Minimal Elimination Ordering Inside a Given Chordal Graph *

Elias Dahlhaus
Dept. of Computer Science and Dept. of Mathematics
University of Cologne, Cologne, Germany
e-mail: dahlhaus@suenner.informatik.uni-koeln.de

Keywords: Gauss elimination, sparse matrices, chordal graphs, nested dissection, minimum degree heuristics.

Abstract

We consider the following problem, called *Relative Minimal Elimination Ordering*. Given a graph $G = (V, E)$ which is a subgraph of the chordal graph $G' = (V, E')$, compute an inclusion minimal chordal graph $G'' = (V, E'')$, such that $E \subseteq E'' \subseteq E'$. We show that this can be done in $O(nm)$ time. This extends the results of [2]. The algorithm is based only on well known results on chordal graphs.

1 Introduction

One of the major problems in computational linear algebra is that of sparse Gauss elimination. The problem is to find a pivoting, such that the number of zero entries of the original matrix that become non zero entries in the elimination process is minimized. In case of symmetric matrices, we would like to restrict pivoting along the diagonal. We consider the graph G consisting of the vertex set $\{1, \ldots, n\}$ where n is the number of rows or columns of the given matrix and two verices i and j are joined by an edge if and only if the corresponding entry of the matrix is a non-zero entry. When restrict pivoting along the diagonal, we create new non-zero entries as follows. An entry a_{ij} becomes a non-zero entry there is a $k < i, j$, such that a_{ki} and a_{jk} are non-zero entries or have become non-zero entries. The graph theoretical interpretation is that in increasing order, we select a vertex i and join all greater neighbors of i (with a greater number than i) pairwise by an edge the edges that are added to the graph G are called *fill-in edges*. The problem of minimum fill-in is to find a permutation of $\{1, \ldots, n\}$, such that during the pivoting process, the number of non-zero entries is minimized. In terms of graph theory, we are interested to get a numbering of the vertices of the given graph, such that the fill-in is minimized.

Unfortunately, this problemm is NP-complete [19]. One approach to relax the problem is to find a numbering of the vertices, such that the corresponding fill-in is minimal with respect to the subset relation (Minimal Elimination Ordering (MEO)). This problem can be solved in $O(nm)$ time [15]. Unfortunately, a minimal fill-in can have a size that is far from the size of a fill-in of minimum cardinality. This is shown by the following example.

*A preliminary version appeared in WG 97 [6], partially supported by ESPRIT Long Term Research Project Nr. 20244 (ALCOM-IT)

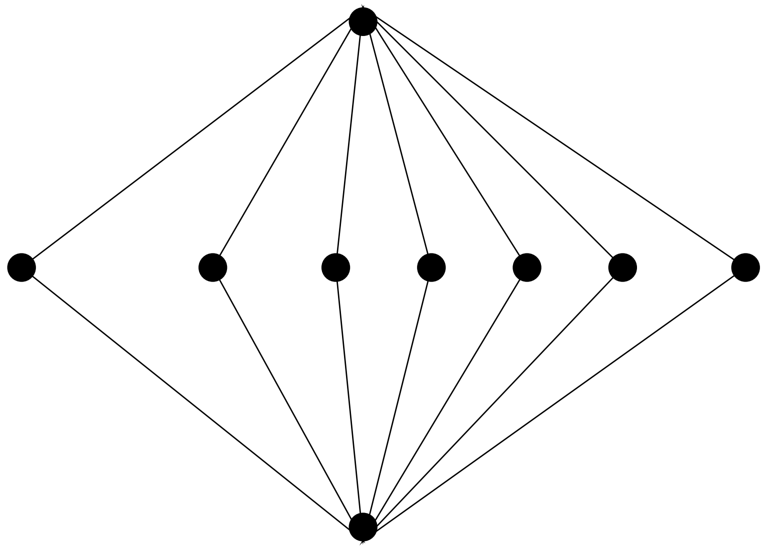

Figure 1: A graph with a small minimum fill-in and a large minimal fill-in

The vertex set of G consists of a vertex set $V = X \cup \{x\} \cup \{y\}$ and an edge set $\{xv|v \in X\} \cup \{yv|v \in X\}$ (see figure 1).

Numbering x first and y last leeds to a fill-in, such that all vertices in X are pairwise adjacent. This fill-in is also a minimal fill-in (see figure 2).

Numbering the vertices of X first and x and y last leeds to a fill-in that consists only of the edge xy (see figure 3).

There are two practical polynomial time heuristics to get "good" elimination orderings, the minimum degree heuristics (see for example [11]) and nested dissection heuristics (see for example [1] or [11]).

Neither the minimum degree heuristics nor the nested dissection method computes necessarily an elimination odering, such that the fill-in is minimal with respect to the subset relation.

The minimum degree heuristics repeatedly selects and numbers a vertex v with a minimum number of unnumbered neighbors and the unnumbered neighbors of v are made pairwise adjacent. We consider the graph G consisting of two vertex disjoint cliques C_1 and C_2 and a vertex v that is adjacent to exactly one vertex of C_1 and one vertex of C_2. The minimum degree heuristics would select v first and create a fill-in edge that joins the two neighbors of v. On the other hand numbering the vertices of C_1 first, the vertices of C_2 second, and numbering v last would leed to an empty set of fill-in edges.

Note that fill-in graphs are always *chordal*, i.e. every cycle of length greater than three has a pair of non consecutive vertices that are joined by an edge (also called *chord*). Chordal graphs are exactly those graphs having an ordering with no fill-in edge (called *perfect elimination ordering*). The problem of minimum fill-in is therefore equivalent to find a smallest extension of the edge set of the given graph that is chordal. The problem of a minimal elimination ordering is equivalent to the problem to find a subset minimal extension of the edge set that is chordal.

We are interested in the problem to combine one of the heuristics as mentioned above with

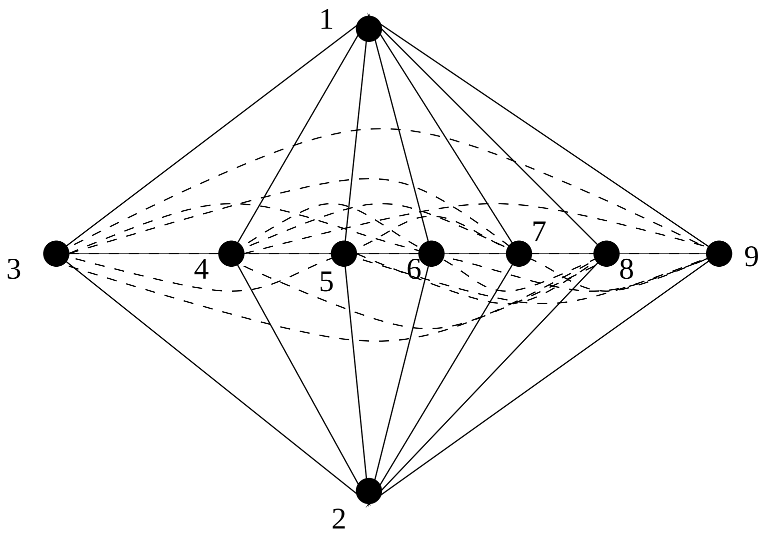

Figure 2: The large minimal fill-in

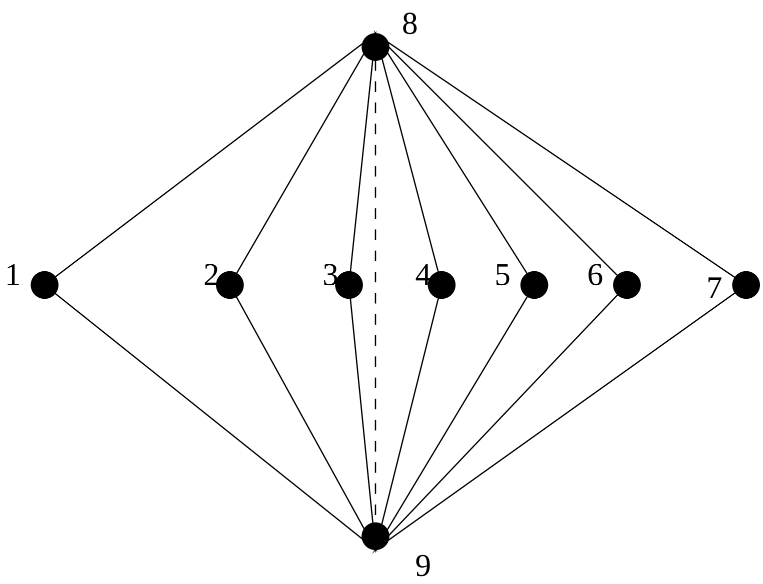

Figure 3: The minimum fill-in

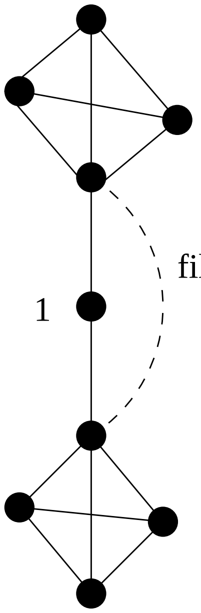

Figure 4: Minimum degree heuristics does not necessarily leed to a minimal fill-in

minimal fill in, i.e. we first apply one of the heuristics to get an odering $<$ and afterwards we further thin out the resulting chordal graph $G'_<$ that consists of the edges of G and the fill-in edges of G and $<$, such that we get a minimal fill in ordering $<'$ with $G'_{<'} \subseteq G'_<$.

In general, we consider the following problem.

Relative Minimal Elimination Ordering: Given a graph $G = (V, E)$ and an ordering $<$, find another ordering $<'$, such that the fill-in of $<'$ is minimal with respect to the subset relation and a subset of the fill-in of $<$.

Blair, Heggernes, and Telle [2] were the first dealing with this problem and the run time of their algorithm is $O(f(m + f))$, where m is the number of original edges and f is the number of fill-in edges, i.e. additional edges of $G_< \setminus G$.

Here we present an algorithm with a time bound of $O(nm)$ that is at least better in theory.

In section 2, we introduce the notation and basic results that are necessary for the paper. In section 3, we describe the basic strategy of the algorithm consisting of a "tree splitting" procedure as a preprocessing procedure to the RTL-algorithm and an improved Rose-Tarjan-Lueker algorithm (improved RTL-algorithmm). In section 4 introduce and show the correctness of the tree splitting procedure. In section 5 we show the correctness of an improved version of the RTL-algorithm.

2 Notation

A *graph* $G = (V, E)$ consists of a *vertex set* V and an *edge set* E. Multiple edges and loops are not allowed. The edge joining x and y is denoted by xy.

We say that x is a *neighbor* of y iff $xy \in E$. The set of neighbors of x is denoted by $N(x)$ and is called the *neighborhood*. The set of neighbors of x and x is denoted by $N[x]$ and is called the *closed neighborhood* of x.

Trees are always directed to the root. The notion of the *parent*, *child*, *ancestor*, and *descendent* are defined as usual.

A *subgraph* of (V, E) is a graph (V', E') such that $V' \subseteq V$, $E' \subseteq E$.

We denote by n the number of vertices and by m the number of edges of G.

A graph is called *chordal* iff each cycle of length greater than three has a chord, i.e. an edge that joins two nonconsecutive vertices of the cycle. Note that chordal graphs are exactly those graphs having a *perfect elimination ordering* $<$, i.e. for each vertex v the neighbors $w > v$ induce a complete subgraph, i.e. they are pairwise joined by an edge [9].

Moreover, chordal graphs $G = (V, E)$ are exactly the intersection graphs of subtrees of a tree [10, 3], i.e. there is a tree T and a collection of subtrees T_v, $v \in V$, such that $vw \in E$ if and only if T_v and T_w share a node. We call $(T, T_v)_{v \in V}$ also a *tree representation of* G. Let $c_t := \{v | t \in T_v\}$. Note that all vertices in c_t are pairwise adjacent. Note that one always can find a tree representation $(T, T_v)_{v \in V}$ of G, such that the sets c_t are exactly the maximal cliques of the chordal graph G [10, 3]. In this case, T is also called a *clique tree* of G. A clique tree can always be determined in linear time [18].

Note that in any chordal graph, the number of maximal cliques is bounded by n and the number of pairs (x, c) such that x is in the clique c is bounded by m.

3 The Basic Algorithmic Idea

We first compute the fill-in E' of G and $<$. Then we compute a clique tree T of $G'_< := (V, E \cup E')$.

Note that the edges of T correspond to the *cuts* of $G'_<$. We consider for each edge st of T the set c_{st} of T_v passing the edge st, i.e. $c_{st} := c_s \cap c_t$. c_{st} separates $G'_<$ into at least two connected components (i.e. $G'_< - c_{st}$ has at least two connected components) and there are at least two connected components C_1 and C_2 of $G'_< - c_{st}$, such that all vertices of c_{st} have a neighbor in C_1 and a neighbor in C_2 (c_{st} is a *cut*). Note that cuts of $G'_<$ are not necessarily cuts of G. But in a minimal elimination ordering, all cuts of $G'_<$ are also cuts of G [13]. We continue as follows.

1. We split the cuts of $G'_<$ into cuts of G. We get a new tree representation $(T_1, T_v)_{v \in V}$ and a chordal graph G_1 that is a subgraph of $G'_<$ and contains G as a subgraph. All cuts of G_1 are cuts of G. $(T_1, T_v)_{v \in V}$ is also called a *quasi-minimal* tree representation of G.

   Theorem 1 *Suppose $(T_1, T_v)_{v \in V}$ is a quasi minimal tree representation of $G_1 = (V, E_1)$. Then all edges uv, such that T_u and T_v share an edge of T_1, appear in each $G'_{<'} = (V, E'')$, such that $<'$ is a minimal elimination ordering and $E'' \subseteq E_1$.*

   Proof: Suppose T_u and T_v share an edge st of T_1. Let C_1 and C_2 be connected components of $G[V \setminus c_{st}]$, such that all vertices of c_{st} have a neighbor in C_1 and a neighbor in $C - 2$. Since the subtrees $T_x, x \in C_1$ and T_y, $y \in C_2$ are separated by the edge st of T_1, there is no edge $xy \in E_1$ and therefore no edge $xy \in E''$, such that $x \in C_1$ and $y \in C_2$. Consider any path p_1 from u to v with inner vertices in C_1 and any path p_2 from v to u with inner vertices in C_2 in the original graph G. The concatenation of p_1 and p_2 forms a cycle in

$G'_{<'}$ of length ≥ 4. Assume there is no edge $uv \in E''$. Then consider any chordless path p'_1 and p'_2 in $G'_{<'}$ from u to v and v to u respectively, such that their vertices are in p_1 and p_2 respectively. Then the concatenation of p'_1 and p'_2 forms a cycle in $G'_{<'}$ of length at least four. Therefore in $G'_{<'}$, it must contain a chord in E''. Since p'_1 and p'_2 are chordless, one incident vertex must be in p'_1 and therefore in C_1, and the other incident vertex must be in p'_2 and therefore in C_2. This is a contradiction to the fact that there is no edge $xy \in E''$ with $x \in C_1$ and $y \in C_2$. $\square$

2. Note that vertices v and w that have not only a clique but even a cut in common are joined by an edge in any minimal fill-in of G that is a subset of G_1. We add those pairs of vertices to the edge set of G. We could determine a minimal fill-in of each clique of G_1 separately using the algorithm of [15]. This would leed to an $O(n^3)$ time algorithm. We also can consider the clique tree of G_1 and we can determine a post order enumeration of the clique set of G_1. We partition the vertex set of G into levels where a vertex v is put into level L_i if the root clique of v (i.e. the root of T_v in T') has the number i. We will see that we can apply the algorithm of [15] globally to determine a minimal elimination ordering with a fill-in that is a subset of G_1.

The complexity of the first step is known.

Lemma 1 *[15] The fill-in of an ordering $<$ of the vertex set of G can be determined in $O(n^2)$ time.*

As a consequence, also a clique tree T of $G'_<$ can be determined in $O(n^2)$ time.

We therefore may assume that a clique tree T of $G'_<$ is given. Due to the construction of [18] of a clique tree, we may assume that if the root of T_x is a proper descendent of the root of T_y then $x < y$. We also may assume that if t is the parent of s in T then there is a T_v that passes s and that has t as its root.

Moreover, we may observe the following.

Lemma 2 *For each node t of T, the set C_t consisting of all vertices x, such that the root of T_x is t or a descendent of t is connected in G.*

Proof: Otherwise we could get a tree representation consisting of two copies T_1 and T_2 of t and its descendents. The root of T_1 and T_2 have the same parent. The trees T_x beeing in the first connected component of C_t are made subtrees of T_1, and the remaining T_x are made subtrees of T_2. The chordal graph G_1 represented by this tree representation is a proper subgraph of $G'_<$ but still contains G as a subgraph. Moreover also in G_1 all greater neighbors of any vertex are pairwise adjacent (with respect to the same ordering $<$). The reason is that also in the tree representation of G_1, if the root of T_x is a descendent of the root of T_y then $x < y$. Therefore $G'_<$ cannot be the fill-in of G and $<$ (i.e. the smallest extension of G to a chordal graph that has $<$ as a perfect elimination ordering). This is a contradiction. $\square$

Recall that C_t is the set of vertices x, such that the root of T_x is t or a descendent of t and that c_{st} is the set of vertices x, such that T_x passes the edge st.

Lemma 3 *Let s be a node of the clique tree T and t be its parent in T. Then c_{st} is the set of neighbors of C_s in G that do not belong to C_s.*

Proof: Note that all neighbors x of C_s in G are also neighbors of C_s in $G'_<$. Therefore for all neighbors x of C_s, T_x contains at least one node of T that is s or a descendent of s. If moreover $x \notin C_t$ then the root of T_x is not in a descendent of s or s. Therefore T_x contains s or a descendent of s and non descendents of s and therefore s and its parent t.

Vice versa suppose that x is not a neighbor of C_s in G. If T_x would contain s or a descendent of s then we only have to delete all nodes u of T_x from T_x that are s or descendents of s. T_x remains a tree. The chordal graph represented by the new tree representation is a subgraph of $G'_<$ and $<$ remains a perfect elimination ordering. That means T_x cannot contain s or a descendent of s. Therefore T_x cannot pass the edge st of T.

□

4 The Tree Splitting Procedure

We start with the initial tree representation $(T_0, T_v^0)_{v\in V}$ of $G_0 := G'_<$.

We compute a sequence $(T_i, T_v^i)_{v\in V}$ of tree representations that represent chordal graphs $G^i = (V, E^i)$. G^{i+1} is a subgraph of G^i and contains G as a subgraph, and the final tree representation $(T_k, T_v^k)_{v\in V}$ is quasiminimal.

Let $e_1, \dots, e_k$ be an enumeration of the edges of T_0, such that if e_i is an ancestor edge of e_j then $i < j$. For example a postorder enumeration is such an enumeration. We call such an enumeration a *top down enumeration*. Let $e_i = s_i t_i$ where t_i is the parent of s_i. During the algorithm, for each edge $f = st$, let $C_{(s,t)}$ be a connected subset of G, such that all T_u with $u \in C_{(s,t)}$ appear on the s-side of st in T and all vertices w, such that T_w pass st, are in the neighborhood of $C_{(s,t)}$. Note that an edge satisfies the condition of quasi minimality if $C_{(s,t)}$ and $C_{(t,s)}$ are defined. Initially, let $C_{(s_i,t_i)} := C_{s_i}$, i.e. the set of vertices u such that T_u appears only at the s_i-side of e_i. By construction of $(T_0, T_v^0)_{v\in V}$, all these sets are connected in G.

Algorithmically we proceed as follows.

For $i = 1, \dots, k$,
compute T_i from T_{i-1}, i.e.

1. **compute the set C_i of connected components of**

$$G[\{v | T_v \textbf{ appears only on the } t_i\textbf{-side of } T_0\}];$$

2. **for each $c \in C_i$, mark c as *good* if there is a $v \in c$, such that $t_i \in T_v^{i-1}$;**

3. **for each good connected component $c \in C_i$, create a tree node t_c and a tree edge $s_i t_c$;**
   $C_{(s_i,t_c)} := C_{(s_i,t_i)}$; $C_{(t_c,s_i)} := c$;

4. **construct T_i from T_{i-1} as follows: for each edge $t_i u$ of T_{i-1}, let d_u be the component $c \in C_i$ that contains $C_{(u,t_i)}$;**
   if d_u is a good component then
   begin replace $t_i u$ by $t_{d_u} u$;

$C_{(t_{d_u},u)} := C_{(t_i,u)}$ **if defined;** $C_{(u,t_{d_u})} := C_{(u,t_i)}$**; if** $t_i u$ **was an** e_j**,** $j > i$ **then** e_j **is updated by** $t_{d_u} u$**, i.e.** $s_j := u$ **and** $t_j := t_{d_u}$**;**
end
else
begin
replace $t_i u$ **by** $s_i u$**;** $C_{(s_i,u)} := C_{(t_i,u)}$ **if defined;** $C_{(u,s_i)} := C_{(u,t_i)}$**; if** $ut_i = e_j$**, for some** $j > i$**, then** e_j **is updated to** us_i **(**$s_j = u$**;** $t_j = s_i$**);**
end;
erase t_i**;**

5. **(updating** T_v**) for** v **with** $t_i \in T_v^{i-1}$**, construct** $T_v = T_v^i$ **from** $T_v = T_v^{i-1}$ **as follows: for any good component** $c \in C_i$**, add** t_c **to** T_v **if and only if** $v \in c$ **or** v **is a neighbor of some vertex in** c **in** G**;**

To prove the correctness, we have to show that the tree representation $(T_k, T_v^k)_{v \in V}$ *is quasi minimal and that the edge set* E^k *of* G^k *contains* E *and is contained in* E^1.

We say that an edge f of T_j *arises from* e_i if either

1. $j = i$ and f is an edge $s_i t_c$, for some good component c of C_i or
2. $j > i$ and (f is also an edge of T_{j-1} and arises from e_i or there is an edge f' of T_{j-1} that arises from e_i and is replaced by f in T_j).

Note that in each T_i, every edge is either some e_j, $j > i$ or arises from some e_j, $j \leq i$. To show that $(T_k, T_v^k)_{v \in V}$ is quasi minimal, we show that for each $j \leq i$ and each edge $f = st$ that arises from e_j, $C_{(s,t)}$ and $C_{(t,s)}$ are both defined, each $C_{(s,t)}$ defines, for each i, an in G connected subset of V that is adjacent to all vertices of $p_f = \{v | T_v^i \text{ passes } f\}$.

By induction on i, we show

Lemma 4 *For each* i*:*

1. *In* $(T_i, T_v^i)_{v \in V}$*, for all edges* $f = st$ *arising from som* e_j*,* $j \leq i$*,* $C_{(s,t)}$ *and* $C_{(t,s)}$ *are defined.*
2. *For all edges* ut *of* T_i *with* $t = s_j$*,* $j \leq i$ *or* $t = t_c$*,* $c \in C_j$*,* $j \leq i$*,* $C_{(u,t)}$ *is defined.*
3. *For each edge* ut_{i+1} *of* T_i*,* $C_{(u,t_{i+1})}$ *is defined.*
4. *If* $C_{(s,t)}$ *is defined in* $(T_i, T_v^i)_{v \in V}$ *then* $C_{(s,t)}$ *is an in* G *connected subset of* V *and*
5. *for all* T_v^i *passing* st*,* vw *is an edge in* G*, for some* $w \in C_{(s,t)}$
6. T_v^i *is a tree, i.e. defines a connected subset of* T_i*.*
7. *For* $j > i$*, if* us_j *is an edge of* T_i *and* $u \neq t_j$ *then* $C_{(u,s_j)}$ *is defined.*

Proof: We simultaneously prove all the statements by induction.

For $i = 0$, statements 1 and 2 are trivially true, because e_j, $j \leq i$ do not exist. Statements 4, 6, and 7 are true, by construction of $(T_0, T_v^0)_{v \in V}$. Note that t_1 is the root of T^0, and therefore also statement 3 is true, for $i = 0$. , since statement 7 is true.

To show the inductive step, observe that whenever $ut = ut_i$ is replaced by ut', $C_{(u,t)}$ is always defined, since statement 3 is true for $i-1$, $C_{(u,t')} = C_{(u,t)}$, and $C_{(t,u)} = C_{(t',u)}$. Moreover, observe that a new $C_{(t,u)}$ is created if and only if $u = s_i$ and $t = t_c$, for some good component c of C_i. $C_{(t_c,s_i)} = c$ is an in G connected subset of V and for all T_v^i passing $t_c s_i$, v is adjacent to some vertex in c in G. Therefore statement 1 and statement 4 are true, for all i.

Statement 2 is true for $t = s_j$, $j < i$ and $t = t_c$ with $c \in C_j$, $j < i$, because it is true in $(T_{i-1}, T_v^{i-1})_{v \in V}$ and t does not get new incident edges in T_i. If $t = s_i$ then all edges incident with s_i in T_i are either edges incident with s_i in T_{i-1} or edges $us_i = t_c s_i$ or replaced edges us_i (i.e. ut_i was an edge in T_{i-1}). In either cases $C_{(u,s_i)}$ is defined and therefore statement 2 is true for s_i. Suppose now that $t = t_c$ and c is a good component of C_i. Incident edges in T_i are $s_i t_c$ and edges ut_c with ut_i in T_{i-1}. Statement 3 follows for $u = s_i$ from statement 1. For the remaining u, statement 2 follows from the observation that whenever ut_i is replaced by some ut' then $C_{(u,t')}$ remains defined.

Statement 7 is always preserved, because any s_j, $j > i$ is not a t_l, $l \leq i$, because $e_1, \ldots, e_k$ is a top down enumeration of the edges of T_0, and therefore s_j is not of the form t_c and not an s_l, $l \leq j$ and therefore up to replacements, s_j has the same incident edges in T_i as in T_{i-1}.

To show statement 3, observe that t_{i+1} is an s_j, $j \leq i$ or a t_c, $c \in C_j$, $j \leq i$, because $e_1, \ldots, e_k$ is a top down enumeration of the edges of T_0. Therefore statement 3 follows from statement 2.

Next observe that, when we create $(T_i, T_v^i)_{v \in V}$ and replace any edge ut_i by ut then either $t = t_c$, for some good component c or $t = s_i$. In the first case, exactly for those T_v^{i-1} passing ut_i, T_v^i contains u and t_c. In the second case, no good component contains $C_{(u,t)}$ and therefore no vertex of $C_{(u,t)}$ is adjacent to some vertex in a good component and therefore for no T_v^{i-1} containing t_i but not s_i, v is adjacent to some vertex in $C_{(u,t)}$. Since for all $T^{i-1}v$ passing ut_i, v is adjacent to some vertex of $C_{(u,t_i)}$, all these T_v^{i-1} pass $s_i t_i$. In either cases T_v^i passes ut if and only if T^{i-1} passes ut_i. Therefore statement 5 is preserved by edge replacements. Moreover observe that if T_v^i passes $s_i t_c$ then T_v^{i-1} passes $s_i t_i$ and therefore in the neighborhood of $C_{(s_i,tc)} = C_{(s_i,t_i)}$ and statement 5 is preserved in any way.

It remains to show statement 6. Replacing ut_i by us_i takes place only in the case that the subtrees passing ut_i form a subset of the subtrees passing $s_i t_i$, and all subtrees T_v^{i-1} remain subtrees. Note that all subtrees $T^{i-1}v$ containing t_i but not s_i are in some good component c. Since when ut_i is replaced by ut_c the set $C_{(u,t_i)}$ is defined, for all T_v^{i-1} passing ut_i, $v \in c$ or T_v passes $s_i t_i$. The only isolated vertex of T_v^{i-1} that might arise from such a replacement is t_i. But t_i will be deleted, and T_v^i is a tree again. □

It remains to show

Lemma 5 *Let E^i be the set of edges of G^i, i.e. $vw \in E^i$ if and only if T_v^i and T_w^i share a node of T. Then*

1. *$E^{i+1} \subseteq E^i$, for $i = 0, \ldots, k-1$ and*
2. *$E \subseteq E_i$, for $i = 0, \ldots, k$.*

Proof: Note that t_{i+1} is the only node that is in T_i, but not in T_{i+1} and the nodes t_c arising from t_{i+1} are those that appear in T_{i+1} but not in T_i. The first statement follows immediately, because in case that $vw \in E^{i+1}$ then either T_v and T_w share in T_{i+1} a node that is also in T_i or they share a node t_c and therefore the node t_i of T_i.

The second statement can be proved by induction on i. For $i = 0$, the statement is true, by construction. Now suppose $vw \in E$ and T_v and T_w share a node in T_i. If they share a node $\neq t_{i+1}$ then also in T_{i+1}, T_v and T_w share a node. If T_v and T_w share only t_{i+1} then at least one of T_v and T_w does not contain s_{i+1}. If they both do not contain s_{i+1} then v and w are in the same good component c and therefore T_v and T_v share t_c in T_{i+1}. If, for example T_v contains s_{i+1} and T_w does not contain s_{i+1} then w belongs to a good component of C_{i+1} and v is adjacent to some vertex (this is w) of the good component c, w belongs to. Therefore also in this case, T_v and T_w share t_c in T_{i+1}. □

The complexity of this algorithm can be checked as follows. We show that the algorithm works, for each i, in $O(n + m)$ time and therefore the overall time bound is $O(nm)$.

The set C_i can be computed in $O(n + m)$ time, because connected components can be computed in the same time bound.

The good components can be computed in $O(n)$ time. We have a list L_i of those vertices v, such that $t_i \in T_v^{i-1}$. For all these vertices v, we mark the $c \in C_i$ it belongs to as good if v belongs to such a c.

The creation of t_c, for each c, can be done in $O(n)$ time.

The connected component $c \in C_i$ that contains $C_{(u,t_i)}$ can be computed, for all u in $O(n)$ time by picking a vertex $x \in C_{(u,t_i)}$ and determining the $c \in C_i$ x belongs to. The edge replacements can be done in the same time bound.

The update procedure for the T_v's can be done in $O(n+m)$ time. First one has to compute in $O(n)$ time the set of all $Ti - 1_v$ passing s_it_i, by initially labelling all vertices v with 0, then labelling all vertices v with $t_i \in T_v^{i-1}$ by 1 and then labelling all 1-labelled vertices v with $s_i \in T_v^{i-1}$ with 2. If $T_v = T_v^{i-1}$ passes s_it_i and $vw \in E$ then one has to check whether w is in a good component (in one step), and if it is in a good component c then one has to add c to T_v. If T_v does not contain s_i but contains t_i (i.e. is 1-labelled) then one has to determine the good component c its belongs to and to add t_c to T_v.

4.1 An Example

We consider the vertex numbered graph as shown in figure 5. The fat edges are the original edges of the graph. The thin edges are the fill-in edges. Also the clique tree of the fill-in graph is also shown. Each node of the clique tree is assigned with the vertices that are contained in the corresponding clique. The edges of the clique tree are to down numbered from $e1$ to $e5$.

Starting with the split of $e1$, we get one good component c that contains exactly vertex with the number 11. The neighbors of 11 are 8 and 9. This leeds to the following tree representation as shown in figure 6.

The chordal graph represented by this tree representation is shown in the same figure. Here we have exactly one good component and no bad component. In so far, the tree itself does not change. Only the clique corresponding to the parent node of $e1$ changes.

Next we split $e2$ and we have two good components. The one consists of 8 and 11, the other consists of the vertex with number 7. This leeds to a tree representation and a fill-in as

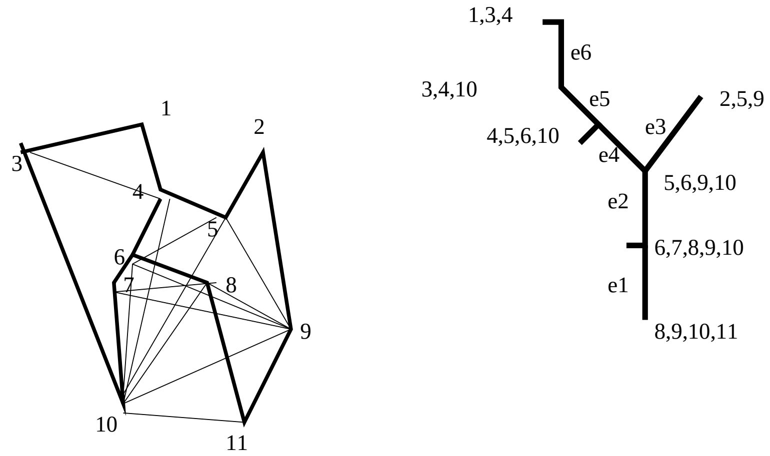

Figure 5: A graph with fill-in edges and the corresponding clique tree

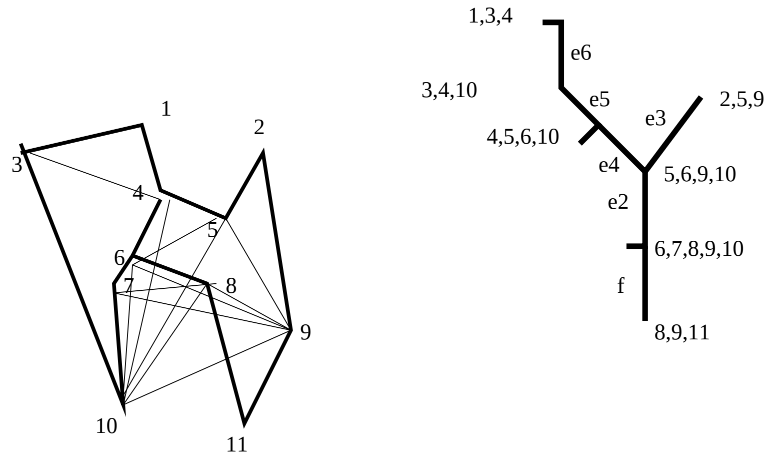

Figure 6: Splitting $e1$

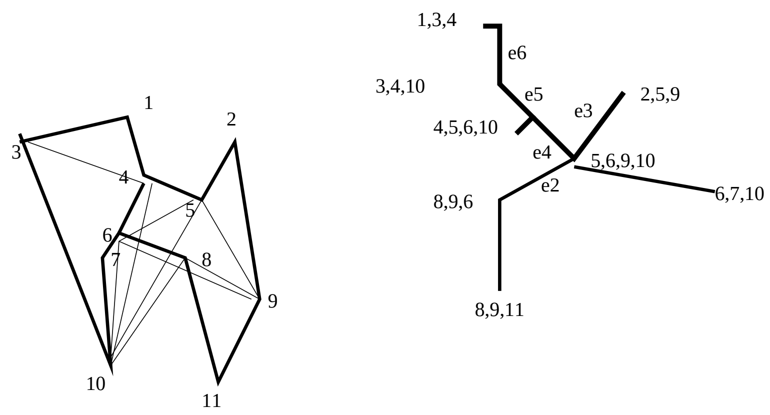

Figure 7: The second tree splitting step

shown in figure 7.

It is left to the reader to verify that further steps of the tree splitting procedure do not change the tree representation, i.e. we now have a quasi-minimal tree representation.

5 The Improved RTL-Algorithm

It remains to eliminate superfluous edges that appear in only one maximal clique, i.e. edges uv, such that T_u and T_v share only a node, but not an edge of $T = T_k$. Here we apply a variation of the algorithm of Rose, Tarjan, and Lueker [15], also called the RTL-algorithm.

The RTL-algorithm works as follows.

Initialize: We start with one list $L_1 := V$;

For $i = n, \ldots, 1$**:**
1. Select a vertex v_i from the nonempty list L_j of the largest index and remove v_i from L_j;
2. for each j and each $y \in L_j$, let $v_i y$ be an edge in E' iff $v_i y \in E$ or y and v_i are neighbors of a connected component C of $G[\bigcup_{\mu<j} L_\mu]$;
3. split each L_j into a list of smaller index containing the non neighbors of v_i with respect to E' and a list of larger index containing the neighbors of v_i with respect to E'; renumber the new lists L_i.

Note that in the last section, we have computed a tree representation, such that all vw such that T_v and T_v have at least two nodes in common then they appear in any chordal extension G'' of G that is a subgraph of the chordal graph G_1 that is represented by the quasi-minimal tree representation as computed in the last section.

We select a root r of the tree T representing G_1. Let $t_1, \ldots, t_k$ be an enumeration of the nodes of T, such that if t_j is the parent of t_i then $i < j$. Such an ordering is called a *bottom up ordering*. Such an ordering can be computed in linear time, for example by postorder.

Let L_i be the list of vertices with $root(T_v) = t_i$. We apply the RTL-algorithm with the only difference that we do not start with one list $L_1 = V$, but with the lists $L_i = \{v | root(T_v) = t_i\}$.

To verify the correctness of the improved RTL-algorithm, one shows that the algorithm does the same as if we would apply the original RTL-algorithm to each graph $G_t = (V_t, E_t)$ where V_t consists of those v with $t \in T_v$ and $vw \in E_t$ if v and w are in V_t and $vw \in E$ or $T_v \cap T_w$ contains at least two nodes of T, i.e. there is an edge of T incident with t that is passed by T_v and T_w.

Lemma 6 *Suppose v is numbered, i.e. v becomes v_i in the improved RTL-algorithm, $w \in L_j$ is not yet numbered, and $v, w \in V_t$. Then vw becomes an edge in E' (i.e. v and w are adjacent in G or are both adjacent to a common connected component of $G[\bigcup_{j'<j} L_{j'}]$) if and only if vw is an edge in E_t or v and w are adjacent to a common connected component of $G_t[\bigcup_{j'<j} L_{j'}]$.*

Proof: Since w is not numbered, $root(T_v)$ is an ancestor of $root(T_w)$ (this includes also equality). If $t \neq root(T_w)$ then $vw \in E_t$, because T_v and T_w share t and $root(T_w)$. Therefore T_v and T_w pass the edge $t\,parent(t)$ of T, and since $(T, T_v)_{v \in V}$ is a quasi minimal tree representation, v and w are adjacent to a connected subset of vertices u, such that all root of T_u are descendents of t, and therefore all these u are in $L_{j'}$, $j' < j$. Therefore vw becomes an edge in E'.

Now assume that $t = root(T_w)$. First suppose there is a path p from v to w in G, such that all inner vertices u are in $L_{j'}$, $j' < j$. Note that if $u_1u_2 \in E$ then T_{u_1} and T_{u_2} share a node of T. Therefore the roots $root(T_u)$ of all inner vertices u of p are descendents of t (equality is possible. let p' be a subpath of p, such that, for the end vertices v', w', $root(T_{v'}) = root(T_{w'}) = t$, and for the inner vertices u, $root(T_u)$ is a proper ancestor of t. Then there is a child t' of t, such that for all these u, T_u is an ancestor of t' (equality is included). Therefore $T_{v'}$ and $T_{w'}$ share the nodes t and t' and therefore $v'w' \in E_t$. Replacing all these subpaths p' by edges in E_t, we get a path q from v to w in E_t with all inner vertices in $L_{j'}$, $j' < j$.

No we assume there is a path q from v to w in E_t, such that all inner vertices u are in $L_{j'}$, $j' < j$. note that all these vertices u are in V_t, and for all these u, $root(T_u) = t$ (not a proper ancestor of t). Suppose v' and w' are consecutive vertices of q. Since $v'w' \in E_t$ either $v'w' \in E$ or there is another node t' that is contained in $T_{v'}$ and $T_{w'}$. t' must be a descendent of t and can be chosen as a child of t. Since $(T, T_v)_{v \in V}$ is quasi minimal, there is a connected subset of u with $root(T_u)$ descendent of t' that is adjacent to v' and w'. Therefore there is a path from v' to w' in G, say p' with inner vertices in $L_{j'}$, $j' < j$. Concatenating all these paths p', we get a path from v to w in G with inner vertices in $L_{j'}$, $j' < j$. □

As a consequence, the improved RTL-algorithm computes, for each G_t, a minimal elimination ordering. The fill-in edges that are created by the improved RTL-algorithm are therefore the edges vw, such that T_v and T_w share more than one node, and the fill-in edges of the graphs G_t.

Corollary 1 *The improved RTL-algorithm computes a minimal elimination ordering $<'$, such that the fill-in graph $G'_{<'}$ is a subset of the graph G_1 that is represented by the quasi-minimal tree representation $(T, T_v)_{v \in V}$ and therefore a subset of the original fill-in graph $G'_{<}$.*

The complexity of the original RTL-algorithm and the improved RTL-algorithm are the same. Therefore we get the following final result.

Theorem 2 *Relative Minimal Elimination Ordering can be solved in $O(nm)$ time.*

6 Conclusions

We developed a sequential algorithm to compute a minimal elimination ordering, such that the fill-in graph is inside a given greater chordal graph. The time bound is $O(nm)$. A better time bound is not to expect, because the minimal elimination ordering problem without the restriction of a larger chordal graph has a time bound of $O(nm)$. Using union find as in finding compact tree representations, the tree splitting procedure might be speeded up a little bit. This is more a practical aspect. One does not get a lower time bound in the order. Another aspect that might be discussed is the parallelization. The components of the tree split procedure are $O(n)$ computations of connected components and reorganization of the tree. First can be parallelized very easily [16]. The parallelization of the second component of the tree split procedure might be a topic for a masters or honors thesis. The improved RTL-algorithm might be replaced by a variation of the algorithm of [8].

References

[1] A. Agrawal, P. Klein, R. Ravi, Cutting Down on Fill-in Using Nested Dissection, in *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, A. George, J. Gilbert, J.W.-H. Liu ed., IMA Volumes in Mathematics and its Applications, Vol. 56, Springer Verlag, 1993, pp. 31-55.

[2] J. Blair, P. Heggernes, J.A. Telle, *Making an Arbitrary Filled Graph Minimal by Removing Fill Edges*, Algorithm Theory-SWAT96, R. Karlsson, A. Lingas ed., LLNCS 1097, pp. 173-184.

[3] P. Bunemann, *A Characterization of Rigid Circuit Graphs*, Discrete Mathematics 9 (1974), pp. 205-212.

[4] E. Dahlhaus, *Fast parallel algorithm for the single link heuristics of hierarchical clustering*, Proceedings of the fourth IEEE Symposium on Parallel and Distributed Processing (1992), pp. 184-186.

[5] E. Dahlhaus, *Efficient Parallel Algorithms on Chordal Graphs with a Sparse Tree Representation*, Proceedings of the 27-th Annual Hawaii International Conference on System Sciences, Vol. II (1994), pp. 150-158.

[6] E. Dahlhaus, *Minimal Elimination Ordering inside a Given Chordal Graph*, Graph-Theoretic Concepts in Computer Science, LLNCS 1335 (1997), pp. 132-143.

[7] Elias Dahlhaus, *Sequential and Parallel Algorithms on Compactly Represented Chordal and Strongly Chordal Graphs*, STACS 97, R. Reischuk, M. Morvan ed., LLNCS 1200 (1997), pp. 487-498.

[8] Elias Dahlhaus, Marek Karpinski, *An Efficient Parallel Algorithm for the Minimal Elimination Ordering (MEO) of an Arbitrary Graph*, Theoretical Computer Science 134 (1994), pp. 493-528.

[9] M. Farber, *Characterizations of Strongly Chordal Graphs*, Discrete Mathematics 43 (1983), pp. 173-189.

[10] F. Gavril, *The Intersection Graphs of Subtrees in Trees Are Exactly the Chordal Graphs*, Journal of Combinatorial Theory Series B, vol. 16(1974), pp. 47-56.

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

[12] J. Gilbert, H. Hafsteinsson, *Parallel Solution of Sparse Linear Systems*, SWAT 88 (1988), LNCS 318, pp. 145-153.

[13] Parra, A., Scheffler, P., *How to use minimal separators for its chordal triangulation*, Proceedings of the 20^{th} International Symposium on Automata, Languages and Programming (ICALP'95), Springer-Verlag Lecture Notes in Computer Science **944**, (1995), pp. 123–134.

[14] D. Rose, *Triangulated Graphs and the Elimination Process*, Journal of Mathematical Analysis and Applications 32 (1970), pp. 597-609.

[15] D. Rose, R. Tarjan, G. Lueker, *Algorithmic Aspects on Vertex Elimination on Graphs*, SIAM Journal on Computing 5 (1976), pp. 266-283.

[16] Y. Shiloach, U. Vishkin, *An O(log n) Parallel Connectivity Algorithm*, Journal of Algorithms 3 (1982), pp. 57-67.

[17] R. Tarjan, *Efficiency of a Good but not Linear Set Union Algorithm*, Journal of the ACM 22 (1975), pp. 215-225.

[18] R. Tarjan, M. Yannakakis, *Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs*, SIAM Journal on Computing 13 (1984), pp. 566-579.
Addendum: SIAM Journal on Computing 14 (1985), pp. 254-255.

[19] M. Yannakakis, Computing the Minimum Fill-in is NP-complete, *SIAM Journal on Algebraic and Discrete Methods* 2 (1981), pp. 77-79.