indication of the general behavior on the whole test collection. For this test, we have taken all the appropriate matrices of the family bcsstk without the size limit of 500 vertices, giving us two larger matrices outside the initial test set. It is interesting to note that our MC code is never more than 10 times slower than the MD code while it is up to 40 times slower than the ND code. On the other hand, substantially more fill is removed from the ND orderings. Compared to the amount of hours spent enhancing the MD and ND implementations, we believe our MC code has an acceptable speed in practice.

# 6 Conclusion

As mentioned in the introduction, minimal low fill orderings are desirable, but minimal fill is not necessarily low fill. Practical algorithms like minimum degree and nested dissection usually produce low fill orderings, but not necessarily minimal fill orderings. Therefore we assumed a filled graph (preferably with low fill) already given and considered the problem of removing fill edges to produce a minimal filled graph (with lower fill). For this purpose, we have introduced Algorithm MinimalChordal. An interesting question that arises is the amount of fill edges that can be removed in practice. The numerical results presented in the previous section show that minimum degree orderings are often close to minimal, whereas nested dissection orderings produce fill that can be reduced considerably by a minimalization. The time complexity of our algorithm is dependent on the number of fill edges we start with. In some cases this number is already low, in other cases it is high and our algorithm may remove many redundant fill edges. Even if we do not believe we can successfully express the worst-case time bound of our algorithm as a function of the number of fill edges removed, our numerical results indicate that on inputs from real applications it may be possible to show an interesting correlation between the runtime of the implementation of our algorithm and the fill removed. Although the worst-case time bound of our algorithm may not be acceptable to the practitioners, the runtime in practice seems to be good and this gives us hope that with further improvements of the implementation, it could be used in practice as intended.


# Acknowledgements

The authors are indebted to Barry Peyton for suggesting the problem of finding a minimal chordal supergraph from a non-minimal chordal supergraph. The FORTRAN90 implementation of MinimalChordal and LEX-M are done by Ole-Martin Kvinge. We are grateful to Joseph Liu for letting us use his minimum degree code for research purposes.


# References


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

[2] J. R. S. Blair, P. Heggernes, and J. A. Telle. Making an arbitrary filled graph minimal by removing fill edges. In R. Karlsson and A. Lingas, editors, *Algorithm Theory - SWAT '96*, pages 173–184. Springer Verlag, 1996. Lecture Notes in Computer Science 1097.

[3] R. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. Dongarra. Matrix market : a web resource for test matrix collections. In R. Boisvert, editor, *The Quality of Numerical Software: Assessment and Enhancement*, pages 125–137. Chapman and Hall, 1997.

[4] R. Boisvert, R. Pozo, K. Remington, B. Miller, and R. Lipman. *NIST Matrix Market.* http://math.nist.gov/MatrixMarket/.

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

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

[7] E. Dahlhaus and M. Karpinski. An efficient parallel algorithm for the minimal elimination ordering of an arbitrary graph. *Proceedings FOCS*, pages 454–459, 1989.

[8] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15:1–14, 1989.

[9] R. E. England, J. R. S. Blair, and M. G. Thomason. Independent computations in a probablistic knowledge-based system. Technical Report CS-90-128, Department of Computer Science, University of Tennessee, Knoxville, Tennessee, 1991.

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

[11] G. Karypis. *METIS Family of multilevel partitioning algorithms*. http://www-users.cs.umn.edu/~karypis/metis/.

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

[13] J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Sci. Stat. Comput.*, 10:1156–1173, 1989.

[14] J. W. H. Liu. Private communication.

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

[16] J. W. H. Liu and A. Mirzaian. A linear reordering algorithm for parallel pivoting of chordal graphs. *SIAM J. Disc. Math.*, 2:100–107, 1989.

[17] 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.

[18] B. W. Peyton. *Some applications of clique trees to the solution of sparse linear systems*. PhD thesis, Dept. of Mathematical Sciences, Clemson University, 1986.

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

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

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

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