An Efficient Parallel Algorithm for the Minimal Elimination Ordering (MEO) of an Arbitrary Graph *

Elias Dahlhaus †
Dept. of Computer Science,
University of Bonn
D-5300 Bonn 1
dahlhaus@cs.su.oz.au

Marek Karpinski ‡
Department of Computer Science,
University of Bonn,
D-5300 Bonn 1
and
International Computer Science Institute
Berkeley, California
marek@icsi.berkeley.edu

Abstract. We design the first efficient parallel algorithm for computing the minimal elimination ordering (MEO) of an arbitrary graph.

The algorithm works in $O(\log^3 n)$ parallel time and $O(nm)$ processors on a CREW PRAM, for an n-vertex, m-edge graph, and is optimal up to a polylogarithmic factor with respect to the best sequential algorithm of Rose, Tarjan and Lueker ([RTL 76]).

The MEO problem for arbitrary graphs arises in a number of combinatorial optimization problems, as well as in database applications, scheduling problems, and the

*An Extended Abstract has appeared in [DK 89].

†present address: Basser Department of Computer Science, University of Sydney, NSW 2006, Australia

‡Research partially supported by the Leibniz Center for Research in Computer Science, by the DFG Grant KA 673/4-1, and by the SERC Grant GR-E 68297.

sparse Gaussian elimination on symmetric matrices. It was believed before to be inherently sequential, and strongly resisting sublinear parallel time (sublinear sequential storage) algorithms.

As an application, this paper gives the first efficient parallel solutions to the problem of *minimal fill-in* for arbitrary graphs and connected combinatorial optimization problems (see [RTL 76], [Ta 85], for example), and to the problem of the Gaussian elimination of sparse symmetric matrices ([Ro 70], [Ro 73]). (The problem of computing a *minimum fill-In* is known to be *NP*-complete, [Ya 81].)

The method of solution involves a development of new techniques for solving connected minimal set system problem, and combining it with some new divide-and-conquer methods.

0 Introduction

The theory of elimination orderings is used in a number of combinatorial optimization and database applications, as well as in scheduling and general divide-and-conquer techniques ([Ro 73], [Ta 85]). Elimination orderings also arise in Gaussian elimination on sparse symmetric matrices ([Ro 73], [RTL 76]).

The minimal elimination problem (MEO) for arbitrary graphs (cf. [Ro 73], [RTL 76], [Ta 85], [DK 88a], [No 88]) is the following.

Let $G = (V, E)$ be any graph and $<$ be an ordering on V given as an enumeration of V. Define $E_<$ to be the *chordal extension* of G related to $<$, i.e. the minimal extension E' of E such that if $x < y$, $x < z$ and xy, then $xz \in E'$ implies $yz \in E'$. The set $F_< = E_< \setminus E$ is called the *fill-in* of $<$ [Ta 85].

The problem is to compute, for any given graph $G = (V, E)$, an ordering $<$ on V such that $E_<$ is (inclusion) minimal. We call such an ordering a *minimal elimination ordering* (MEO) of G ([RTL 76], [Ta 85]). An MEO algorithm is an algorithm computing for an arbitrary input graph $G = (V, E)$ an ordering on V such that $E_<$ is (inclusion) minimal.

MEO Algorithm (I/O)

INPUT:	A graph $G = (V, E)$.
OUTPUT:	An ordering $<$ on V such that $E_<$ is inclusion minimal.

$$F_< \;=\; \{\, vw \mid v \neq w,\ vw \notin E,\ \exists\, p \text{ a path } p = v_1 v_2 \ldots v_k \text{ in } G \text{ such that } v_1 = v,\ v_k = w,\ \text{and } v_i < \min\{v, w\} \text{ for } i = 2, \ldots, k-1 \,\}.$$

In the case the ordering $<$ satisfies $E = E_<$, (V, E) is chordal and the ordering $<$ is called a *perfect elimination ordering* (PEO).

It is known that the computation of a minimum (cardinality) chordal extension or minimum cardinality fill-in is *NP*-complete ([Ya 81]). Rose, Tarjan and Lueker have relativized this problem to the computation of an MEO $E_<$ of a given graph. Their sequential algorithm works in $O(nm)$ time and $O(n+m)$ storage ([RTL 76]).

There are efficient parallel algorithms to recognize chordal graphs and to compute the perfect elimination ordering for chordal graphs ([Ed 87], [NNS 87], [DK 86], [DK 87], [Kl 88]).

In this paper we give a parallel solution to the MEO Problem by designing an algorithm computing an MEO for any given graph which works in $O(\log^3 n)$ parallel time and $O(nm)$ processors on a CRCW PRAM.

The MEO algorithm of this paper directly entails recent results on existence of *NC*-algorithms for Clique Separator Decomposition ([DK 88b], [DK 88a], [DKN 89]) and for the *first* time provides a parallel technique of computing the minimal fill-in (cf. [Ta 85]) for arbitrary graphs, and combining our algorithm with the Cholesky factorization algorithm of Gilbert and Hafsteinsson ([GH 88]), an efficient parallel algorithm for the Gaussian elimination on sparse symmetric matrices (cf. [Ro 73]).

The paper is organized as follows.

In Section 1, the notational and fundamental concepts of this paper are introduced. Section 2 describes the global strategy which is a divide-and-conquer strategy.

Section 3 presents the simple case of a graph G being the disjoint union of two cliques C_1 and C_2. In this case the problem is equivalent to the following set system problem:

Given a set V and a set S of subsets of V, compute an ordering $(S_1 < \ldots < S_n)$ of S such that for $i = 1, \ldots, n$, $S_i \setminus \bigcup_{j<i} S_j$ is inclusion minimal in $\{S_k \setminus \bigcup_{j<i} S_j \mid k \geq i\}$.

In Section 4 we complete the algorithm using the special case of Section 3.

1 Basic Concepts and Notations

Throughout the whole paper, graphs are undirected, without loops and multiple edges.

A graph $G = (V, E)$ consists of a *vertex set* V and an *edge set* E. The edge joining x and y is denoted by xy.

Define $N(x) := N_G(x) = \{x\} \cup \{y \mid xy \in E\}$ as the *neighborhood* of the vertex x in G (including x). For $M \subseteq V$, define also $N(M) := N_G(M) = \bigcup_{x \in M} N(x)$.

The subgraph of G induced by a subset V' of the vertex set V of G is denoted by $G|V'$. Generally we call an edge-preserving subgraph an *induced* subgraph.

A *connected* subset of G is a subset V' of its vertex set such that $G|V'$ is connected. An inclusion maximal connected subset is called a *connected component.*

A *spanning tree* of the connected graph $G = (V, E)$ is a tree T with vertex set V and an edge set $E' \subseteq E$.

A *spanning forest* of any graph consists of spanning trees for its connected components.

Given a set A, we define $\#A$ to be the cardinality of A.

The computation models are the concurrent-read concurrent-write parallel random access machine (CRCW PRAM) and the concurrent read exclusive write parallel random access machine (CREW-PRAM) (cf. e. g. [FW 78], [Co 85], [KR 88]). Note that each CREW-PRAM working in T time using P processors is also a CRCW-PRAM working in the same time bounds. Vice versa, a CRCW-PRAM working in T time using P processors can be simulated by a CREW-PRAM in time $O(T \log P)$ using $O(P)$ processors. For example, a CRCW-PRAM, working in $O(\log n)$ time using $O(n + m)$ processors can be simulated by a CREW-PRAM working in $O(\log^2 n)$ time using $O(n + m)$ processors.

We assume that each arithmetic operation needs one time unit and one processor unit.

In general, n is the number of vertices of $G = (V, E)$, and m is the number of edges.

We assume that the reader is familiar with the following results in parallel computation.

Theorem 1 *(i) (see [SV 82])*
The connected components and a spanning tree of any graph can be computed in $O(\log n)$ CRCW-time and $O(n+m)$ processors and therefore in $O(\log^2 n)$ CREW-time using $O(n + m)$ processors.

(ii) (see [Co 86])
n numbers can be sorted in $O(\log n)$ CREW-time and $O(n)$ processors.

Let $T = (V_T, E_T)$ be a tree with a root r. We can define the unique *direction* $\vec{T} = (V_T, A_r)$ or (y, x) of any edge xy of T to the root r. If $(x, y) \in A_r$, then x is a *child* of y and y is the *parent* of x. For each vertex x of T let $\{y_1^x, \ldots, y_{d(x)}^x\}$ be the set of its children. The edge $y_i^x x$ is labelled by i, $l(y_i^x x) = i$.

Let $P_x := (e_1 \ldots e_P)$ be the sequence of the edges of the unique path from r to x (that means $e_1 = ry_1$, $e_P = y_{P-1}x$). Then $l^*(x) := (l(e_1), \ldots, l(e_P))$.

The *preorder* $\prec$ is defined as follows:

For $x, y \in V_T$, $x \prec y$ iff $l^*(x)$ is a subsequence of $l^*(y)$ or $l^*(x)$ is lexicographically smaller than $l^*(y)$.

Theorem 2 (see for example [TV 85]) *A preorder can be computed in $O(\log n)$ CREW-time and $O(n)$ processors. For a tree T and a 'root' $r \in V_T$ an ordering (numbering) such that each initial segment induces a subtree containing r can be computed in $O(\log n)$ CREW-time and by $O(n)$ processors.*

A graph is called *chordal* iff it has no induced cycle of length > 3 (each cycle of length > 3 has an edge joining nonconsecutive vertices).

For a tree T and a collection S of subtrees of T the *vertex intersection graph* of T and $\mathcal{S}$ is defined as follows:

(i) The vertex set is $\mathcal{S}$.

(ii) $S_1, S_2 \in \mathcal{S}$ are joined by an edge iff they have a common vertex of T.

Chordal graphs can be characterized as follows (cf. [Ga74], [Ta 85]):

Theorem 3 *The following statements are equivalent:*

(i) $G = (V, E)$ *is chordal.*

(ii) $G = (V, E)$ *has a* perfect elimination erdering $<$, *i.e. if* $x < y$, $x < z$ *and* xy, $xz \in E$, *then* $yz \in E$.

(iii) $G = (V, E)$ *is the vertex intersection graph ([Ga74], [Bu 74]) of a collection* $\mathcal{S}_G$ *of subtrees of some tree* T. *That means* $\mathcal{S}_G$ *is of the form* $\{S_v : v \in V\}$ *and, for vertices* $v, w \in V$, $vw \in E$ *iff* S_v *and* S_w *have at least one common vertex of* T. *We call* $(T, \mathcal{S}_G)$ *a* subtree representation *of* G.

Remark. It is easily seen that the number of maximal cliques of a chordal graph is bounded by $n = \#V$.

Suppose that G is the vertex intersection graph of the collection $\mathcal{S}_G$ of subtrees of T. For $v \in V$ let S_v be the corresponding subtree in $\mathcal{S}_G$. For $t \in V_G$ let c_t be the set $\{S \in \mathcal{S}_G \mid t \in S\}$. We may assume that the maximal cliques of G are exactly the sets $\hat{c}_t := \{v \mid S_v \in c_t\}$ ([Ga 72],[Bu 74]).

Klein proved the following result ([Kl 88]):

Theorem 4 *There is a parallel algorithm for computing for each chordal graph G a perfect elimination ordering and the* subtree representation *$(T_G, \mathcal{S}_G)$ in time $O(\log^2 n)$ and $O(n+m)$ processors on a CRCW PRAM.*

Consider any ordering $<$ on the vertex set V of the graph $G = (V, E)$. Then the *chordal extension* $E_<$ of G and $<$ is the smallest extension of E such that $<$ is a perfect elimination ordering, i.e. $F_<$ is the smallest set F such that

1. $E \subseteq F$ and
2. if $xy \in F$, $xz \in F$, $x < y$, and $x < z$, then $yz \in F$.

Theorem 5 *[Ya 81] The computation of an ordering $<$ such that its chordal extension is minimal by cardinality, is NP-complete.*

In contrast, Rose, Tarjan, and Lueker proved:

Theorem 6 *For any graph $G = (V, E)$, an ordering $<$ can be computed in sequential time of $O(nm)$ such that $F_<$ is minimal by inclusion.*

Definition 1 *An ordering $<$ on the vertex set V of a graph $G = (V, E)$ is called a* minimal elimination ordering (MEO) *if there is no ordering $<'$ such that $F_{<'} \subsetneq F_<$ ($F_<$ is minimal with respect to inclusion).*

Obviously an MEO is an ordering $<$ such that its chordal extension is an inclusion minimal extension of G to a chordal graph.

An MEO can also be characterized as follows.

Lemma 1 ([RTL 76]) *$<$ is an MEO of (V, E) iff, for all $e \in F_< \setminus E$, $(V, F_< \setminus \{e\})$ has an induced cycle of four vertices (and edges).*

This result is essential in the whole paper.

2 The Global Strategy

We introduce the notion of an *endsegment* of an MEO:

A subset V_0 of the vertex set V of a graph G is called an *endsegment* of an MEO iff there is a minimal elimination ordering $<$ for G and a vertex $v \in V_0$ such that $V_0 = \{w \in V \mid v = w \text{ or } v < w\}$.

First, in a similar way as in the procedure NONE in Klein's perfect elimination ordering (PEO) algorithm ([Kl 88]) , we shall compute an *endsegment* $V_0 \subseteq V$.

1. We add new edges to $G|V_0$ which are necessarily in the fill in of each ordering, having V_0 as endsegment, i.e. those pairs xy of vertices of V_0 which are adjacent to the same connected component of $G|(V \setminus V_0)$. We denote the resulting extension of $G|V_0$ by $G_0 = (V_0, E_0)$. This will be done by the procedure ENDSEGMENT.
2. We consider the connected components $V_1, \ldots, V_k$ of $G|(V \setminus V_0)$. We compute recursively in parallel MEOs and their corresponding chordal extensions for $G_1 = G|V_1, \ldots, G_k = G|V_k$, and for G_0.
3. Using the MEOs for $G_0, \ldots, G_k$ and their corresponding chordal extensions, we compute an MEO for G and its corresponding chordal extension.

The key for the computation of G_0 is the following result:

Theorem 7 *Let $G = (V, E)$ and V_0 be a subset of V. Let V_i be a connected component of $G|(V \setminus V_0)$. Let $x, y \in V_0$, $x', y' \in V_i$, and $xx', yy' \in E$. Then xy belongs to the chordal extension of any ordering $<$, having V_0 as an endsegment.*

Proof: We use the following result in [Ta 85]:

Lemma 2 *Given an arbitrary graph $G = (V, E)$ and an ordering $<$ on V, the Fill-In $F_<$ of G under ordering $<$ is the set of edges defined as follows (cf. [Ta 85]):*

$$\begin{aligned} F_< \;=\; \{\, vw \mid v \neq w,\ vw \notin E,\ &\exists p \text{ a path } p = v_1 v_2 \ldots v_k \text{ in } G \text{ such that} \\ &v_1 = v,\ v_k = w,\ \text{and } v_i < \min\{v, w\} \text{ for } i = 2, \ldots, k-1 \,\}. \end{aligned}$$

Consider x and y, x' and y', and V_0 as stated in the theorem. Then there is a path from x' to y' in G, using only vertices in $V \setminus V_0$. Consider any ordering $<$, having V_0 as an endsegment. Then all these vertices of the path are smaller than x and y. Since xx' and

yy' are in E, we can add x at the beginning and y at the end of this path. All vertices different from x and y are smaller than x and y. Therefore there is a path from x to y such that all vertices different from x and y are smaller than x and y.

We determine G_0 from V_0 as follows:

1. For any connected component V_i of $G|(V \setminus V_0)$, we determine the neighborhood of V_i in V_0 $(N(V_i) \cap V_0)$.
2. We set $xy \in E_0$ iff x and y are in V_0 and $xy \in E$ or $x, y \in N(V_i)$ for some connected component V_i of $G|(V \setminus V_0)$, i.e. $N(V_i) \cap V_0$ is made complete.

A remark to step 2): To guarantee a logarithmic recursion depth, G_0 has to be complete or the size of V_0 may not exceed $2/3\sharp V$. Moreover, all connected components V_i of $G|(V \setminus V_0)$ must have a size of at most $2/3\sharp V$. In the case that G_0 is complete, we can order V_0 in any way.

A remark to step 3): The idea of step 3) is to compute an MEO $<_i$ for each graph $\hat{G}_i$ consisting of the vertex set $\hat{V}_i = V_i \cup (N(V_i) \cap V_0)$ and of the edge set $\hat{E}_i = \{xy | x, y \in \hat{V}_i \text{ and } (xy \in E \text{ or } x, y \in N(V_i)\}$. $N(V_i) \cap V_0$ is an end segment of $<_i$ for each i. This will be done with the help of an MEO for $G_i = G|V_i$. Let $<_0$ be an MEO for G_0. Then $<$ is set to be the concatenation of all $<_i |V_i$ and and $<_0$ at last. To guarantee that $<$ is really an MEO independently which orderings $<_i$ are chosen, we introduce the notion of a *good endsegment*:

Definition 2 *Let V_0 be an endsegment of an MEO and $V_1, \ldots, V_k$ be the connected components of $G|(V \setminus V_0)$. Let $\hat{G}_i = (\hat{V}_i, \hat{E}_i)$ and G_0 be defined as above. Then V_0 is a* good endsegment *iff, for all MEOs $<_i$ for $\hat{G}_i$ with $N(V_i) \cap V_0$ as an end segment and each MEO $<_0$ on G_0, the concatenation of $<_1 |V_1$, ... , $<_k |V_k$, and $<_0$ at last is an MEO.*

In the rest of this section we consider the problem how to compute such a set V_0 . Note that in Klein's PEO-algorithm, the fact is used that the neighborhood of a connected subset of a chordal graph is an endsegment of a PEO. An analogous result for MEOs is the following.

Theorem 8 *Let M be a connected subset of V. Then $N(M)$ is a good endsegment.*

Proof Let $V_0 = N(M)$ and $V_1 \dots V_k$ be the connected components of $V \setminus N(M)$.Let

$$\tilde{C}_i := N(V_i) \cap N(M) \text{ for } i = 1, \dots, k.$$

Let $\hat{V}_i := V_i \cup \tilde{C}_i$ and let $G_0 := (V_0, E_0)$ where E_0 arises from E by completing all $\tilde{C}_i$.
Let $\hat{G}_i := (\hat{V}_i, \hat{E}_i)$ where $\hat{E}_i$ arises from E restricted to $\hat{V}_i$ by completing $\tilde{C}_i$.
Let $(\hat{V}_i, E'_i)$ and (V_0, E'_0) be inclusion minimal chordal extensions of $(\hat{V}_i, \hat{E}_i)$ and (V_0, E_0) resp.

Claim 1 $G' := (V, E'_0 \cup \bigcup_i E'_i)$ *is a minimal chordal extension.*

Proof of Claim 1 First we show that G' is chordal. Consider any cycle C of G'. As long as C is only in one V_i, it has a chord or is of length three, because, by construction, $G'|V_i$ is chordal. Suppose C passes more than one V_i. If C passes V_i and some other V_j, then it must pass $N(V_i) \cap V_0$. Suppose C passes only V_i and $N(V_i) \cap V_0$. Then it has a chord or is of length three, because $G'|(V_i \cup (N(V_i) \cap V_0))$ is chordal by construction. Suppose now that C leaves also $N(V_i) \cap V_0$. Then it must also return via $N(V_i) \cap V_0$. Therefore at least two vertices of C are in $N(V_i) \cap V_0$. These two vertices are a chord in G'.

To prove that G' is a minimal chordal extension of G, we have only to show that each edge $xy \in E'_i \setminus E,\ x, y \in \tilde{C}_i$ forms an induced cycle after its deletion.
Since x, y are adjacent to the same connected component V_i, one finds a path $x, y_1 \dots y_l, y$ such that $y_i \in V_i$. This forms a cycle in G'.
Since G' is chordal, there is a y_i which is adjacent to x and y in G'.
Since $x, y \in N(M)$ and M is connected, one finds a path $xm_1 \dots m_l y$ such that all $m_i \in M$.
Since also $xm_1 \dots y$ forms a cycle and G' is chordal, one finds an m_j which is adjacent to x and y.

By the construction of G' $(v_i\ v_i, x, m_j, y, v_i)$ forms an induced cycle of length four after the deletion of xy.

(Claim)

We complete the proof of theorem 8: Let $<_i$ be MEOs for $\hat{G}_i$ and $<_0$ be an MEO for G_0. Moreover, for each $i = 1, \dots k$ let $N(V_i) \cap V_0$ be an end segment of $<_i$. Since the sets $N(V_i) \cap V_0$ are complete in $\hat{G}_i$, we can order $N(V_i) \cap V_0$ in any way, and each $<_i$ remains an MEO. Let for each $i = 0, \dots, k$ let F_i be the corresponding chordal extension of $<_i$. By the claim, the union E' of all F_i is a minimal chordal extension of G.

Let $<$ be the concatenation of $<_1, \dots, <_k$, and at last $<_0$. Then $<$ is a PEO for $G' = (V, E')$ and therefore an MEO:

Suppose xy and xz are in E' and $x < y$ and $x < z$. If $x \in V_0$, then also y and z are in V_0, and $x <_0 y, z$. If $x \in V_i$, then $y, z \in V_i \cup (N(V_i) \cap V_0)$ Suppose $x < y < z$. If $z \in V_i$, then $y \in V_i$, because V_0 is an endsegment of $<$. Then $x <_i y <_i z$ and $yz \in F_i \subset E'$, because $<_i$ is a PEO for F_i. Suppose $y \in V_i$, but $z \in V_0$. Then still $y <_i z$, because $N(V_i) \cap V_0$ is an endsegment of $<_i$. By the same argument as in the case that y and z are in V_i, $yz \in E'$. Suppose $y \in V_0$. Then also $z \in V_0$, because V_0 is an endsegment of $<$. That means y and z are both in the complete set $N(V_i) \cap V_0$ and therefore joined by an edge in E'.

Last theorem will be used in the "low degree refinement" part of the ENDSEGMENT procedurei, i.e. if there is a large connected subset of low degree vertices.

For the "high degree refinement" part of the ENDSEGMENT procedure, the following result is useful.

Theorem 9 *Let $G = (V, E)$ be any graph and $x \in V$. Let W be a connected component of $G|(V \setminus N(W))$. Then $\{x\} \cup (N(x) \cap N(W))$ is a good endsegment.*

Proof By previous theorem, $N(x)$ is a good endsegment. By theorem 7, $N(x) \cap N(W)$ is complete in any chordal extension of an ordering, having $N(x)$ as an endsegment. Moreover, $N(x) \cap N(W)$ is also complete in any chordal extension of an ordering, having $V_0 = \{x\} \cup (N(x) \cap N(W))$ as an endsegment. Consider the connected components $V_1, \ldots, V_k$ of $G|(V \setminus V_0)$. Then one of the V_i is W. As in the previous theorem we fix MEOs $<_i$ on the graphs $\hat{G}_i$ which arise from $G|(V_i \cup (N(V_i) \cap V_0))$ by making $V_0 \cap N(V_i)$ complete. As in the previous theorem, we also concatenate all $<_i |V_i$ and at last any ordering $<_0$ on V_0 to an ordering $<$. Now V_0 is made complete only by making $V_0 \cap N(W) = N(x) \cap N(W)$ complete. When we restrict $<$ to $V_0 \cup W$, V_0 is a good endsegment of $G|(V_0 \cup W)$, because it is the neighborhood of x. Since all other V_i have neighbors only in V_0 and V_0 is made complete by W alone, the removal of any fill-in edge in V_0, i.e. of any edge that appears in the chordal extension but not in the original graph, induces a chordless cycle of length four. All other fill-in edges not in E, are fill-in edges of some $\hat{G}_i$ which are also nonedges in $\hat{G}_i$.

Now we are able to compute a set V_0 in parallel which is a good endsegment.

We compute the set D_1 of sparse vertices and the set D_2 of "dense" vertices. Here "sparse" means that the degree is at most $2/3$ of the number n of vertices. A vertex is "dense" iff it is not sparse.

In the case that there are two nonadjacent dense vertices x and y, their common neighborhood $N(x) \cap N(y)$ is at least $1/3$ of the number of vertices of the whole graph

$G = (V, E)$. Let W be the connected component of $G|(V \setminus N(x))$, y belongs to. Then V_0 is set to be $\{x\} \cup (N(x) \cap N(W))$.

Now we consider the case that the set of dense vertices is complete. Trivially the set D_2 of dense vertices can be taken as an endsegment of an MEO. In the case that all connected components of $G|_{V \setminus D_2}$ have a cardinality of at most $2/3$ of the number of vertices, we are done and set $V_0 = D_2$. Otherwise we consider the connected component C_1 of sparse vertices whose size is greater than $2/3$ of the number of vertices of the whole graph. We can compute on this connected component C_1 a spanning tree T_1. As a root we choose a sparse vertex r of maximal degree. If the degree of r is between $1/3$ and $2/3$ of the number of vertices of the whole graph, then we are done, since we only have to take the neighborhood of r as an endsegment. This is a good endsegment by Theorem 8.

It remains the case that the degrees of all sparse vertices of C_1 are less than $1/3$ of the number of vertices. But then we can compute an enumeration $(v_1, \ldots, v_p)$ of the sparse connected component C_1 such that each initial segment $(v_1, \ldots, v_l)$ is a subtree of the above spanning tree T_1. Since all neighborhoods $N(v_i)$ are less than $1/3$ of the number of vertices and the size of C_1 is greater than $2/3$ of the number of vertices, we find an initial segment $\{v_1, \ldots, v_l\}$ such that the size of its neighborhood lies between $1/3$ and $2/3$ of the number of vertices of the whole graph. It also is a good endsegment by Theorem 8.

The computation of a good end segment satisfying above requirements consists of the computation of connected components and spanning trees, neighborhoods of initial segments, and of common neighborhoods. Therefore we get the same time and processor bound as in the procedure NONE of the perfect elimination algorithm of Klein [Kl].

Theorem 10 *For any graph $G = (V, E)$, we can compute a good endsegment V_0, such that $\#V_0 \leq \frac{2}{3}\#V$ or V_0 is complete in each chordal extension of an ordering, having V_0 as an end segment, and, for each connected component C of $V \setminus V_0$, $\#C \leq \frac{2}{3}\#V$, in CREW-time $O(\log^2 n)$ and $O(n + m) \leq O(n^2)$ processors.*

How to make the neighborhood of each connected component of $G|_{V \setminus V_0}$ a complete subgraph, will be discussed in a later section.

3 A Simple Case

We assume in this Section that the vertex set V of $G = (V, E)$ is the disjoint union of two complete subsets V' and W' with additional edges between V' and W'. This appears as an essential subprocedure of the general case.

For $v \in W'$, let $N'(v) := N(v) \cap V'$ be the set of neighbors of the vertex v' which are in V'.

For the development of an MEO-algorithm for the simple case, the following structural result is useful.

Lemma 3 *G is chordal iff for $v_1, v_2 \in W'$, $N'(v_1)$ and $N'(v_2)$ are comparable with respect to inclusion (compare also [NNS 87]).*

Proof

"$\Rightarrow$": Suppose $w_1 \in N'(v_1) \setminus N'(v_2)$ and $w_2 \in N'(v_2) \setminus N'(v_1)$.
Then $v_1 v_2 w_2 w_1$ forms a chordless cycle of length four.

"$\Leftarrow$": We assume that G is not chordal.
Then a chordless cycle must be of the form $v_1 v_2 w_1 w_2$ such that $v_1, v_2 \in W'$ and $w_1, w_2 \in V'$. Longer chordless cycles are not possible. But then $N'(v_1)$ and $N'(v_2)$ are not comparable by inclusion. This is a contradiction.

For the case that G is not chordal, we compute an MEO $<$ with V' as an endsegment. Since V' is complete, we can V' order in any way. It remains the problem how to order W'. We compute an enumeration $(u_i)_i$ of W', whose corresponding ordering is the restriction of a minimal elimination ordering to W'. The chordal extension E' defined by $(u_i)_i$ is

$$E' = E \cup \{u_i v : v \in V' \text{ and there is an } j \leq i, u_j v \in E\}$$

.

Define $N''(u_i) = \{v \in V' : u_i v \in E'\} = \bigcup_{j \leq i} N'(u_j)$ and $G' := (V, E')$. $N''(u_i)$ is also called the *extended neighborhood* of u_i.

Clearly, by Lemma 3, G' is a chordal extension of G. Our aim is to compute a minimal chordal extension G', that means we would like to compute an enumeration $(u_i)_i$ such that the following minimality condition which we call *Property M* (minimality property) is satisfied:

If $u_i v \in E' \setminus E$ then there is a u_j with $j < i$ and a $w \in V'$ such that $u_j v \in E$, $u_i w \in E$, and $u_j w \notin E'$.

It is easily seen that the deletion of the edge $u_i v$ induces a 4-cycle $u_i w v u_j$.

We also can describe the minimality property in terms of neighborhoods.

If $v \in N''(u_i)$ but $v \notin N'(u_i)$, then there is a u_j, $j < i$, and a $w \in N(u_i)$ such that $v \in N'(u_j)$ and $w \in N'(u_i) \setminus N''(u_j)$.

We shall show that an enumeration $(u_i)_i$ satisfying the Property M ever exists. The following result proves the existence of such an enumeration and gives also a hint how to compute it.

Lemma 4 *If, for each i, $N'(u_i) \setminus \bigcup_{j<i} N'(u_j)$ is inclusion minimal in*

$$\{N'(u_k) \setminus \bigcup_{j<i} N'(u_j) \mid k \geq i\}$$

then $(u_i)_i$ satisfies the Property M.

Proof. Consider any $v \in N''(u_l) \setminus N'(u_l)$. Then $l \neq 1$. Let i be the minimum such that $v \in N'(u_i)$.

Since $N'(u_i) \setminus \bigcup_{j<i} N'(u_j)$ is minimal for

$$\{N'(u_k) \setminus \bigcup_{j<i} N'(u_j)$$

with respect to inclusion, and $N'(u_{i+1}) \setminus \bigcup_{j<i} N'(u_j) \neq N'(u_l) \setminus \bigcup_{j<i} N'(u_j)$ (they differ by v), there is a $w \in N'(u_l) \setminus N''(u_i)$ which is not in $N''(u_{i+1})$.

To compute a sequence $(u_i)_i$ satisfying the assumption of Lemma 3 is clearly equivalent to the following computation problem on set systems:

Given a set system (family of sets) $\xi \subset P(V)$, compute an enumeration A_i of ξ which satisfies the following *Property I* (inclusion property):

$$I : A_{i+1} \setminus \bigcup_{j \leq i} A_j \text{ is inclusion minimal in } \{A_k \setminus \bigcup_{j \leq i} A_j \,:\, k > i\}.$$

Theorem 11 *Under the assumption that ξ is presented as the bipartite graph consisting of $V \cup \xi$ as the vertex set and the membership relation as the edge set with n vertices and m edges, an enumeration satisfying the Property I can be computed in CREW-time $O(\log^2 n)$ by $O(n+m)$ processors.*

Proof We shall state a recursive divide-and-conquer algorithm computing an enumeration satisfying Property I:

Here we divide the sets in ξ into small and large sets, that means sets A_i with a size smaller than $1/3$ of the size of the ground set V and sets A_i with a size at least $1/3$ of the size of the ground set. Our aim is to divide the problem to smaller ground sets V' and V''. V' is the union of some small sets in ξ. If the small ground sets cover at least $1/3$ of the whole ground set V then we can take V' as the union of some small sets such that the size of V' is between $1/3$ and $2/3$ of the size of V. The V'' is taken to be the complement of V'. Clearly the size of V'' also lies between $1/3$ and $2/3$ of the size of V. We divide the sets in ξ by the following way:

If $A_i \in \xi$ is a subset of V' then it belongs to the part of V'.
Otherwise $A_i' := A_i \setminus V'$ is taken to the part of V''.

We continue recursively the procedure to V' and all $A_i \subseteq V'$ and to V'' and all A_i' such that $A_i \not\subset V'$. We concatenate the sequence of all A_i belonging to V' and afterwards the sequence of A_i belonging to V''.

In the case that the small sets A_i cover less than $1/3$ of V, we also take V' as the union of all small sets in ξ. But we cannot define V'' as in the case before. To make V'' not too large, we choose a large A_i, say A' such that $A_i \setminus V'$ is minimal. Let V'' be the complement of $A' \cup V'$.

We recursively apply the procedure to V' and all subsets A_i of V' and to V'' and all $A_i' := A_i \setminus A' \setminus V' = A_i \cap V''$. Here we first concatenate the sequence of A_i belonging to V'. Afterwards we take the one $A_i = A'$, and then we take the sequence of A_i belonging to V'' (that means A_i is not A' and is not a subset of V').

Formally we proceed as in Algorithm 2.

The correctness of Algorithm 2 can be shown as follows:

Let J_1 be the set of i such that $A_i \subseteq V'$, J_2 be the set of i such that $A_i \setminus V' = V'' \setminus V'$, and J_3 be the set of remaining $i \in I$ (as defined in the procedure Property I).

Let $< i_1 \ldots, i_k >$ be an enumeration of J_1 satisfying the property I and $< j_1, \ldots, j_{k'} >$ be an enumeration of J_3 such that $(B_{j_i})i = 1^{k'}$ satisfies the property I.

Since all A_i with $i \notin J_1$ are no subsets of V', $A_{i_l} \setminus \bigcup_{j=1}^{l-1} A_{i_j}$ is an inclusion minimal set of all $A_j \setminus \bigcup_{j=1}^{l} A_{i_j}$ with $j \in I \setminus \{i_1, \ldots, i_{l-1}\}$. For $i \in J_2$, the minimality conditions are preserved by construction and the fact that $V' = \bigcup_{i \in J_1} A_i$.

Let $P_3 =< j_1 \ldots j_{k'} >$ defined as above. Then $B_{j_i} \setminus \bigcup_{l<i} B_{j_l} = A_{j_i} \setminus (\bigcup_{l \leq i} A_{j_l} \cup \bigcup_{j \in J_1 \cup J_2} A_j)$.

PROCEDURE Property I ($\{A_i \mid i \in I\}$, V, P)
Input Parameter: A family $\{A_i : i \in I\}$ of subsets of V
Output Parameter: Sequence $P := (i_1 \ldots i_I)$ which enumerates I
BEGIN

1) Let $I_1 := \{i \mid \#A_i < 1/3\,\#V\}$, $\quad I_2 := \{i \mid \#A_i \geq 1/3\,\#V\}$;

a) If $\#\bigcup_{i\in I_1} A_i \geq 2/3\,\#V$, then select $I_1' \subset I_1$ such that

$$1/3\,\#V \leq \#\bigcup_{i\in I_1'} A_i \leq 2/3\,\#V\,;\; V' := \bigcup_{i\in I_1'} A_i\,;\; V'' := V \setminus V'.$$

b) If $\#\bigcup_{i\in I_1} A_i \in [1/3\,\#V, 2/3\,\#V]$ then

$$V' := \bigcup_{i\in I_1} A_i\,;\; V'' := V \setminus V'.$$

c) If $\#\bigcup_{i\in I_1} A_i < 1/3\,\#V$ then

$$V' := \bigcup_{i\in I_1} A_i.$$

Let A be an $i_2 \in I_2$ such that $\#A_i \setminus V'$ is minimal; let $V'' := V \setminus V' \setminus A$.

2) Let
$J_1 := \{i : A_i \subset V'\}$;
$J_2 := \{i \in I \setminus J_1 : A_i \cup \bigcup_{i\in I_1} A_i = V \setminus V''\}$;
$J_3 := I \setminus (J_1 \cup J_2)$;
$\{B_i : i \in J_1\} := \{A_i : i \in J_1\}$;
$\{B_i : i \in J_3\} := \{A_i \setminus V'' : i \in J_3\}$;

3) *Property* I $(\{B_i : i \in J_1\}, V', P_1)$ (if $J_1 > 1$)
Property I $(\{B_i : i \in J_3\}, W', P_1)$ (if $J_3 > 1$)
Let P_2 be any injective sequence enumerating J_2.
$P := P_1 \frown P_2 \frown P_3$ is the concatenation of P_1, P_2 and P_3;

END.

Algorithm 2

This completes the correctness proof.

The computation of all $\#A_i$ needs $O(\log n)$ CREW-time and $O(n+m)$ processors. The same is true for the computation of I_1 and I_2. Therefore the preface of 1) can be executed in $O(\log n)$ CREW-time by $O(n+m)$ processors.

The selection of I_1' as in 1a) can be done as follows:

Sort I_1' with respect to $\#A_i$ in decreasing order $I_1' := \{i_1 \dots i_p\}$;

Compute, for each v, the least j, say $j(v)$ such that $v \in A_{i_j}$ and let

$$s_j := \#\{v : j(v) = j\}; \;\; S_j := \#(A_{i_j} \setminus \bigcup_{j'<j} A_{i_j})).$$

Compute by bisection a k such that $\sum_{j \le k} S_j \in [\frac{1}{3}\#V, \frac{2}{3}\#V]$ (this exists, since for each j, $S_j < \frac{1}{3}\#V$).

It is easily seen that each step needs at most $O(\log n)$ CREW-time and $O(n+m)$ processors. Therefore 1a) can be executed in $O(\log n)$ CREW-time by $O(n+m)$ processors.

Since $\bigcup_{i \in I_1} A_i$ can be computed in constant CRCW-time by $O(n+m)$ processors, 1b) and the first part of 1c) can be computed in constant CRCW-time by $O(n+m)$ processors. $\#A_{i_2} \setminus V'$ can be computed in $O(\log n)$ CREW-time and by $O(n+m)$ processors. Therefore 1c) can be executed in $O(\log n)$ CREW-time and $O(n+m)$ processors.

Since V' is a fixed set, the it can be checked for all i simultaneously in constant CRCW-time with $O(n+_m)$ processors, whether $A_i \subseteq V'$. By the same arguments as in the computation of V', J_2, J_3, $\{B_i : i \in J_1\}$, $\{B_i : i \in J_3\}$ can be computed in constant CRCW-time and $O(n+m)$ processors.

Therefore part 1) and part 2) of the procedure Property I have a processor bound of $O(n+m)$ and a time bound of $O(\log n)$ on a CREW-PRAM.

Since V' and V'' are constructed such that $\#V' \le \frac{2}{3}\#V$ and $V' \setminus V'' \le \frac{2}{3}\#V$ the recursion depth as in 3) is $O(\log n)$. Therefore the whole procedure needs $O(\log^2 n)$ CREW-time and $O(n+m)$ processors. Note that J_1 and J_3 are disjoint, therefore the processor number needed for the procedure Property I is the sum of the processor numbers $O(n_1+m_1)$ and $O(n_2+m_2)$ needed for
Property I $(\{B_i : i \in J_1\}, V', P_1)$,
Property I $(\{B_i : i \in J_3\}, V' \setminus V'', P_3)$, respectively.

But $n_1 + n_2 \le n$ and $m_1 + m_2 \le m$, since $J_1 \cap J_2 = \emptyset$ and $V' \cap (V \setminus V'') = \emptyset$.

The procedure SIMPLE CASE is nothing else than the computation of an enumeration $(u_i)_{i=1}^k$ of W' such that $(N'(u_i))_{i=1}^k$ satisfies property I.

4 The General Case

In Section 2 we computed a good endsegment $V_0 \subset V$ of an MEO. For each connected component V_i of $G|_{V \setminus V_0}$, we made $N(V_i) \cap V_0$ a complete subgraph. We defined E_0 as the set of edges in V_0, which are in E or arise as edges after $N(V_i) \cap V_0$ has been made a complete subgraph. We defined $G_0 = (V_0, E_0)$ and $G_i = (V_i, E|V_i)$, for $i = 1, \ldots, k$.

Now MEO is applied recursively to all G_i with $i = 0, \ldots, k$. We get orderings $<_0$ and $<'_1, \ldots, <'_k$ of V_0 and $V_1, \ldots, V_k$ respectively. We denote the chordal extensions of G_i with respect to $<_0$ and $<'_i$, $i = 1, \ldots k$ by $\tilde{G}_0, \ldots, \tilde{G}_k$.

The goal is to find perfect elimination orderings $<_1, \ldots, <_k$, such that the concatenation $<$ of $<_1, \ldots, <_k$ and at last $<_0$ is an MEO.

It is sufficient to find perfect elimination orderings $<_i$ such that any extension of $<_i$ to an ordering $<''_i$ on $V_i \cup (N(V_i) \cap V_0)$, with $N(V_i) \cap V_0$ as an end segment, is an MEO of the graph $(V_i \cup (N(V_i) \cap V_0), E|(V_i \cup (N(V_i) \cap V_0)) \cup \{xy | x, y \in N(V_i) \cap V_0\})$. Then since V_0 is a good endsegment, $<$ is an MEO of G.

Lemma 5 *The chordal extension $F_<$ of $<$ consists of the edges in $\tilde{G}_0, \ldots, \tilde{G}_k$ and of additional edges between G_i and G_0.*

That means, all fill-in edges inside V_i are just edges of $\tilde{G}_i$ and there are no fill-in edges joining any $x \in V_i$ and $y \in V_j$ with $i, j \geq 1$ and $i \neq j$.

Proof of lemma. Suppose $x < y$ and $xy \in F_<$. Then $xy \in E$ or there is a path P in G from x to y, such that all internal vertices are smaller than x.

Suppose $x \in V_0$. Suppose x' and y' are vertices of P such that all vertices between x' and y' are not in V_0, but $x', y' \in V_0$. Then $x'y' \in E_0$, because x' and y' are adjacent to the same connected component. Therefore we can replace P by a path P_0 in G_0 such that all internal vertices are smaller thatn x. Since $<_0 = <|V_0$ is a perfect elimination ordering of $\tilde{G}_0$, xy is an edge of $\tilde{G}_0$.

Suppose $x \notin V_0$ and $y \in V_0$. Then we are done.

Suppose $x \in V_i$ and $y \in V_j$. Then $i = j$, because, for all $v < x$ and therefore for all internal vertices v of P, $v \notin V_0$ and every path in G joining vertices in different V_i must

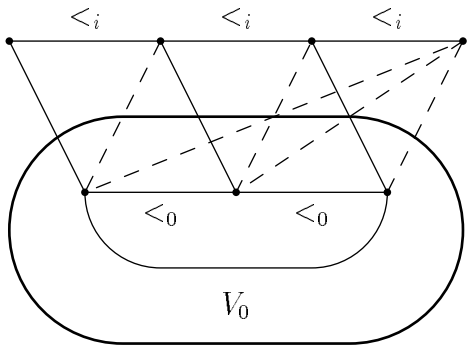

Figure 1

pass V_0. By the same argument, also all internal vertices of P are in V_i. Therefore for all internal vertices v of P, $v <_i x$. Since $<_i$ is a perfect elimination ordering on $\tilde{G}_i$, xy is an edge in $\tilde{G}_i$.

As shown by the following counterexample in figure 1, we cannot take each perfect elimination ordering on $\tilde{G}_i$. Edges of the original graph are denoted by continuous lines. Fill-in edges are assigned by broken lines.

To get a better feeling on the structure of perfect elimination orderings and the possibility to find the right perfect elimination ordering of $\tilde{G}_i$, we introduce the notion of a *cut*. For vertices v, w of a graph G a *v-w-cut* is an inclusion minimal v and w separating set of vertices. A *cut* of G is a v-w-cut of some two vertices v and w of G.

For chordal graphs we know the following about cuts [Di 76]:

Theorem 12 *Each cut c of a chordal graph $G = (V, E)$ is complete. Moreover, it is the intersection of two maximal cliques (and therefore the intersection of the neighborhood of two nonadjacent vertices in different connected components of $G|_{V \setminus c}$).*

We also introduce the notion of a *saturated* connected component of $G|(V \setminus c)$ of a cut c of G: A connected component D of $G|(V \setminus c)$ is a saturated connected component of the cut c iff each $x \in c$ is adjacent to some $y \in D$.

Corollary 1 *Each cut has at least two saturated connected components.*

Proof. Suppose c is an $x-y$-cut. Let D_1 be the connected component of $G|(V \setminus c)$, x belongs to and D_2 be the connected component of $G|(V \setminus c)$, y belongs to. Then D_1 and D_2 are both saturated connected components of c: If $z \in c$ is not adjacent to a vertex of D_2 or not adjacent to a vertex of D_2 then $c \setminus \{z\}$ still separates x and y. That is a contradiction to the minimality condition.

Lemma 6 *Each cut of a chordal graph $G = (V, E)$ with a perfect elimination ordering $<$ is of the form*

$$c_x = \{y : x < y \wedge xy \in E\}.$$

Proof We use the fact that each cut c is the intersection of the neighborhoods of two nonadjacent vertices x and y of G. Consider the subgraph of G induced by $c \cup \{x, y\}$. Then one of the vertices x and y is the smallest in $c \cup \{x, y\}$ with respect to the perfect elimination ordering $<$, since only x and y have a simplicial neighborhood in $c \cup \{x, y\}$. We may assume that x is the smallest element. We also may assume that x and y are in different connected components of $G|_{V \setminus c}$. Therefore for at least one of these connected components D and all $z \in D$ adjacent to all vertices of c, $z < w$, for all $w \in c$. From this connected component we choose a largest z adjacent to all vertices of c. But then the complete set $\{y : zy \in E \wedge z < y\}$ is exactly c.

Corollary The number of cuts of a chordal graph is bounded by the number n of its vertices.

Theorem 13 (Klein) *The cuts of any chordal graph G can be computed by a CRCW in $O(\log^2 n)$ time by $O(n + m)$ processors. Moreover, if a perfect elimination ordering of G is known then we get a CREW-time bound of $O(\log n)$.*

We return to the computation of $<_i$.

We compute the set Cut_i of all cuts of the chordal extension $\tilde{G}_i$ of G_i. This can be done immediately, for each V_i, by $O(\#V_i + \#\tilde{E})$ processors and $O(\log n)$ time by a CREW-PRAM ([Kl 88]). Therefore the overall complexity of computing all cuts of any $\tilde{G}_i$ consists of a CREW-time bound of $O(\log n)$ and a processor bound of $O(n^2)$. This bound remains true also in each recursion step of the MEO-algorithm.

We begin with an overview of the procedure Minchord that computes, for each i, the right perfect elimination ordering of $\tilde{G}_i$. Note that for any perfect elimination ordering $<_i$ of $\tilde{G}_i$ and any cut c of G_i, for all but one connected components D of $\tilde{G}_i|_{V_i \setminus c}$, all vertices in D are smaller than all vertices in c. By some criteria, we find out that connected component D that does not satisfy above requirement with respect to the still unknown perfect elimination ordering that satisfies the minimal elimination ordering requirements. We shall call this connected component the dominator of c and all other connected compontents of $\tilde{G}_i|_{V_i \setminus c}$ are called *non dominating*. We replace any edge xy such that x appears in a non dominating connected component of some cut that contains y by a directed edge $x \rightarrow_i y$. This partial orientation can be seen as a first approximation of the required perfect elimination ordering $<_i$ of $\tilde{G}_i$. The next step is to compute a pre-fill-in, i.e. for $x \in V_0$ and $y \in V_i$, xy is a pre-fill-in edge iff there is a y' such that $xy' \in E$ and (y', y) is in the transitive closure $\rightarrow_i^*$ of $\rightarrow_i$. We shall find out that those edges that remain undirected define an equivalence relation on the vertices, i.e. they form a disjoint union of cliques. For each such clique, we apply the SIMPLE CASE procedure.

To get an algorithm with a processor bound of $O(n^3)$ and a time bound of $O(\log^3 n)$ is quiet straightforward. The difficulty is to get a processor bound of $O(nm)$ in each recursion step of the MEO procedure.

4.1 How to find out the dominator of a cut

For each $c \in Cut_i$ and each saturated connected component D of $\tilde{G}_i|V_i \setminus c$, we compute the number $\text{num}(D) := \#(N_G(D) \cap V_0)$, the cardinality of the neighborhood of D in V_0 with respect to the original graph G.

The *dominator* of $c \in Cut_i$ is the only saturated connected component of $\tilde{G}|(V_i \setminus c)$ such that $\text{num}(D)$ is maximal. If there is more than one such saturated component then it is the unique connected component D such that $\text{num}(D)$ and $\#D$ are maximal, if such a unique connected component exists. Otherwise c has no dominator.

A saturated connected component D is a *nondominating* connected component of c iff D is not the dominator of c. We denote the set of nondominating connected components of any cut c of $\tilde{G}_i$ by ND_i.

We give an example: Figure 7 shows a graph G with the complete set V_0 as a good endsegment. Moreover, here G is constructed in such a way that $G|_{V \setminus V_0}$ is connected and chordal. In this example $V_i = V \setminus V_0$, $G_i = \tilde{G}_i$, and $G_0 = G|V_0$.

The cuts of $\tilde{G}_i = G|_{V \setminus V_0}$ are $c_1, \ldots, c_3$ as shown in figure 8.

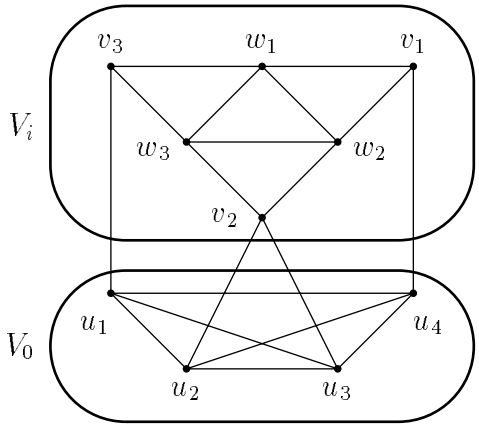

Figure 7

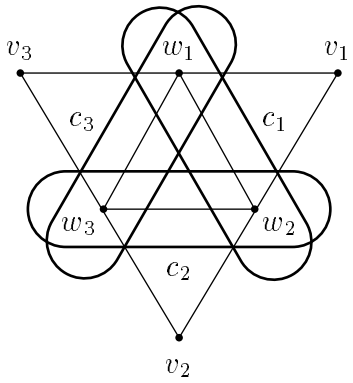

Figure 8

The dominating connecting component of c_1 is $D := \{v_2, v_3, w_3\}$, it has the greatest number of neighbors in V_0.

The dominating connected component of c_3 is $\{v_2, w_2, v_1\}$, for the same reasons.

The connected components of $\tilde{G}_i | V_i \setminus c$ both have the same maximal number of neighbors in V_0.

Since $\{v_1, v_3, w_1\}$ has the greatest own cardinality, it is the dominating connected component belonging to c_2.

4.2 The orientation of edges and the clique structure of non oriented edges

To determine $<_i$, we introduce the following relation $\rightarrow_i$.

If xy is in $\tilde{E}_i$ (the edge set of $\tilde{G}_i$), y is in $c \in Cut_i$, and x is in a nondominating saturated connected component of $G_i | (V_i \setminus c)$ then we set $x \rightarrow_i y$.

The following result justifies that the transitive closure $\rightarrow_i^*$ of $\rightarrow_i$ can be interpreted as a first approximation of a minimal elimination ordering.

Theorem 14 *(i) $\rightarrow_i$ is cycle free.*

(ii) If $x \rightarrow_i y$ and $x \rightarrow_i z$ then $yz \in \tilde{E}_i$.

(iii) If $x \rightarrow_i y$, $y \rightarrow_i z$ and $xz \in \tilde{E}_i$, then $x \rightarrow_i z$.

(iv) Let $\equiv_i$ be the following relation: $x \equiv_i y$ iff $xy \in \tilde{E}_i$ and not $x \rightarrow_i y$ or $y \rightarrow_i x$. Then $\equiv_i$ is an equivalence relation.

Proof Let $x \rightarrow_i y$ and $x \rightarrow_i z$. Let c_1 and c_2 be cuts such that $y \in c_1$, $z \in c_2$ and such that x is in a nondominating saturated connected component of c_1 and in a nondominating saturated connected component of c_2.

Claim 2 *At least one of the vertices y, z is in $c_1 \cap c_2$.*

Proof of the Claim Assume that $y \in c_1 \setminus c_2$ and $z \in c_2 \setminus c_1$. Let D_1 be a saturated connected component of $\tilde{G}|(V_i \setminus c_1)$ not containing x and D_2 be a saturated connected component of $G|(V_i \setminus c_2)$ not containing x. Then D_1 and D_2 are disjoint, and moreover, $D_1 \cap c_2 = \emptyset$ and $D_2 \cap c_1 = \emptyset$. Otherwise y is a neighbor of a vertex of D_2 or z is a neighbor of D_1, and therefore $x \in D_2$ or $x \in D_1$.

Note that $D_2 \cup \{x\}$ is contained in a saturated connected component of c_1 and vice versa.

Suppose $x \rightarrow_i y$ and $x \rightarrow_i z$. Then we find saturated connected components D_1 and D_2 of c_1 and c_2 respectively which do not contain x with the additional property that $\#(N_G(D_1) \cap V_0)$ and $\#(N_G(D_2) \cap V_0)$ are maximal. Therefore

$$\#(N_G(D_1) \cap V_0) \geq \#(N_G(D_2 \cup \{x\}) \cap V_0) \geq \#N_G((D_2) \cap V_0).$$

But also the other direction of the inequation is true by the same argument and thus the equality. Moreover, one finds such D_1, D_2 such that $\#D_1 \geq \#D_2 \cup \{x\}$ and $\#D_2 \geq \#D_1 \cup \{x\}$. That means $\#D_1 \geq \#D_2 + 1$ and $\#D_2 \geq \#D_1 + 1$. This is a contradiction. (Claim)

Since cuts are complete, (ii) follows from Claim 2.

Now we consider the case $x \rightarrow_i y$, $y \rightarrow_i z$: Let c_1 be a cut such that $y \in c_1$ and x is in a nondominating saturated component of $\tilde{G}_i|(V_i \setminus c_1)$ and let c_2 be a cut such that $z \in c_2$ and y is in a nondominating saturated component of $\tilde{G}_i|_{V_i \setminus c_2}$.

Claim 3 *Each path from x to some vertex of c_2 must pass a vertex of c_1.*

Proof of the Claim We assume that the Claim is not true. Let D_1 be a saturated connected component of $\tilde{G}_i|(V_i \setminus c_1$ not containing x and D_2 be a saturated connected component of $\tilde{G}_i|(V_i \setminus c_2)$ not containing y. Since we assume that there is a path from x to c_2 not passing c_1, all vertices of $c_2 \setminus c_1$ and x are in the same (saturated) connected component of $\tilde{G}_i|_{V_i \setminus c_1}$. Moreover, also D_2 is in this saturated connected component. Since $y \in c_1 \setminus c_2$, D_1 and y are in one saturated connected component of $\tilde{G}_i|(V \setminus c_2)$.

If we assume again that D_1 and D_2 satisfy the maximality conditions with respect to $\#(N_G(D_i) \cap V_0)$ and their own cardinality we get the same contradiction as in Claim 2. (Claim)

By Claim 3, $zy \in c_1 \cap c_2$ if $xz \in \tilde{E}_i$. Therefore also $x \rightarrow_i z$, (iii) has been proved.

We are now able to prove (i): Consider any chain $x_1 \rightarrow_i x_2 \rightarrow_i x_3 \cdots x_k$. Then we find cuts $c_2, \ldots, c_k$ such that for all $i \leq k : x_i \in c_i$, $x_{i-1} \notin x_i$ and all paths from x_i to

c_{i+2} pass c_{i+1}. Therefore also $x_i \notin c_{i+2}$. Otherwise c_{i+2} is reachable from x_i not passing c_{i+1} (by a path of length 0).

By induction one can prove that $x_i \notin c_{i+k}$, for any $k > 0$: For $k = 1, 2$ this is just shown. Assume $x_i \in c_{i+k+1}$. Then there is a path $x_{i+k-1}x_{i+k-2}\ldots x_i$ from x_{i+k-1} to c_{i+k+1} not passing c_{i+k}. This is a contradiction.

Therefore it is impossible that $x_1 = x_k$. Therefore the cycle freeness of $\rightarrow_i$ has been proved.

To prove (iv) we proceed as follows: If xy, $yz \in \tilde{E}_i$ but $xz \notin \tilde{E}_i$ then there is a cut c between x and z. y is in this cut c. x and y must be in different saturated connected components of $\tilde{G}_i|_{(V_i \setminus c)}$. But only one connected component can be a dominator.

Therefore $x \rightarrow_i y$ or $z \rightarrow_i y$ is satisfied. (*)

If $xy \in \tilde{E}_i$, $yz \in \tilde{E}_i$ and $x \rightarrow_i z$, then $x \rightarrow_i y$ or $y \rightarrow_i z$. (**)

Let c be a cut such that $z \in c$ and x is in a nondominating saturated connected component of $\tilde{G}_i|(V_i \setminus c)$. Then in the case that $y \notin c$, y is also in the same nondominating connected component as x, and therefore $y \geq z$.

If $y \in c$ we have $x \rightarrow_i y$. By (*) and (**), $\equiv_i$ is an equivalence relation.

Moreover, we get the following extended result.

Theorem 15 *$x \rightarrow_i y$, $z \equiv_i x$ implies $z \rightarrow_i y$.*

Proof If $x \equiv_i z$, then $xz \in E$. Since $x \rightarrow_i y$, $zy \in E$. Otherwise $z \rightarrow_i y$, by (*). Therefore, by Theorem 14, $z \rightarrow_i y$.

We determine the right $<_i$ is as follows:

Let $\rightarrow_i^*$ be the transitive closure of $\rightarrow_i$. Then for $x \in V_i$ and $y \in V_0$, we set

$xy \in E_i'$ iff there is an $x' \rightarrow_i^*$ such that $x'y \in E$.

E_i' is also called the *pre-fill-in* of $\rightarrow_i$.

For each $\equiv_i$-equivalence class A, we apply the procedure SIMPLE CASE to the subgraph G_A, consisting of the complete sets A (of $\tilde{G}_i$) and $N(V_i) \cap V_0$ (of G_0) and of the edges of E_i' which join each a vertex of A and a vertex of $N(V_i) \cap V_0$. We denote

the resulting ordering on A by $<_A$ and the resulting chordal extension by $\hat{E}_A$. $\hat{E}_i$ is the union of all $\hat{E}_A$ such that A is an equivalence class of $\equiv_i$.

$<_i$ is a total ordering on V_i which exceeds $\rightarrow_i^*$ and, for each $\equiv_i$-equivalence class A, the ordering $<_A$.

Let $<$ be the concatenation of all $<_i$ and of $<_0$ at last.

Lemma 7 *1. $<_i$ is a perfect elimination ordering of $\tilde{G}_i$.*

2. The chordal extension $F_<$ of $<$ is $\tilde{E}_0 \cup \tilde{E}_1 \cup \ldots \cup \tilde{E}_k \cup \bigcup_{i=1}^k \hat{E}_i$.

Proof. The first statement is proved as follows.

Let $x <_i y$ and $x <_i z$. If $x \rightarrow_i y$ and $x \rightarrow_i z$, then, by theorem 14, $yz \in \tilde{E}_i$.

Suppose $x <_A y$ and $y \rightarrow_i z$. Then $x \equiv_i y$, and therefore $y \rightarrow_i z$, by theorem 15.

If $x <_A y$ and $x <_A z$ then, by $y \equiv_i x \equiv z$, $y \equiv_i z$, and therefore $yz \in \tilde{E}_i$.

The second statement of this lemma follows immediately from the first statement of this lemma, by lemma 5.

4.3 The MEO-Property of $<_i$

Theorem 16 *$\tilde{G} := (V, \tilde{E})$ is a minimal chordal extension of $G = (V, E)$.*

Proof Since $\tilde{G}|V_i = \tilde{G}_i$ and $\tilde{G}|V_0$ are minimal chordal extensions of G_i and of G_0 it remains to prove that, in $\tilde{G}$, the deletion of any edge between V_0 and V_i not being in E forces a cycle of length four.

To check the minimality of $\tilde{G}$ as a chordal extension, we proceed as follows:

Let $yv \in \tilde{E} \setminus E$, $y \in V_i$, and $v \in V_0$. Then one of the following two statements is true.

1. There is an x' such that x' and y are in the same $\equiv_i$-equivalence class A, $x' <_A y$, and $x'v \in E_i'$. Since $<_A$ is an MEO of the graph G_A consisting of the complete sets $V_0 \cap N(V_i$ and A and the edges of E_i' between A and $N(V_i) \cap V_0$, $yv \in E_i'$ or the deletion of $y'v$ causes a chordless cycle of length four.

2. $yv \in E_i' \setminus E$. Then there is an $x'' \rightarrow_i^* y$, such that $x''v \in E$. Therefore we find an $x' \rightarrow_i y$, such that $x'v \in E_i'$.

It remains to consider the second case.

Then there is a cut c such that $y \in c$ and x' is in a saturated component of $\tilde{G}_i|(V_i \setminus c)$.

But then there is also a saturated component D' of c not containing x' and $\#(N(D) \cap V_0) \leq \#(N(D') \cap V_0))$. Therefore one finds a vertex $v' \in N(D') \cap V_0$ not being in $N(D) \cap V_0$ or $v \in N(D') \cap V_0 = N(D) \cap V_0$.

In the next paragraphs v is any vertex of $N(D') \cap V_0$ and v' is any vertex in $N(D') \cap V_0 \setminus N(D)$ if $N(D') \cap V_0 \setminus N(D) \neq \emptyset$ and $v' = v$ if $N(D') \cap V_0 = N(D) \cap V_0$.

Let $\tilde{x} \in D'$ and $\tilde{x}v' \in E$. Let $\tilde{x} = x_1 x_2 \ldots x_k x_{k+1} = y$ be a shortest path in $\tilde{G}_i$ joining $\tilde{x}$ and y such that all x_i are in D'. Such a path exists since D' is a saturated connected component.

First we consider the case that D' is not the dominator of c. Then $x_k \rightarrow_i y$.

Moreover, $x_{k-1} \rightarrow_i x_k, \cdots x_0 \rightarrow x_1$ and therefore $\tilde{x} \rightarrow_i^* x_k \rightarrow_i y$:

This can be proved by backward induction. Since $x_{j-1}x_j \in \tilde{E}_i$, $x_{j-1} \equiv_i x_j$ or $x_j \rightarrow_i x_{j-1}$ or $x_{j-1} \rightarrow_i x_j$. We know that $x_j \rightarrow x_{j+1}$ and that $x_{j-1}x_{j+1} \notin \tilde{E}_i$ (we chose a shortest path). Then, by theorem 15, $x_j \equiv x_{j-1}$ is excluded, and, by theorem 14, $x_j \rightarrow_i x_{j-1}$ is excluded.

For the case that $v' \in N(D') \setminus N(D)$ we get a cycle $(yv'vx'y)$ after deletion of the edge yv (see figure 3, continuous lines are edges, dashed lines fill-in edges. The cycle without an arrow shows a cycle of length four.)

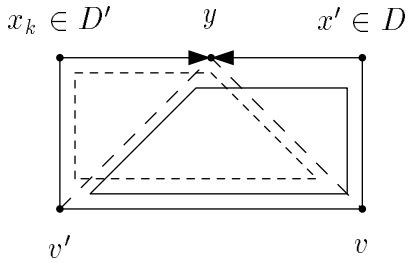

Figure 3

For the case that $v' = v$ we get a cycle $(x_k y x' v x_k)$ (see figure 3a, dashed lines between points denote edges of the original graph. The non-dashed cycle arises from deleting the edge zv. The dashed cycle arises from the deletion of yv'.)

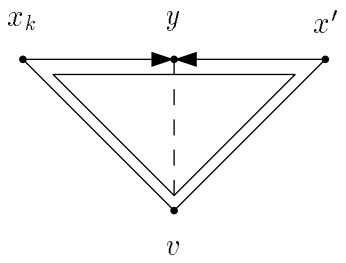

Figure 3a

Now we consider the case that D' is the dominator. Again we consider the shortest path $\tilde{x} = x_1 \ldots x_k y$.

If $y \rightarrow_i x_k$ then also $x_k v \in E_i'$, and we get the cycle $x_k y x' v x_k$ after the deletion of yv (see figure 4).

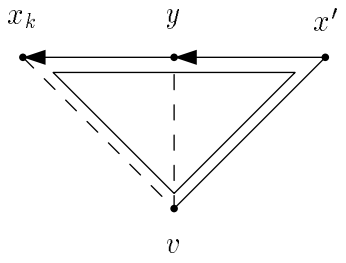

Figure 4

If not $y \rightarrow_i x_k$ then $x_k \rightarrow y$ or $x_k \equiv_i y$. In the first case we can proceed as in the case that D' is not the dominator.

Also in the second case, $x_{k-1} \rightarrow_i x_k$, because $x_{k-1} y \notin \tilde{E}_i$, $\equiv_i$ equivalence classes form complete sets, and $x_k \rightarrow x_{k-1}$ cannot be the case, by theorem 15. Therefore $x_j \rightarrow_i x_{j+1}$, for all $j < k$, by the same argument as in the case that D' is not the dominator.

Therefore $x_k v' \in E_i'$, since $x \rightarrow_i^* x_k$. If $v' = v$ then again, after deletion of yv, a cycle of length four arises (see figure 5).

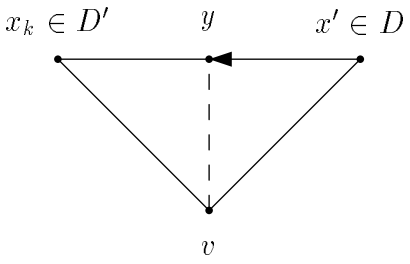

Figure 5

If $v \neq v'$, we can use the fact that $x_k \equiv_i y$, and $x_k v \in \tilde{E}$ or $yv' \in \tilde{E}$ (see figure 6).

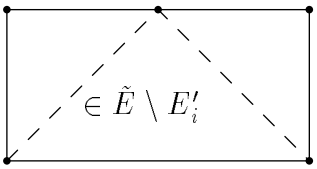

Figure 6

In both cases the deletion of the edge yv causes a chordless cycle of length four.

Therefore the Theorem is proved.

4.4 Structural Properties of Nondominating Components

We continue with a lemma which is useful for an efficient parallel algorithm that computes $<_i$.

Lemma 8 *For each vertex $x \in V_i$ the set $\mathcal{D}_x := \{D : D$ is a nondominating connected component of some cut of $\tilde{G}_i$ and $x \in D\}$ is totally ordered by inclusion.*

Proof Let D_1 and D_2 be nondominating connected components for cuts c_1 and c_2 respectively. Let $x \in D_1 \cap D_2$. Then each path from x to c_1 must pass c_2 or vice versa, otherwise, as in the proof of Claim 2 of Theorem 14, one of the components D_1 or D_2 is dominating. But then clearly $D_1 \subseteq D_2$ if each path from x to c_2 passes c_1 or vice versa.

Corollary 2 *For all nondominating connected components D_1 and D_2, D_1 and D_2 are comparable by inclusion or are disjoint.*

Lemma 9 *Let D_x be the unique smallest $D \in \mathcal{D}_x$. Then*

1. *$x \rightarrow_i y$ iff $xy \in \tilde{E}_i$ and $y \notin D_x$.*
2. *$D_x = \{y : \exists x' \equiv_i x\, y \rightarrow_i^* x'$ or $y \equiv_i x\}$.*

Proof. The first part can be seen as follows. $x \rightarrow_i y$ is equivalent to the statement that there is a nondominating saturated connected component D of some cut c such that $x \in D$ and $y \in c$. On the other hand, c is exactly the set of neighbors of D outside D. Therefore, for $xy \in \tilde{E}_i$, $x \rightarrow y$ iff there is a saturated connected component of any cut such that $x \in D$ and $y \notin D$. Therefore for $xy \in \tilde{E}_i$, $x \rightarrow y$ iff $x \in D_x$ and $y \notin D_x$, because D_x is the inclusion minimal saturated connected component of some cut which contains x.

The second statement can be seen as follows: D_x is a saturated connected component of some cut c. Suppose $y \rightarrow_i^* x$. Then no y' on the $\rightarrow_i$-chain from y to x can be in c (otherwise the $\rightarrow_i$-symbol would be turned). Therefore all elements of the $\rightarrow_i$-chain from y to x is in D_x. Suppose $y \equiv_i x$. Then $y \notin c$ and therefore $y \in D_x$.

Suppose vice versa that $y \in D_x$. Then there is a path from y to x, say $y = y_0, \ldots, y_l = x$ such that all y_j on this path are not in c. By theorem 14, we may abbreviate this path in such a way that $y_j \rightarrow_i y_{j-1}$ and $y_j \rightarrow_i y_{j+1}$ cannot be the case simultaneously. Moreover, by theorem 15, we can abbreviate this path in such a way, that the following situation cannot happen: $y_j \equiv_i y_{j+(-)1} \rightarrow_i y_{j+(-)2}$. Moreover, the following situation is excluded:
$x = y_l \rightarrow y_{l-1}$. Otherwise $y_{l-1} \in c$.
Therefore the following possibility remains: The sequence $y = y_0, \ldots, y_l = x$ has an initial segment of $\rightarrow_i$ (which may be empty) and an end segment of $\equiv_i$.

Corollary 3 *Let D_x be defined as in the previous lemma. Then for $xy \in \tilde{E}_i$,*

1. $x \rightarrow_i y$ *iff* $\#D_x < \#D_y$ *and*

2. $x \equiv_i y$ *iff* $D_x = D_y$ *iff* $\#D_x = \#D_y$.

We go back to our example:

We get $D_{v_1} := \{v_1\}$, $D_{v_2} := \{v_2\}$ and $D_{v_3} := \{v_3\}$, and the following direction $\rightarrow_i$ as in figure 9.

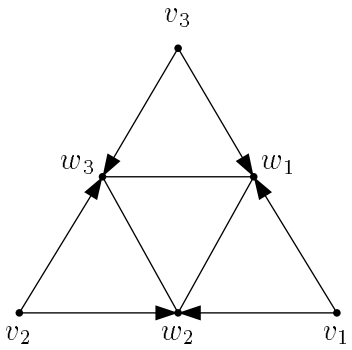

Figure 9

The pre-fill-in E_i' consists of the following additional edges:

w_3v_2, w_3u_1, w_3u_3, w_2u_2, w_2u_3, w_2u_4, w_1u_4, and w_1u_1 (see figure 10). Note that $w_1 \equiv_i w_2 \equiv_i w_3$.

Note that w_1, w_2, and w_3 are not in any proper nondominating and therefore $D_{w_1} = D_{w_2} = D_{w_3} = V_i$.

If we apply the SIMPLE CASE procedure to $W = \{w_1, w_2, w_3\}$ and V_0 as complete sets and edges of E_i' between W and V_0 then (w_1, w_2, w_3) is a suitable enumeration, and we get a fill-in as in figure 12.

For complexity considerations, the following result is useful.

Lemma 10 *The number of saturated connected components of all cuts is bounded by* $O(\#V_i)$.

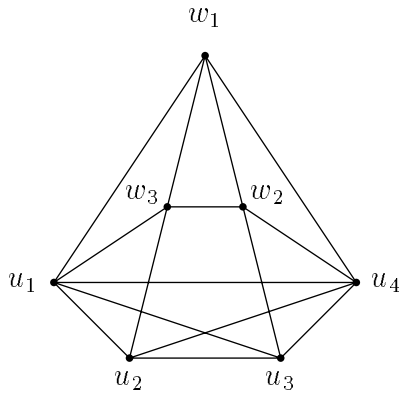

Figure 10

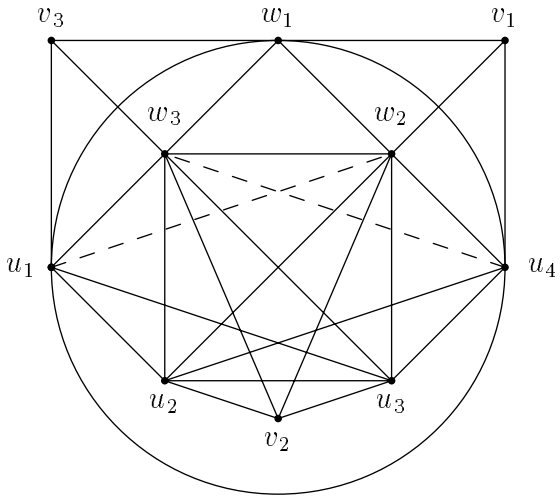

Figure 12

Proof. Suppose D is a nondominating connected component. Then D is of the form D_x. Therefore the number of nondominating components is bounded by the number of vertices in V_i. The number of dominating connected components is bounded by the number of cuts and therefore bounded by the number of vertices.

Corollary 4 *Assume the set ND_i of nondominating saturated connected components of all cuts is known. Then D_x and $\equiv_i$, can be computed in $O(\log n)$ CREW-time using $O(n^2)$ processors.*

Proof. D_x is computed by the computation of the $\#D$ with $x \in D$ of minimal cardinality. $\equiv_i$ is computed by comparing $\#D_x$ and $\#D_y$.

Remark: We could compute $\rightarrow_i$ with the same amount of complexity. But we never will use $\rightarrow$ explicitely in the algorithm.

We still have to compute the pre-fill-in E_i', for each $\equiv_i$-equivalence class A, the ordering $<_A$, and from these both results, the overall MEO $<$ and its chordal extension.

If we would go straightforward, we had to compute $\rightarrow_i^*$. To compute $\rightarrow_i^*$, we would need $O(n^3)$ processors.

To compute E_i' efficiently, we use the following simple fact:

Lemma 11 *For $x \in V_i$ and $v \in V_0$, $xv \in E_i'$ iff $xv \in E$ or there is a $y \rightarrow_i x$ such that $v \in N(D_y) \cap V_0$. (Remember that D_y is the smallest nondominating component which contains y).*

Proof. Suppose $xv \notin E$. Then $xv \in E_i'$ is equivalent to the statement that there is a y' such that $y' = y_0 \rightarrow_i y_1 \rightarrow_i \ldots \rightarrow_i y_l \rightarrow_i x$ and $y'v \in E$. Then by lemma 9, $y' \in D_{y_l}$ and $v \in N(D_{y_l})$. Vice versa let $v \in N(D_y)$, for some $y \rightarrow_i x$. Then, by lemma 9, there are a y'' such that $y \equiv y''$ and $y''v \in E$ or a y' such that $y'v \in E$ and $y'' \rightarrow_i^* y'$. By Theorem 15, $y'' \rightarrow_i^* x$ and therefore $xv \in E_i'$.

For purposes of efficiency, we consider, for each $x \in V_i$, only those $y \rightarrow_i x$ such that D_y is maximal by inclusion. Moreover, we are not interested in y itself, but in D_y.

Set $y \rightarrow_i' x$ iff $D_y \rightarrow_i' x$ iff y is a $y \rightarrow_i x$ such that D_y is maximal by inclusion.

Since the set ND_i of nondominating connected components is of the property that each pair is disjoint or comparable with respect to inclusion, for each $D_y \in ND_i$, there is at most one inclusion minimal $D'_y \in ND_i$ such that $D_y \subsetneqq D'_y$. If no such D'_y exists then we set $D'_y = V_i$.

We can compute these D'_y by computing, for $D = D_x$, the second smallest $D'_x \in ND_i$ which contains x.

Lemma 12 *For each $D_y \in ND_i$, let D'_y the unique inclusion minimal set in $ND_i \cup \{V_i\}$ such that $D_y \subsetneqq D'_y$. Then $D_y \rightarrow'_i x$ iff $x \in D'_y \setminus D_x$ and $yx \in \tilde{E}_i$.*

Proof. Suppose $y' \rightarrow x$, $y' \in D_y$, and $y' \rightarrow_i x$. Then there is a y'' with $D_y = D_{y''}$ and $y' \rightarrow_i^* y''$ or $y'' = y'$. By iterative application of theorem 14, for all vertices z on the $\rightarrow_i$-chain from y' to y'', $zx \in \tilde{E}_i$, and therefore $z \rightarrow_i x$. Therefore $y'' \rightarrow_i x$. By Theorem 15, $y \rightarrow_i x$.

Therefore $y \rightarrow_i x$ is equivalent to the statement that there is a $y' \in D_y$ such that $y' \rightarrow_i x$.

Therefore $D_y \rightarrow'_i x$ iff $y \rightarrow_i x$ and there is no $D_{y'}$ such that $D_y \subsetneqq D_{y'}$ and $y' \rightarrow x$, iff $y \rightarrow_i x$ and $x \in D'_y$ iff $x \in D'_y \setminus D_y$ and $yx \in \tilde{E}_i$.

Corollary 5 *For $v \in V_0$ and $x \in V_i$, $xv \in E'_i$ iff $xv \in E$ or there is a $D_y \rightarrow'_i x$ such that $v \in N(D_y) \cap V_0$.*

Corollary 6 *Suppose the number of pairs (D_y, x) such that $D_y \rightarrow'_i x$ is bounded by $m_i \geq \#V_i$. Suppose ND_i and, for each $D \in ND_i$, $N(D) \cap V_0$ is known. Then the pre-fill-in E'_i can be computed in $O(\log n)$ CREW-time using $O(nm_i)$ processors.*

Proof. For all $x \in V_i$, we compute D_x simultaneously in $O(\log n)$ CREW-time using $O(\#V_i^2)$ processors. For each $D \in ND_i$, we select a representative $x = x_D$ such that $D = D_x$ in $O(\log(\#V_i)$ CREW-time using $O(\#V_i)$ processors. $D' = D'_x$ is the second smallest $D'' \in ND_i$ which contains x_D. That can be computed in $O(\log \#V_i)$ CREW-time using $O(\#V_i^2)$ processors. We set $D_y \rightarrow'_i x$ iff $x \in D'_y \setminus D_y$ and $yx \in \tilde{E}_i$. Last can be checked in constant CREW-time using $O(\#V_i^2)$ processors. We set $E'_i = \{xv | x \in V_i, v \in V_0,$ such that $xv \in E$ or there is a $D \in ND_i$ with $D \rightarrow'_i x$ and $v \in N(D)\}$. Then E'_i can be computed in $O(\log n)$ CREW-time using nm_i processors.

PROCEDURE MEO($G = (V, E), <, E_<$)
Input Parameter: G
Output Parameter: $<, E_<$

BEGIN

1. If G is complete then $<$ is any ordering on V.
2. Apply Endsegment(G, V_0)
3. Compute the connected components $V_1, \ldots, V_k$ of $G|(V \setminus V_0)$
4. *For* $i = 1, \ldots, k$ *compute* $N_i := \{y \in V_0 : \exists x \in V_i xy \in E\}$, $E_0 := \{xy : x, y \in V_0, xy \in E$ *or* $\exists i : xy \in N_i\}$.
5. Apply MEO$((V_0, E_0), <_0, \tilde{E}_0)$ and for each $i = 1, \ldots, k$, MEO$(G|V_i, <'_i, \tilde{E}_i)$
6. Apply for each $i = 1, \ldots, k$: MinChord$((V_i, \tilde{E}_i), E|(N_i \cup V_i), N_i, <_i, \hat{E}_i)$ (*to compute the right* $<_i$ *and the corresponding chordal extension* $\hat{E}_i$)
7. $<$ is the concatenation of $<_i$ and of $<_0$, $E_< := \bigcup_{i=1}^{k}(\hat{E}_i) \cup \tilde{E}_0$

END.

Algorithm 4

The whole recursive MEO-procedure is described in Algorithm 4.

The procedure Minchord, as described in Algorithm 5, computes $<_i$ and the corresponding chordal extension $\hat{E}_i$.

To check the complexity of MEO, we still have to fill out the steps of MEO and MinChord which are written in italics. We have to do it in such a way, that a processor bound of $O(nm)$ is preserved in all recursion steps.

We computed a good endsegment V_0 and the connected components $V_1, \ldots, V_k$ of the complement. We let $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ be $G|V_1, \ldots, G|V_k$, and $G_0 = (V_0, E_0)$ arises from $G|V_0$ by making each neighborhood of any V_i complete.

Generally, we define graphs $G_{i_1,\ldots,i_q} = (V_{i_1,\ldots,i_q}, E_{i_1,\ldots,i_q})$, where $G_{i_1,\ldots,i_{q-1},0}$ is the graph corresponding to G_0, if we apply MEO to $G_{i_1,\ldots,i_{q-1}}$, and, for $j \neq 0$, $G_{i_1,\ldots,i_{q-1},j}$ is $G_{i_1,\ldots,i_{q-1}}|V_{i_1,\ldots,i_{q-1},j}$, where $V_{i_1,\ldots,i_{q-1},j}$ is the j^{th} connected component of $G_{i_1,\ldots,i_{q-1}}|(G_{i_1,\ldots,i_{q-1}} \setminus V_{i_1,\ldots,i_{q-1},0})$.

Obviously we get a subtree representation on all the $G_{i_1,\ldots,i_p}$. The parent of $G_{i_1,\ldots,i_p}$ is $G_{i_1,\ldots,i_{p-1}}$. Obviously, the vertex set of any $G_{i_1,\ldots,i_p}$ is a subset of the vertex set of its parent. Obviously, the children of the same parent are disjoint. Therefore we get:

Lemma 13 *$G_{i_1,\ldots,i_q}$ and $G_{i_1,\ldots,i_p}$ have nondisjoint vertex sets iff $G_{i_1,\ldots,i_q}$ is an ancestor of $G_{i_1,\ldots,i_p}$ or vice versa.*

PROCEDURE MinChord($\tilde{G}_i = (V_i, \tilde{E}_i), E|(N_i \cup V_i), N, <_i, \hat{E}_i$)
Input Parameter: $\tilde{G}_i := (V_i, \tilde{E}_i), N_i, E|(N_i \cup V_i)$
Output Parameter: $<_i, \hat{E}_i$

BEGIN

1. Compute Cut_i := set of cuts of $(V_i, \tilde{E}_i)$
2. **Compute the saturated connected components:** *Compute for each $c \in Cut_i$, the set $\mathcal{D}_c$ of connected components of $G_i|(V_i \setminus c)$;*
   for each $c \in \mathrm{Cut}_i$ and each $D \in \mathcal{D}_c$, *compute the set $N_D = N(D) \cap c$ of neighbors of D in c;*
   erase those D from $\mathcal{D}_c$ such that $\#N_D < \#c$.
3. *Compute for each $D \in \bigcup_{c \in Cut_i} \mathcal{D}_c$, the set $N(D) \cap V_0 = \{v \in N : \exists w \in D\ vw \in E\}$;*
   set $\mathrm{Num}(D) = \#N(D) \cap V_0$;
   set $\mathrm{Num'}D := \#D$.
4. $D \in \mathcal{D}_c$ is dominating iff $(\mathrm{Num}(D), \mathrm{Num}'(D))$ is maximal with respect to the lexicographic order and D is the unique maximal element of $\mathcal{D}_c$.
   $ND_i := \bigcup_{c \in \mathrm{Cut(i)}} \{D \in \mathcal{D}_c : D \text{ is not dominating }\}$
5. For all $x \in V_i : D_x$ is the $D \in ND_i$ such that $x \in D$ and $\#D$ is minimal;
   for each $D \in ND_i$, set $X_D = \{x : D_x = D\}$; x_D is a distinguished $x \in X_D$;
   for each $D \in ND_i$, D' is the second largest $D'' \in ND \cap \{V_i\}$, such that $x_D \in D''$.
6. **Compute the pre-fill-in E'_i**:
   - E' 1 For each $D \in ND_i$ and each $y \in D' \setminus D$ with $x_D y \in \tilde{E}_i$, set $D \rightarrow'_i y$;
   - E' 2 *For each $x \in V_i$ and each $v \in V_0$, set $xv \in E'_i$ iff $xv \in E$ or there is a $D \in ND_i$ such that $D \rightarrow'_i x$ and $v \in N(D) \cap V_0$.*
7. **Apply SIMPLE CASE for $\equiv_i$-equivalence classes**:
   - A 1 for each $D \in ND_i$, let $A_D = \{x : D_x = D\}$ and $A_{V_i} = V_i \setminus \bigcup_{D \in ND_i} D$ be the set of those vertices, appearing in no $D \in ND_i$.
   - A 2 for each $D \in ND$, apply SIMPLE CASE to $G_D = (A_D \cup (N(V_i) \cap V_0), E'_i|(A_D \cup V_0) \cup \tilde{E}_i|A_D \cup \tilde{E}_0|(N(V_i) \cap V_0))$ with $N(V_i \cap V_0)$ as an end segment, $<_D$ as the resulting ordering on A_D, and $\hat{E}_D$ as the resulting chordal extension.
8. **Compute $<_i$**:
   - S 1 Sort all $D \in ND_i$ with respect to $\#D$ to an ordering $<_{ND_i}$;
   - S 2 for $x, y \in V_0$, let $x <_i y$ iff $D_x <_{ND_i} D_y$ or $D_x = D_y$ and $x <_{D_x} y$.
   - S 3 $\hat{E}_i = \bigcup_{D \in ND_i \cup \{V_i\}} \hat{E}_D$

END.

Algorithm 5

We introduce some notions which will be useful to compute connected components in any $G_{i_1,\ldots,i_q}$.

Definition 3 *If $i_1,\ldots,i_p,0$ is an initial segment of $i_1,\ldots,i_q$ ($i_1,\ldots,i_p,0$ and $i_1,\ldots,i_q$ may be equal), then $G_{i_1,\ldots,i_p,0}$ is called a* zero ancestor *of $G_{i_1,\ldots,i_q}$.*

$G_{i_1,\ldots,i_q,j}$ is called a nonzero sibling *of $G_{i_1,\ldots,i_q,0}$, if $j \neq 0$.*

We denote by $V'_{i_1,\ldots,i_q}$ the set of all vertices of V which appear in a nonzero sibling of some zero ancestor $G_{i_1,\ldots,i_p,0}$ of $G_{i_1,\ldots,i_q}$.

The key for an nm processor bound of the procedure MEO in all recursion steps is the following:

Theorem 17 *xy is an edge in $G_{i_1,\ldots,i_p}$ iff there is a path from x to y in $G|V'_{i_1,\ldots,i_p}$.*

Proof. We prove the theorem by induction on p. For $p = 1$, we are done, by definition of G_0 and $G_1,\ldots,G_k$.

Suppose, $p > 1$. Then we consider the cases $i_p = 0$ and $i_p \neq 0$.

Suppose, $i_p = 0$. Then xy is an edge in $G_{i_1,\ldots,i_p}$ iff xy is an edge in $G_{i_1,\ldots,i_{p-1}}$ or there is a nonzero sibling $G_{i_1,\ldots,i_{p-1},j}$ and vertices x' and y' of $G_{i_1,\ldots,i_{p-1},j}$, such that xx' and yy' are edges in $G_{i_1,\ldots,i_{p-1}}$.

By construction, $G_{i_1,\ldots,i_{p-1},j}$ consists only of edges in $G_{i_1,\ldots,i_{p-1}}$. Note that, by construction, $G_{i_1,\ldots,i_{p-1},j}$ is connected. Hence there is an edge xy in $G_{i_1,\ldots,i_{p-1},0}$ iff xy is an edge in $G_{i_1,\ldots,i_{p-1}}$ or there is a nonzero sibling $G_{i_1,\ldots,i_{p-1},j}$ of $G_{i_1,\ldots,i_{p-1},0}$, such that there is a path from x to y in $G_{i_1,\ldots,i_{p-1}}|(V_{i_1,\ldots,i_{p-1},j} \cup \{x,y\}$. By the induction hypothesis, each edge in $G_{i_1,\ldots,i_{p-1}}$ can be replaced by a path in G consisting of vertices in $V'_{i_1,\ldots,i_{p-1}}$ and vertices of $V_{i_1,\ldots,i_{p-1},j}$, $j \neq 0$.

Therefore xy is an edge in $G_{i_1,\ldots,i_{p-1},0}$ iff there is a path from x to y in $G|V'_{i_1,\ldots,i_{p-1},0} = G|(V'_{i_1,\ldots,i_{p-1}} \cup \bigcup_{j>0} V_{i_1,\ldots,i_{p-1},j})$.

Now we consider the case that $i_p \neq 0$. Then all edges in $G_{i_1,\ldots,i_p}$ are edges in $G_{i_1,\ldots,i_{p-1}}$. By the induction hypothesis, we are done.

Also the following theorem is useful:

Theorem 18 *For any cut c of the chordal extension $F_<$ of an MEO $<$ the graph $G = (V, E)$, the (saturated) connected components of $G|(V \setminus c)$ and of $(V, F_<)|(V \setminus c)$ coincide.*

Proof Clearly each connected component of $(V, F_<)|_{(V \setminus c)}$ is the disjoint union of connected components of $G_i|V \setminus c$. Let $D_1, D_2, \ldots, D_k$ be connected components of $G|(V \setminus c)$ and $D_1 \cup D_2 \cup \ldots D_k$ be a connected component of $(V, F_<)|(V_i \setminus c)$. Then after deletion of all edges between different components D_i, D_j the the graph remains chordal, since no induced (chord free) cycle of the remaining graph can act in different connected components D_i, D_j since they are separated by the complete set c. Therefore for any minimal chordal extension $(V, F_<)$ of G, each connected component of $(V, F_<)|(V \setminus c)$ is also one connected component of $G|(V \setminus c)$.

Also saturatedness is preserved. It is caused by the fact that for any connected component D of $(V, F_<)|(V \setminus c)$ and for any vertex $x \in c$ which is in the neighborhood of D with respect to $F_<$ but not in the neighborhood of D with respect to G, we only had to erase all edges between x and D, and the the graph remains chordal: a cycle of $F_<$ becoming chordless after the deletion of all edges between x and D can have only two vertices of c (c is complete) and must pass x. This is a contradiction, because $x \in c$ and the cycle must leave c via x. Therefore the neighborhoods of D in c with respect to E and with respect to $F_<$ coincide.

Corollary 7

1. *For each $x \in V_{i_1,\ldots,i_p,0}$, there is an edge to a vertex $y \in V_{i_1,\ldots,i_p,j}$ in $G_{i_1,\ldots,i_p}$ iff there is an edge of G from x to a vertex y' in the connected component $V''_{i_1,\ldots,i_p,j}$ of $G|((V_{i_1,\ldots,i_p} \cup V'_{i_1,\ldots,i_p}) \setminus V_{i_1,\ldots,i_p,0}$, $V_{i_1,\ldots,i_p,j}$ belongs to.*

2. *Let c be a cut of the minimal chordal extension $\tilde{G}_{i_1,\ldots,i_p}$ of $G_{i_1,\ldots,i_p}$. Let $\mathcal{D}'_c$ be the set of connected components of $G|((V_{i_1,\ldots,i_p} \cup V'_{i_1,\ldots,i_p}) \setminus c)$ and $\mathcal{D}_c$ be the set of connected components of $G_{i_1,\ldots,i_p}|(V_{i_1,\ldots,i_p} \setminus c)$. Then the sets $\mathcal{D}$ and $\{D' \cap V_{i_1,\ldots,i_p} \neq \emptyset | D' \in \mathcal{D}'_c\}$ coincide.*

3. *Let c be a cut of the minimal chordal extension $\tilde{G}_{i_1,\ldots,i_p}$ of $G_{i_1,\ldots,i_p}$. Let D' be a connected component of $G|(V_{i_1,\ldots,i_p} \cup V'_{i_1,\ldots,i_p})$ and $D = D' \cap V_{i_1,\ldots,i_p}$ its corresponding connected component of $G_{i_1,\ldots,i_p}|(V_{i_1,\ldots,i_p} \setminus c)$. Then $x \in c$ is a neighbor of a vertex $v \in D$ with respect to $G_{i_1,\ldots,x_p}$ iff x is a neighbor of a vertex $v \in D'$ with respect to G.*

4. *Let c be a cut of the minimal chordal extension $\tilde{G}_{i_1,\ldots,i_p}$ of $G_{i_1,\ldots,i_p}$. Let D' be a connected component of $G|((V_{i_1,\ldots,i_p} \cup V'_{i_1,\ldots,i_p}) \setminus c)$ and $D = D' \cap V_{i_1,\ldots,i_k}$ its corresponding connected component of $G_{i_1,\ldots,i_p}|(V_{i_1,\ldots,i_p} \setminus c)$. Moreover, assume that $i_p \neq \emptyset$. Then a vertex $x \in V_{i_1,\ldots,i_{p-1},0}$ is adjacent to some vertex of D with respect to $G_{i_1,\ldots,i_{p-1}}$ iff x is adjacent to some vertex in D' with respect to G.*

Proof.

1 : There is an edge of $G_{i_1,\ldots,x_p}$ from x to a vertex $y \in V_{i_1,\ldots,x_p,j}$ iff $xy \in E$ or there is a path P from x to y of G such that all internal vertices of P are in $V'_{i_1,\ldots,x_p}$. Let y' be the first internal vertex of P. Then $y' \in V''_{i_1,\ldots,x_p,j}$, because y' and y can be joined by a path using only internal vertices from $V'_{i_1,\ldots,x_p}$ and therefore no vertices from $V_{i_1,\ldots,x_p,0}$. Then x is adjacent to a vertex of $V''_{i_1,\ldots,x_p,j}$ in G.

Vice versa, let x be adjacent to $y' \in V''_{i_1,\ldots,x_p,j}$. One possibility is that $y \in V_{i_1,\ldots,x_p,j}$. Then we are done. It remains to consider the case that $y \in V'_{i_1,\ldots,x_p}$. Then there is a path in G from y' to some $y \in V_{i_1,\ldots,x_p,j}$, where all internal edges are in $V'_{i_1,\ldots,x_p}$. Then xy is an edge of $G_{i_1,\ldots,x_p}$.

2: By theorem 17, there is a path from u to v in $G_{i_1,\ldots,i_p}$ using only vertices not in c iff there is a path from u to v in $G|(V_{i_1,\ldots,x_p} \cup V'_{i_1,\ldots,x_p})$ using only vertices not in c. Therefore the set $\mathcal{D}$ of connected components of $G_{i_1,\ldots,x_p}|V_{i_1,\ldots,x_p} \setminus c$ and the set of nonempty intersections of $G|((V_{i_1,\ldots,x_p} \cup V'_{i_1,\ldots,x_p}) \setminus c)$ with $V_{i_1,\ldots,x_p}$ coincide.

3 The argument is the same as in 1. Suppose $x \in c$. Then x is adjacent to some $y \in D$ with respect to $G_{i_1,\ldots,x_p}$ iff $xy \in E$ or there is there is a path P from x to y in G whose internal vertices are all in $V'_{i_1,\ldots,x_p}$. The internal vertices are all in D', because they are all not in $V_{i_1,\ldots,x_p}$ and therefore not in c and therefore all with y in the same connected component of $G|((V_{i_1,\ldots,x_p} \cup V'_{i_1,\ldots,x_p}) \setminus c)$. Therefore x is adjacent to some vertex $y' \in D'$ in G.

Vice versa, suppose x is adjacent to some vertex $y' \in D'$ in G. Then we get a path P' in G from y' to a vertex $y \in D$ such that all internal vertices are not in $V_{i_1,\ldots,x_p}$ and therefore in $V'_{i_1,\ldots,x_p}$. Therefore we get a path P from x to y in G such that all internal vertices are in $V'_{i_1,\ldots,x_p}$. Therefore xy is an edge in $G_{i_1,\ldots,x_p}$.

4 can be proved in the same way as 3.

To get an algorithm which makes the neighborhood of $V_{i_1,\ldots,x_p,j}$ in $V_{i_1,\ldots,x_p,0}$ with respect to $G_{i_1,\ldots,x_p}$ complete, we use the following trivial consequence of the last corollary,

Corollary 8 *For each vertex $v \in V_{i_1,\ldots,x_p,0}$, the number of j such that there is a vertex $y \in V_{i_1,\ldots,x_p,j}$, is bounded by the number of edges in G which are incident with x.*

Proof. This is an immediate consequence of the last corollary, item 1.

PROCEDURE MAKE THE NEIGHBORHOOD COMPLETE($V_0, V_1, \ldots, V_k, G, E_0$)
Input Parameter: $V_0, \ldots, V_k, G = (V, E)$
Output Parameter: E_0

BEGIN

1. For $x \in V_0$, $y \in V_j$, and $xy \in E$, set $xV_j \in R$.
2. For $x, y \in V_0$ set $xy \in E_0$ iff there is an $xV_j \in R$ such that $yV_j \in R$.

END.

Algorithm 6

We can compute G_0 as in algorithm 6.

Clearly Algorithm 6 makes $N(V_j) \cap V_0$ complete, for each $j = 1, \ldots, k$. Since each x is only in one V_j, R can be computed in constant CRCW-time using $O(n^2)$ processors (in each recursion step), and therefore in $O(\log n)$ CREW-time using $O(n^2)$ processors. By the last corollary, the second step can be executed in $O(\log n)$ CREW-time using $O(nm)$ processors. This bound is valid in all recursion steps.

We continue with the computation of the connected components of $G_{i_1,\ldots,i_k}|(V_{i_1,\ldots,i_k} \setminus c)$. This is done in Algorithm 7.

PROCEDURE COMPUTE THE SET $\mathcal{D}_c$ OF CONNECTED COMPONENTS OF $G_{i_1,\ldots,i_p}|(V_{i_1,\ldots,i_p} \setminus c)$
Input Parameter: $G_{i_1,\ldots,i_p}, G, c$
Output Parameter: $\mathcal{D}_c$

BEGIN

1. Compute the set Z of indices of zero-ancestors of $G_{i_1,\ldots,i_p}$.
2. For each $i_1, \ldots, i_q, 0$ in Z compute the set $S(i_1, \ldots, i_q)$ of indices of nonzero siblings of $G_{i_1,\ldots,i_q,0}$.
3. Set $S = \bigcup_{i_1,\ldots,i_q,0 \in Z} S(i_1, \ldots, i_q)$.
4. Set $V'_{i_1,\ldots,i_p} = \bigcup_{j_1,\ldots,j'_q \in S} V_{j_1,\ldots,j'_q}$.
5. Compute the connected components $\mathcal{D}'_c$ of $G|((V_{i_1,\ldots,i_p} \cup V'_{i_1,\ldots,i_p}) \setminus c)$.
6. For each $D' \in \mathcal{D}'_c$, set $D = D' \cap V_{i_1,\ldots,i_p}$.
   If $D \neq \emptyset$, set $D \in \mathcal{D}_c$

END.

Algorithm 7

By 2 of the second last corollary, the algorithm computes the set of connected components of $G_{i_1,\ldots,i_p}|(V_{i_1,\ldots,i_p} \setminus c)$.

The complexity is checked as follows:

Step 1 can be done sequentially in logarithmic time, because the recursion depth of MEO is logarithmic and therefore each $G_{i_1,\ldots,i_p}$ has only logarithmically many ancestors, that means $p \leq \log n$.

The computation of nonzero siblings of any zero ancestor can be done by as many processors as nonzero siblings exist in constant CREW-time. The processor bound is n.

The computation the set of all nonzero siblings in 3 needs $O(n)$ processors and $O(\log n)$ CREW-time.

In 4, the set $V'_{i_1,\ldots,i_p}$ is computed in $O(\log n)$ CREW-time using $O(n)$ processors. Note that all $V_{j_1,\ldots,j_{q'}}$ are pairwise disjoint.

The last step 5 is bounded by $O(n+m)$ processors and a CRCW-time of $O(\log n)$ [SV 82].

Since the number of cuts is bounded by n, the overall complexity of computing $\mathcal{D}_c$, for all cuts c simultaneously, is bounded by nm.

To compute the set of neighbors of D in c with respect to $G_{i_1,\ldots,i_p}$, we compute the set of neighbors of the corresponding component $D' \in \mathcal{D}'_c$ in c with respect to G. We proceed as in Algorithm 8.

PROCEDURE COMPUTE FOR ALL c AND ALL $D' \in \mathcal{D}'_c$ THE SET $N(D) \cap c$ OF NEIGHBORS IN c.
Input Parameter: $c, \mathcal{D}'_c, G$
Output Parameter: $\{N(D') \cap c | D' \in \mathcal{D}'_c\}$

BEGIN

1. For each $c \in Cut_{i_1,\ldots,i_p}$ and each $xy \in E$ such that $x \in c$ and $y \notin c$, determine the $D'_{y,c} \in , D'_c$ such that $y \in D'_{y,c}$.
2. For $c \in Cut_{i_1,\ldots,i_p}$ and each edge $xy \in E$ such that $x \in c$ and $y \notin c$, x is set into thge neighborhood of $D'_{y,c}$.

END.

Algorithm 8

Clearly this algorithm computes, for each cut c and each connected component belonging to c, the set of neighbors in c.

The first and the second step can be executed in constant CREW-time using $O(nm)$ processors, since the number of cuts is bounded by n and the number of edges in E is m.

We continue with the computation of the neighbors of D in $V_{i_1,\ldots,i_p,0}$ with respect to $G_{i_1,\ldots,i_p}$, where D is a saturated connected component of some cut $c \in Cut_{i_1,\ldots,i_p,j}$. Again we compute the neighborhood of D' with respect to G.

The algorithm works as Algorithm 8. We only replace c by $V_{i_1,\ldots,i_p,0}$.

It remains to fill out step E'2. The algorithm was mentioned in the proof of corollary 6. To get a processor bound of $O(nm)$ in all recursion steps, we have to show that, for any x, the number of D_y, $D_y \rightarrow_i' x$ is bounded by the number of neighbors of x in G. Note that D_{y_1} and D_{y_2} are equal or disjoint if $D_{y_1} \rightarrow_i' x$ and $D_{y_2} \rightarrow_i' x$. Moreove, in the latter case, D'_{y_1} and D'_{y_2} are disjoint. A necessary condition that x is in the neighborhood of D_y in $G_{i_1,\ldots,i_p}$ and therefore in the neighborhood of D'_y in G, i.e. there is a $z \in D'_y$ such that $zx \in E$. Therefore the number of D_y with $D_y \rightarrow_i' x$ is bounded by the number of neighbors of x in G.

Hereby, all gaps in MEO and MinChord are filled. Putting all the results together, we get:

Theorem 19 *An MEO and a minimal chordal extension can be computed in $O(\log^3 n)$ CREW-time by $O(nm)$ processors.*

5 Applications

We summarize some applications of our parallel MEO algorithm. We refer to [Ro 73], [Ta 85], [Kl 88], [DK 88a], [GH 88] for fundamentals. One application is *symmetric sparse Gaussian elimination*. The problem is to compute, for any symmetric matrix with nonzero entries on the diagonal, a Gaussian elimination scheme such that the set of entries becoming nonzero is minimized with respect to inclusion [OCF 76, Ro 73]. We call such an elimination scheme a *sparse Gaussian elimination*. To compute a sparse Gaussian elimination, we proceed as follows. For any symmetric matrix $A = (a_{i,j})_{i,j=1}^n$, we consider the corresponding graph $G_A = (V_A, E_A) = (\{1,\ldots,n\}, \{ij | a_{i,j} \neq 0\}$. By [Ro 73], the problem to compute a sparse Gaussian elimination for A is equivalent to the problem of the computation of an MEO for G_A. Therefore we get immediately.

Theorem 20 *There is a CREW-algorithm which computes, for any symmetric $n \times n$-matrix with nonzero entries on the diagonal and m nonzero entries, a sparse Gaussian elimination scheme in $O(\log n)^3$ time using $O(nm)$ processors.*

Another application of MEO is *clique decomposition*. The problem of clique decomposition is, given a graph $G = (V, E)$, to compute the set of cuts of G which induce a complete subgraph of G and to compute, with the help of the set of cuts, the inclusion maximal components of G which are not decomposable by complete cuts. Sequentially,

this problem can be solved in $O(nm)$ time [Ta 85]. He computed the cuts of the chordal extension of an MEO and selected those cuts of the chordal extension which are also complete in the original graph. It is not difficult to parallelize this procedure in $O(\log n)$ CREW-time using $O(nm)$ processors. To get the components of the clique decomposition, we consider the cliques sets of the chordal extension. We compute the *clique tree* for the chordal extension. It consists of the set of cliques of the chordal extension as vertex set and has the property that, for each vertex x of the given graph, the set of cliques containing x forms a subtree [Bu 74, Ga74]. Note that each edge of the clique tree corresponds to the cut of those vertices of the chordal extension which are in both incident cliques. A clique tree for the chordal extension can be computed from the MEO in $O(\log n)$ CREW-time using $O(n^2)$ processors [Kl 88]. To compute the components, we unify those cliques of the chordal extension to one component which are not separable by an edge of the clique tree corresponding to a cut of the chordal extension which is also complete in the given graph. This can be done by tree contraction techniques in $O(\log n)$ CREW-time using $O(n)$ processors. *Therefore the overall complexity of clique decomposition is $O(nm)$ processors and $O(\log^3 m)$ CREW-time.*

6 Further Research

From the main result of this paper the following questions arise.

1. Is there a way to improve our algorithm with respect to the number of processors ($O(nm)$) giving the MEO an even better sequential time algorithm ([RTL 76] provides an $O(nm)$ time algorithm)?

2. Is it possible to modify our algorithm to work in $O(\log^2 n)$ parallel time and in the same number of processors on a CRCW PRAM (P. Klein asked this question in [Kl])? The recursive structure of any such MEO algorithm working in a 'shallow' $O(\log^2 n)$ parallel time would be of its own interest!

3. The Breadth-First Search (BFS) algorithm of Theorem 20 uses the chordal 'parent-richest neighbors' method of [Kl 88] applied to an MEO of an input graph. Are there MEOs which can be used directly to construct the *Depth-First Search (DFS)* tree for an arbitrary graph? Can our MEO algorithm be modified as to generate efficiently DFS trees of arbitrary graphs? (In general it will be very interesting to shed some light on the connection between DFS orderings and MEOs. At present there is not much known.)

Acknowledgements

Bob Tarjan has raised to us the question of an efficient parallel algorithm for the MEO Problem. We are thankful to him, and also to Philip Klein for the stimulating discussions.

References

[AA 87] Aggarwal, A., Anderson, R., *A Random NC Algorithm for Depth First Search*, Proc. 19$^{\text{th}}$ ACM STOC (1987), pp. 325–334.

[Bu 74] Bunemann, P., *A Characterization on Digid Circuit Graphs*, Discrete Mathematics 9 (1974), pp. 205-212.

[Co 86] Cole, R., *Parallel Merge Sort*, 27$^{\text{th}}$ FOCS (1986), pp. 511-516.

[Co 85] Cook, S.A., *A Taxonomy of Problems with Fast Parallel Algorithms*, Information and Control 64 (1985), pp. 2-22.

[CSV 82] Chandra, A., Stockmeyer, L., Vishkin, U., *A Complexity Theory for Unbounded Fan-In Parallelism*, 23$^{\text{th}}$ FOCS (1982), pp. 1-13.

[DK 86] Dahlhaus, E., Karpinski, M., *The Matching Problem for Strongly Chordal Graphs is in NC*, Research Report No. 855-CS, University of Bonn (Dec. 1986).

[DK 87] Dahlhaus, E., Karpinski, M., *Fast Parallel Computation of Perfect and Strongly Perfect Elimination Schemes*, Research Report No. 8513-CS, University of Bonn (Nov. 1987), submitted for publication.

[DK 88a] Dahlhaus, E., Karpinski, M., *Fast Parallel Decomposition by Clique Separators*, Research Report No. 8525-CS, University of Bonn (May 1988).

[DK 88b] Dahlhaus, E., Karpinski, M., *Efficient Parallel Algorithm for Clique Separator Decomposition*, Research Report No. 8531-CS, University of Bonn (Nov. 1988).

[DK 89] Dahlhaus, E., Karpinski, M., *An efficient Parallel Algorithm for the Minimal Elimination Ordering (MEO) of an Arbitrary Graph (Extended Abstract)*, Proc. 30$^{\text{th}}$ FOCS (1989), pp. 454-459.

[DKN 89] Dahlhaus, E., Karpinski, M. Novick, M., *Fast parallel algorithms for the clique separator decomposition*, 1$^{\text{st}}$ ACM-SIAM Symposium on Discrete Algorithms (1990), pp. 244-251.

[Di 76] Dirac, G.A., *On Rigid Circuit Graphs*, Abh. Math. Sem. der Univ. Hamburg 25 (1961), pp. 71-76.

[Di 87] Diestel, R., *Simplicial Decomposition of Graphs — Some Uniqueness Results*, Journal of Combinatorial Theory, Ser. B 42 (1987), pp. 133-145.

[Ed 87] Edenbrandt, A., *Chordal Graph Recognition is in NC*, Information Processing Letters 24 (1987), pp. 239-241.

[FW 78] Fortune, S., Wyllie, S., *Parallelism in Random Access Machines*, Proc. 10^{th} ACM-STOC (1978), pp. 114-118.

[Ga 72] Gavril, F., *Algorithms for Minimum Coloring, Maximum Clique, Minimum Coloring by Cliques, and Maximum Independent Sets of a Chordal Graph*, SIAM J. Comput. (1972), pp. 180-187.

[Ga74] F. Gavril, *The Intersection Graphs of Subtrees in Trees Are Exactely the Chordal Graphs*, Journal of Cobinatorial Theory Series B, vol. 16(1974), S. 47-56.

[GM 87] Gazit, H., Miller, G.L., *An Improved Algorithm for BFS of a Directed Graph*, manuscript, USC (1987).

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

[GRE 84] Gilbert, J., Rose D., Edenbrandt, A., *A Separator Theorem for Chordal Graphs*, SIAM J. for Algebraic and Discrete Methods 15 (1984), pp. 306-313.

[Go 80] Golumbic, M.C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, N.Y., 1980.

[HL 88] Ho, C.W., Lee, R.C.T. *Efficient Parallel Algorithms for Finding Maximal Cliques, Clique Trees and Minimum Coloring on Chordal Graphs*, Information Processing Letters 28 (1988), pp. 301-309.

[KR 88] R. Karp, V. Ramachandran, *A Survey of Parallel Algorithms for Shared-Memory Machines*, Research Report No. UCB/CSD 88/407, University of California, Berkeley (1988); to appear in: Handbook of Theoretical Computer Science, North Holland (1989).

[Kl] Klein, Ph., personal communication.

[Kl 88] Klein, Ph., *Efficient Parallel Algorithms on Chordal Graphs*, Proc. 29^{th} IEEE FOCS (1988).

[LF 80] R. Ladner, M. Fischer, *Parallel Prefix Computation*, Journal of the ACM 27 (1980), S. 831-838.

[NNS 87] Naor, J., Naor, M., Schäffer, A., *Fast Parallel Algorithms for Chordal Graphs*, Proc. 19^{th} ACM STOC (1987), pp. 355-364.

[No 88] Novick, M.B., *NC Algorithms for the Clique Separator Decomposition*, Cornell University, manuscript (Nov. 1988).

[OCF 76] Ohtsuki, T., Cheung, L.K., Fujisawa, T., *Minimal Triangulation of a Graph and Optimal Pivoting Order in a Sparse Matrix*, J. Math. Analysis and Applications 54 (1976), pp. 622-633.

[Ro 70] Rose, D.J., *Triangulated Graphs and the Elimination Process*, J. Math. Appl. 32 (1970), pp. 597-609.

[Ro 73] Rose, D.J., *A Graph Theoretic Study of the Numerical Solution of Sparse Positive Definit Systems of Linear Equations*, in: R. Read (ed.), *Graph Theory and Computing*, Academic Press, New York (1973), pp. 183-217.

[RTL 76] Rose, D., Tarjan, R.E., Lueker, G., *Algorithmic Aspects of Vertex Elimination on Graphs*, SIAM J. Comput. 5, pp. 266-283.

[Sa 76] Savage, J.E., *The Complexity of Computing*, Wiley, N.Y. (1976).

[SV 82] Shiloach, Y., Vishkin, U., *An $O(\log n)$ Parallel Connectivity Algorithm*, Journal of Algorithms 3 (1982), pp. 57-67.

[Ta 85] Tarjan, R.E., *Decomposition by Clique Separators*, Discrete Mathematics 55 (1985), pp. 221-232.

[TV 85] R. Tarjan, U. Vishkin, *An Efficient Parallel Biconnectivity Algorithm*, SIAM-Journal on Computing 14 (1985), pp. 862-873.

[TY 84] Tarjan, R.E., Yannakakis, M., *Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclitic y of Graphs, Test Acyclitic y of Hypergraphs, and Selectively Reduce Hypergraphs*, SIAM J. Comput. (1984), pp. 556-579.

[TY 85] Tarjan, R.E., Yannakakis, M., *Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclitic y of Graphs, Test Acyclitic y of Hypergraphs, and Selectively Reduce Hypergraphs*, Addendum, SIAM J. Comput. 14 (1985), pp. 254-255.

[Ya 81] Yannakakis, M., *Computing the Minimum Fill-In is NP-Complete*, SIAM J. Algebraic Discrete Math. 2 (1981), pp. 77-79.