Angewandte Mathematik und
Informatik
Universität zu Köln

Report No. 354

**Minimal Elimination Ordering for Graphs of Bounded Degree**

by

Elias Dahlhaus

1999

Partially supported by ESPRIT Long Term Research Project Nr. 20244 (ALCOM-IT).

Institut für Informatik, Universität zu Köln,
Pohligstrasse 1, 50969 Köln, Germany

**Abstract**

We show that for graphs of bounded degree, a subset minimal elimination ordering can be determined in almost linear time.

# 1 Introduction

One of the major problems in computational linear algebra is that of sparse Gauss elimination along the diagonal of a positive definite matrix. It is well known that it translates into the following graph theoretic problem [11].

**Minimum Elimination Ordering:** For an ordering $<$ on the vertices, we consider the fill-in graph $G'_< = (V, E')$ of $G = (V, E)$. $G'_<$ contains first the edges in $E$ and secondly two vertices $x$ and $y$ form an edge in $G'_<$ if they have a common smaller neighbor in $G'_<$. *The problem of Minimum Elimination ordering is, given a graph $G = (V, E)$, find an ordering $<$, such that $G'_<$ has a minimum number of fill-in edges.* Note that this problem is NP-complete [13].

For that reason, we relativize the problem.

**Minimal Elimination Ordering:** *Given a graph $G$, find an ordering $<$, such that the edge set of $G'_<$ is minimal with respect to inclusion.* This problem can be solved in $O(nm)$ time [12].

Last problem could be solved for planar graphs in linear time [3].

Another approach is the *nested dissection* [1]. It is known that this problem can be solved more efficiently than for graphs in general for planar graphs and for bounded degree graphs [9].

In so far, it is reasonable to develop also a minimal elimination ordering algorithm that runs for bounded degree graphs more efficiently than for graphs in general. We show the following.

**Theorem 1** *For bounded degree graphs, a minimal elimination ordering can be determined in $O(n)\alpha(n)$ time. Here $\alpha(n)$ is the inverse Ackemann function.*

The strategy of the algorithm is as follows.

1. We divide the vertex set $V$ of the graph into levels $V_1, \ldots, V_k$ of bounded size, such that we there is a minimal elimination ordering, such that with $x \in V_i$, $y \in V_j$, and $i < j$, we have $x < y$. This partition of the vertex set is also called a *bounded approximation* of a minimal elimination ordering. This is done in section 3

2. In section 4, we determine an *elimination tree* of the partition $V_1, \ldots, V_k$ with the nodes $V_1, \ldots, V_k$ in the same way as one determines an elimination tree of the fill-in graph of an ordering of the vertices (see for example [8, 4]).

3. In section 6, we determine for each vertex $v$ of $G$ a subtree $T_v$ of the elimination tree. To keep a space bound of $O(n)$, we will use a compact tree representation.

4. The elimination tree gives one enough information how to refine the levels $V_i$. This is done in section 7.

# 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 $x$.

The *degree* of a vertex $v$ is the size of its neighborhood and is denoted by $d(v)$. The maximum degree of a graph $G$ is denoted by $\Delta$.

The number of vertices of a graph $G$ is denoted by $n$, and the number of its edges is denoted by $m$.

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 [6].

Moreover, chordal graphs $G = (V, E)$ are exactly the intersection graphs of subtrees of a tree [7, 2], 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* $G$. Whenever the trees $T_v$ are represented by their leaves, we call $(T, T_v)_{v \in V}$ a *compact tree representation* of $G$. Note that it is sufficient to mention the leaves of $T_v$, because with $s$ and $t$ in $T_v$, all vertices of $T$ on the unique path from $s$ to $t$ in $T$ are also in $T_v$.

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 the number of edges.

# 3 First Approximation

**Theorem 2** *Let $T$ be a spanning tree of $G$ and $v_1, \ldots, v_n$ be a postorder enumeration of $T$. Let $V_i$ be the set of vertices that have $v_i$ in the closed neighborhood*

*but no $v_j$ with $j > i$. Then there is a minimal elimination ordering $<$, such that with $x \in V_i$, $y \in V_j$, and $i < j$, $x < y$.*

We call such an ordering also *compatible* with $V_1, \ldots, V_k$.

*Proof of Theorem*: Let $C$ be a connected component of $V_1 \cup \ldots \cup V_{i-1}$. Consider the neighborhood $N(C)$ of $C$ in $V_i \cup \ldots \cup V_n$. Note that every vertex in $N(C)$ has a neighbor in the connected vertex set $\{v_i, \ldots, v_n\}$, but no vertex in $N(C)$ is a vertex $v_i, \ldots, v_n$. Therefore $N(C)$ is a minimal set that separates $C$ and the connected set $\{v_i, \ldots, v_n\}$, i.e. $N(C)$ is a cut. Note that the sets $N(C)$ do not cross, i.e. $N(C_1)$ intersects at most one connected component of $G - N(C_2)$. By a result of [10], one gets a minimal fill-in such that all these sets $N(C)$ are cuts. We always can choose $v_k$ as the vertex of maximum number of the minimal fill-in ordering $<$. Then all vertices in $C$ have a smaller number than the vertices in $N(C)$. That means also that if $x \in V_i$, $y \in V_j$, $i < j$, and $xy$ is an edge of $G$ or a fill-in edge then $x < y$. We therefore can transform the ordering $<$ into an ordering $<'$, such that with $x \in V_i$, $y \in V_j$, and $i < j$, we have $x <' y$. Therefore there is a minimal elimination ordering with this fill-in that meets the requirement of the theorem.
Q.E.D.

Note that the ordered partition $(V_1, \ldots, V_k)$ can be determined in $O(n+m) = O(n)$ time.

# 4 Further Level Refinement and Construction of the Elimination Tree

Using union-find, we determine the connected components of $\bigcup_{j \leq i} V_j$, for all $i$. Vertices $x$ and $y$ in $V_i$ are called *equivalent* if they belong to the same connected component of $\bigcup_{j \leq i} V_j$.

The *elimination tree* of $G$ is constructed as follows. The nodes of the elimination tree $T$ are the equivalence classes as defined in previous section. The parent of an equivalence class $C$ of $V_i$ is the equivalence class $C'$ of $V_j$, such that $j > i$, $C'$ is adjacent to the connected component $D$ of $\bigcup_{j \leq i}$ that contains $C$, and $j$ is minimal.

**Lemma 1** *The equivalence classes and the elimination tree can be determined in $O(n\alpha(n))$ time.*

*Proof*: We first determine the sets $D_i$ of connected components of $G[V_i]$. This can be done in linear time. The set of equivalence classes of $V_1$ is just the set $D_1$ of connected components of $V_1$. To determine the equivalence classes of $V_i$, we determine the connected components of $V_1 \cup \ldots \cup V_i$ using union-find. The initial set $Comp$ of components is $D_1$. For each $D \in Comp$, let $i_D$ be the largest $i$, such that $D$ intersects $V_i$. Initially, for each $D \in Comp$, we have

$i_D = 1$. For each $C \in Comp$, let $Eq(C) = C \cap V_{i_C}$ if this set is an equivalence class. Initially, for each $D \in D_1$, $Eq(D) := D$.

Starting with $i = 2$, we consider each $D \in D_i$. For each edge $xy$ with $x \in D$ and $y \in \bigcup_{j<i} V_j$, we find the largest $C \in Comp$ containing $y$ by the find operation. We create the union $D' = C \cup D$. We set $i_{D'} := i$ and if $i_D < i$, we set a pointer $p(C) := x$. The parent of $Eq(C)$ should be the equivalence class that contains $x$. After we considered all $D \in D_i$, we find, for each $x \in V_i$, the largest $C_x \in Comp$ that contains $x$ and put $x$ into $Eq(C_x)$. If $p(C) = x$, for some $C \in Comp$ then we set $Parent(Eq(C)) := Eq(C_x)$ and erase the pointer $p(C) = x$.

This procedure computes the elimination tree and has the same time bound as union-find. The time bound is $O(n\alpha(n))$.
Q.E.D.

# 5 Pre-Fill-in of Equivalence Classes

We determine, for each equivalence class $C$, the set $E_C$ of edges that are in any fill-in of an ordering that meets is compatible with $V_1, \ldots, V_n$. They are just those edges $xy$ joining two vertices in $C \subseteq V_i$ that are in $E$ or adjacent to a particular connected component of $V_1 \cup \ldots \cup V_{i-1}$.

Let $v \in C \subseteq V_i$ and $C$ be an equivalence class. Let $vw \in E$ and $w \in C' \subseteq V_j$ and $j < i$. Then $C'$ is a descendent of $C$. Let $C''$ be the child of $C$ that is an ancestor of $C'$. We say $v\, adj\, C''$ ($v$ is adjacent with $C''$).

**Remark 1** *The number of children of $C$ that are adjacent with $v$ is bounded by $\Delta$.*

**Lemma 2** *For all vertices $v \in C$, the set of children of $C$ that are adjacent with $v$ can be determined in $O(n)$ time.*

*Proof*: We can determine a preorder enumeration $C_1, \ldots, C_k$ of the elimination tree in linear time. We therefore get a sorting of the edges joining $C$ with some descendent $C'$ of $C$ with respect to the number of $C'$ together with the children of $C$ in linear time. For each edge $vw$ joining $v \in C$ with some descendant $C'$ of $C$, we determine the last child $C''$ coming before $C'$ in the preorder enumeration, and we get as a result that $v$ is adjacent with $C''$.
Q.E.D.

Let $v$ and $w$ be in $C$. Then $vw \in E_C$ if $vw \in E$ or $v$ and $w$ have a common adjacent child of $C$.

**Lemma 3** *$E_C$ can be determined in $O(\Delta^3)$.*

*Proof*: This is true, because the number of vertices in $C$ is bounded by $\Delta$ and the number of adjacent children of any $v \in C$ is bounded by $\Delta$.
Q.E.D.

# 6 Compact Tree Representations

For each vertex $v$ of $G$, let $C_v$ be the equivalence class containing $v$ and $S_v$ be the set of descendants of $C_v$ containing a neighbor of $v$. $T_v$ is the smallest subtree of the elimination tree $T$ that contains $C_v$ and all equivalence classes in $S_v$. One gets a *compact representation* $Comp_v$ of $T_v$ consisting of $C_v$, all nodes of $S_v$, and all nodes of $T_v$ containing at least two children in $T_v$. Note that the size of $Comp_v$ is in the order of the degree of $v$ in $G$ (and therefore bounded by $\Delta$). Note that $(Comp_v)_{v \in V}$ can be determined in linear time determining $S_v$ and all least common ancestors of consecutive nodes in $S_v$ in the preorder enumeration of $S_v$ (see also [5, 4].

# 7 Final Elimination Ordering Using Modified Lexical Search

It remains to refine each equivalence class $C$. We may assume that $E_C$ is known.

In general, one finds a minimal elimination ordering using lexical breadth-first search as follows [12].

We iteratively refine an ordered partition $L_1, \ldots, L_k$ as follows.

1. We select an unnumbered vertex $v$ of the largest level and number it.
2. We determine the fill-in edges incident with $v$ as follows. A vertex $x \in L_i$ is made adjacent with $v$ if $x$ and $v$ are adjacent to a common connected component of $\bigcup_{j<i} L_j$.
3. Each $L_i$ is split into a greater level of neighbors of $v$ and a smaller level of nonneighbors of $v$.

We call this algorithm due to Rose, Tarjan, and Lueker, also the *RTL-algorithm*.

The key result for further refinement is the following.

**Lemma 4** *Let $v$ belong to an ancestor equivalence class of $C$ and let $x \in C$. Then $v$ and $x$ are adjacent in any fill-in of a compatible ordering if $v$ is adjacent to a vertex $y$ that belongs to an equivalence class $C'$ that is a descendant of an adjacent child of $x$.*

*Proof*: Assume that $x \in C \subset V_i$. Let $x$ be adjacent to the child $C''$ of $C$ and therefore be adjacent to the vertex $z$ belonging to the descendant $C'''$ of $C''$. Let $y$ be defined as in the lemma. $y$ also belongs to an equivalence class $C'$ that is a descendant of $C''$. $C''$ is the set of vertices of some $V_j$, $j < i$ belonging to one connected component $D$ of $V_1 \cup \ldots \cup V_j$. Also $C'$ and $C'''$ belong to this connected component. Therefore $x$ and $v$ are neighbors of the same connected component of $V_1 \cup \ldots \cup V_{i-1}$. Therefore $x$ and $v$ must be joined by an edge of

$G$ or by a fill-in edge.
Q.E.D.

By the last lemma, we may take any vertex $v$ in an ancestor of $C$, consider a vertex $x \in C$ as *adjacent with* $v$ if they are adjacent in $G$ or adjacent with some child component of $C$. We split $C$ into the equivalence classes of neighbors of $v$ and nonneighbors of $v$.

We determine a minimal fill-in of the graphs $G'_C$ as follows. The vertex set $V'_C$ onsists of

1. The vertices of $C$,
2. the vertices $v$ that appear in some ancestor of $C$ and that are adjacent with some vertex in $C$.

The edge set $E'_C$ is defined as follows.

1. The edges in $E_C$ belong to $E'_C$,
2. the vertices in $V'_C$ that do not belong to $C$ form a complete set,
3. $v \in V'_C \setminus C$ and $x \in C$ are joined by an edge in $E'_C$ is $x$ is adjacent with $v$.

We immediately can observe the following.

**Lemma 5** *1. $V'_C$ is the set of all vertices $v$, such that $T_v$ contains $C$.*

*2. $vw$ is an edge in $G'_C$ if $vw$ is an edge of $G$ or $T_v$ and $T_v$ share another equivalence class than $C$.*

*3. The edges of $G'_C$ are necessarily in any fill-in graph of a compatible elimination ordering.*

*4. If $x$ and $y$ are adjacent in some $G'_C$ then they are adjacent in all $G'_C$ they belong to.*

*Proof*: The first statement follows from the definition of the set $T_v$.

The second statement can be checked as follows. If $v$ and $w$ are in $C$ then $vw \in E_C$ if and only if $vw \in E$ or $v$ and $w$ have a common adjacent child $C'$. The second case is equivalent to the fact that $T_v$ and $T_w$ share also $C'$. If $v$ is not in $C$, but $w \in C$ then either $vw \in E$ or there is a child $C'$, such that $v$ and $w$ are adjacent to vertices belonging to descendants of $C'$. The second case is equivalent to the fact that $T_v$ and $T_w$ contains $C'$. If $v$ and $w$ are both not in $C$ then $T_v$ and $T_w$ contain both also the parent of $C$.

The third statement follows directly from the construction of the edge sets $E'_C$.

The fourth statement follows directly from the second statement.
Q.E.D.

**Lemma 6** *If we determine a minimal fill-in, for each $G'_C$ separately, then we get a minimal fill-in of $G$.*

*Proof*: We only have to show that we get a fill-in of the whole graph, i.e. that we get a chordal graph. Let $T_C$ be any tree representation of a minimal fill-in of $G'_C$. Consider any edge $CC'$ of the elimination tree. Note that all vertices $v$, such that $T_v$ passes $CC'$ are pairwise adjacent in $G'_C$ and therefore the corresponding subtrees $T'_v$ in $G'_C$ share a node $t$ of $T_C$. Therefore we can link the edge $CC'$ with $t$. When we do this, for all $C$ and all edges $CC'$, we get a tree representation of the graph $G'$ that comes up if we determine a minimal fill-in, for each $G'_C$ separately. Therefore $G'$ is chordal.
Q.E.D.

To determine a minimal fill-in of $G'_C$, we proceed as in [12]. We consider the vertices not in $C$ as vertices with larger numbers than those in $C$. Since the vertices not in $C$ are pairwise adjacent, we may assume any ordering of the vertices not in $C$. We may assume that the vertices $v$ with $C \notin Comp_v$ are of greater number than those $v$ with $C \in Comp_v$.

In detail, we proceed as follows.

1. We first consider the vertices $v$, such that $C \in T_v$, but $C \notin Comp_v$.

   We call a child $C'$ of $C$ a *pass through* child of $C$ if there is a $T_v$ passing $C'$ and $C$, such that $C$ is not in $Comp_v$ and therefore $C'$ is the only child of $C$ in $T_v$. Such a vertex $v$ is also called a *pass through vertex* of $C$.

   We determine the pass through children of all $C$ as follows.

   - For each node $C$ of the elimination tree, we determine the distance $dist(C)$ from the root.
   - For each $C$, let $min(C)$ be the minimum distance $dist(C')$ of the root of $T_v$ with $C \in Comp_v$.
   - For each node $C$, let $Min(C)$ be the minimum of all $min(C')$, such that $C'$ is a descendant of $C$ or $C' = C$.
   - A child $C'$ of $C$ is a pass through child of $C$ if and only if $Min(C') < dist(C)$.

   This part can be done in $O(n+m) = O(n)$ time.

   Note that if $v_1$ and $v_2$ are pass through vertices associated with the same pass through child of $C$ then $v_1$ and $v_2$ have the same closed neighborhood in $G'_C$. Therefore we can shrink the vertices associated with the same pass through child $C'$ to one vertex $v_{C'}$. After determining the minimal fill-in, we again replace $v_{C'}$ by the the complete set of vertices $v$ associated with the pass through child $C'$, and the resulting fill-in is a minimal fill-in.

   We run now the RTL-algorithm to number the pass through vertices, i.e. the pass through children.

Let $C'_1, \ldots, C'_k$ be an enumeration of the pass through children of $C$. For $i = 1, \ldots, k$, we determine the $i$-refinement of $C$.

Let $x \, adj \, C'_i$ be defined as in section 5.

- Initially the 0-refinement consists of $C$.
- Suppose the $i$-refinement consists of $C_1, \ldots, C_l$. The $i+1$-refinement of $C$ is determined as follows.
  - A vertex $x \in C_j$ is called adjacent with $C'_{i+1}$ if $x \, adj \, C'_{i+1}$ or there is a connected component $D$ of $\bigcup_{j'<j} C_{j'}$ in $E_C$, such that $x$ is adjacent to some vertex in $D$ and there is some $y \in D$, such that $y \, adj \, C'_{i+1}$.
  - We get the $i+1$-refinement splitting each $C_i$ into a greater component of vertices adjacent with $C'_{i+1}$ and a smaller component of vertices not adjacent with $C'_{i+1}$.

The time needed for one $C'_i$ is bounded by $\Delta^2$. The number of all $C'_i$ in the whole graph $G$ is at most $n$. Therefore we get a time bound of $O(n\Delta^2)$.

2. We consider the vertices $v$, such that $C \in Comp_v$ and $C_v \neq C$. *That means we apply the RTL-algorithm and number the vertices of $G'_C$ that are not in $C$ but no pass through vertices.*

   For each vertex $v$, we do the *$v$-refinement* of all $C$ with $C \in Comp_v \setminus \{C_v\}$.

   - For all $C \in Comp_v \setminus \{C_v\}$, we determine the set $Child_v(C)$ of children of $C$ that are ancestors of some $C' \in Comp_v$.
   - For all $x \in C \in Comp_v \setminus \{C_v\}$, we put $x$ into $N_C(v)$ if $xv \in E$ or $x \, adj \, C'$, for some $C' \in Child_v(C)$.
   - Let $C_1, \ldots, C_l$ be the present refinement of $C$. We put $x \in C_j$ into $N'_C(v)$ if $x \in N_C(v)$ or there is a connected component $D$ of $C_1 \cup \ldots \cup C_{j-1}$ in $E_C$, such that $x$ is adjacent to some vertex in $D$ and $D$ contains a vertex in $N_C(v)$.
   - We get the $v$-refinement of $C$ by splitting each $C_j$ into a greater component containing the vertices in $N_C(v)$ and a smaller component containing the vertices not in $N_C(v)$.

   This step can be done in $O(\Delta^3)$ time, for each $v$, and therefore the overall time is linear.

3. The rest refinement of $C$ considers only vertices of $C$ and is done as in [12]. This can obviously done in $O(|C|^3)$ time, for each $C$, and therefore in overall time $O(n\Delta^2) = O(n)$.

*The overall time bound is* $O(n(\Delta^3 + \alpha(n)) = O(n\alpha(n))$ This proves the main result of the paper.

# 8 Conclusions

Note that a minimal elimination fill-in can be far from a minimum fill-in. But the approach of this paper might also be helpful to improve elimination heuristics. For example, the second refinement procedure might be helpful to make an ordered vertex partition a minimal elimination ordering. Future research might go in that direction to improve this algorithm such that the resulting ordering has also good approximation properties.

The time bound that has been proved is $O(n(\Delta^3 + \alpha(n))$. The exponent of $\Delta$ might be lower.

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

[3] E. Dahlhaus, *Minimal Elimination of Planar Graphs*, SWAT'98.

[4] 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.

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

[7] 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.

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

[9] J. Gilbert, R. Tarjan, The Analysis of a Nested Dissection Algorithm, Numerische mathematik 50 (1987), pp. 377-404.

[10] 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.

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

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

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