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

Elias Dahlhaus[1], Marek Karpinski[2]

TR-89-024

May, 1989

Abstract

We design the first efficient parallel algorithm for computing Minimal Elimination Ordering (MEO) of an arbitrary graph.

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

The MEO Problem for arbitrary graphs arises in a number of combinatorial optimization problems, as well as in database applications, scheduling problems, and the sparse Gaussian elimination of 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), cf., e.g., [RTL 76], [Ta 85], and to the problem of the Gaussian elimination of sparse symmetric matrices [Ro 70], [Ro 73]. (The problem of computing *Minimum Fill-In is* known to be NP-complete [Ya 81].) It gives also an alternative to [GM 87] efficient parallel algorithm for computing Breadth-First Search (BFS) trees in arbitrary graphs using $O(n\,m)$ processors on a CRCW PRAM.

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.

---

1 Dept. of Computer Science, University of Bonn.

2 On leave from the University of Bonn, research partially supported by the Leibniz Center for Research in Computer Science, by the DFG Grant KA 673/2-1, and by the SERC Grant GR-E 68297.

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. 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.

Given any graph $G = (V, E)$ and an ordering (one-to-one numbering) $<$ on V. Define $E_<$ to be the corresponding *chordal extension* of G, i.e. the minimal extension E' of E, such that if $x < y$, $x < z$ and $xy,\ xz \in E' \Longrightarrow yz \in E$.

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.

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]):

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

An ordering $<$ is *minimal* if there is no other ordering $<'$ of V such that $F_{<'} \subset F_<$.

The Minimal Fill-In Problem is to compute for arbitrary given graph $G = (V, E)$, a Minimal Fill-In of G (cf. [RTL 76], [Ta 85]).

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) $E_<$ 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 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 and working in $O(\log^3 n)$ parallel time and $O(nm)$ processors on 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], [No 88]), and for the *first* time provides a parallel technique of computing Minimal Fill-In (cf. [Ta 85]) for arbitrary graphs, and combining our algorithm with Cholesky factorization algorithm of Gilbert and Hafsteinsson [GH 88], an efficient parallel algorithm for the Gaussian Elimination on sparse symmetric matrices (cf. [Ro 73]). It gives also an alternative to [GM 87] parallel algorithm for computing Breadth-First Search (BFS) tree in arbitrary graphs working in $O(n\,m)$ processors. Our BFS algorithm is based on Klein's [Kl 88] chordal method of assigning parents to be 'richest neighbors' applied to an MEO of a given graph.

It is an interesting open question whether our MEO-algorithm can be used to design an efficient (deterministic) parallel algorithm for a Depth-First Search (DFS) tree in arbitrary graphs (cf. [AA 87] for the randomized solution; see also [KR 88]).

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.

In 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 $N_G(x) = \{y \mid xy \in E\}$. For $M \subseteq V$ $N_G(M) = \bigcup_{x \in M} N(x)$.

The computation model is the concurrent-read concurrent-write parallel random access machine (CRCW PRAM) (cf. [FW 78], [Co 85], [KR 88]). We assume that basic operations and predicates like the addition, the subtraction and the predicate $<$ for n-ary binary numbers need $O(n^3)$ processors and constant time or $O(n)$ processors and $O(\log n)$ time, and the multiplication of two n-ary numbers needs $O(n \log n \log\log n)$ processors and $O(\log n)$ parallel time.

There exist uniform boolean circuits classes with an unbounded fan-in and the same sizes and depths, realizing these operations ([CSV 82], [KR 88]).

Whenever nothing else is said, 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 on parallel computation.

Theorem 1.

i) (see [SV 82])
The transitive closure and a spanning tree of any graph can be computed in $O(\log n)$ parallel time and $O(n + m)$ processors.

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

Given a tree $T = (V_T, E_T)$ with a root r. Then 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 $(x, y) \in A_r$, 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 *depth-first-ordering* $\prec$ is defined as follows:

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

Theorem 2. (see [Kl 88]) For a tree T and a 'root' $r \in V_T$ an ordering (numbering), such that each initial segment is a subtree containing r, can be computed in $O(\log n)$ parallel time and $O(n)$ processors.

A graph is called *chordal* iff it has no induced edge of length > 3.

Chordal graphs can be characterized in the following way.

Theorem 3. The following statements are equivalent:

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

ii) $G = (V, E)$ has a *Perfect Elimination Ordering* $<$, that means if $x < y$, $x < z$ and xy, $xz \in E$, then $yz \in E$.

iii) $G = (V, E)$ is the vertex intersection graph ([Ga 72], [Bu 74]) of a collection $\mathcal{S}_G$ of subtrees of some tree T.

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

Suppose, 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]).

Philip Klein [Kl 88] proved the following result

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

2 The Global Strategy.

Firstly, in an analogy to Philip Klein's perfect elimination (PEO) algorithm ([Kl 88]) we shall compute an endsegment $V' \subseteq V$ of an MEO, such that $\#V' \leq \frac{2}{3}\#V$ and for all connected components C of $V \setminus V' \#C \leq \frac{2}{3}\#V$ by a later defined procedure *Endsegment*.

The divide and conquer strategy works as follows:

1. All vertices $v_1, v_2 \in V'$ adjacent to the same connected component of $V \setminus V'$ are joined by an edge $\in \tilde{E}$.
2. We apply the recursion of computing a minimal chordal extension to $(V', \tilde{E})$ and for each connected component C of $V \setminus V'$ to $G \downarrow C := (C, \{xy \in E \mid x, y \in C\})$.
3. For each connected component C of $V \setminus V'$ and the set $C' := \{y \in V \mid vv' \in E\}$ add suitable additional edges between C and C' by a procedure *Minchord*.

Remark: We begin with a first complexity remark on the first step. The connected components can be computed in $O(n^2)$ processors and $O(\log n)$ time. This is true also in the case of recursive application, since at the same time it is operated on disjoint subgraphs. Since we can assume that $n \leq m+1$ we also have a processor bound $O(nm)$. The computation of $\tilde{E}$ needs in the first step constant time and $O(nm)$ processors. In the recursive application eventually new edges are used. But for a new edge $e = uv$ one can find edges $e_1 = u_1v_1$ and $v_1'v$ generating it. Successively we get edges $e_{i+1} = uv_{i+1}$ and $v_{i+1}'v_i$ generating e_i until some e_k is in E, which we also call the representative of e. Since the recursion operates on disjoint vertex sets on the same time, $e_k = u_1u_2$ may be a representative of edges incident to u_1 and incident to u_2. But edges, incident to u_1 having the same representative join u_1 by the same connected component of $V \setminus V'$. Therefore after contraction of the connected components of $V \setminus V'$, which can be done in constant time by $O(n^2)$ processors, we have again an edge bound of m. Therefore $\tilde{E}$ can be computed in $O(nm)$ processors also in the recursive application.

In the rest of this section we look for the problem how to compute such a set V' . For this purpose we use the following known result.

Lemma 1. [RTL 76] (V, E') is an inclusion minimal extension of (V, E), iff $E \subseteq E'$, (V, E') is chordal and for $e \in E' \setminus E$ $(V, E' \setminus \{e\})$ has an induced cycle of four vertices (and edges).

The key result for the construction of V' is the following

Theorem 5. Let M be a connected subset of V. Then $N(M)$ is an endsegment of an MEO.

PROOF Let $C_1 \dots C_k$ be the connected components of $V \setminus N(M)$ and $\tilde{C}_i := N(C_i) \cap N(M)$ for $i = 1, \dots, k$.

Let $C_i' := C_i \cup \tilde{C}_i$ and let $G_M := (N(M), E_M')$ where E_M arises from E by making all $\tilde{C}_i$ complete. Let $G_i' := (C_i', E_i)$ where E_i arises from E restricted to C_i' by making $\tilde{C}_i$ complete.
Let (C_i', E_i') and (C_M', E_M') be minimal chordal extensions of (C_i', E_i) and (C', E_M) resp..

Claim. $G' := (V, E_M' \cup \bigcup_i E_i')$ is a minimal chordal extension.

PROOF OF THE CLAIM It is easily seen, that G' is chordal. To prove that G' is a minimal chordal extension of G, we have only to prove that each edge $xy \in E_i' \setminus E$, $x, y \in \tilde{C}_i$ forms after its deletion an induced cycle.

Since x, y are adjacent to the same connected component C_i, one finds a path $x, y_1 \dots y_l, y$, s.t. $y_i \in C_i$. 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 $x, m_1 \dots m_l, y$, s.t. all $m_i \in M$.
Since also $x, m_1 \dots, y$ 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 after deletion of xy an induced cycle of length four. □

It remains to check, that $N(M)$ is an endsegment of a perfect elimination ordering of G'. But this is trivial, since $N(M)$ is closed by chordless paths of G'. □

We continue with another useful remark.

Theorem 6. For $xy \notin E$, $C_{xy} := x \cup (N(x) \cap N(y))$ is an endsegment of an MEO.

PROOF Let $\tilde{E}' := E \cup \{uv \mid u, v \in C_{xy}\}$. It is clear that each minimal chordal extension of $(V, \tilde{E})$ is a minimal chordal extension G' of (V, E), since for $uv \notin E$, x, u, y, v forms a cycle of length four. Since C_{xy} is complete in G', it is an endsegment of a perfect elimination ordering of G'. □

Now we are able to compute a set V', which is an endsegment of an MEO, in parallel.

Procedure Endsegment (G, V')

INPUT PARAMETER : $G = (V, E)$
OUTPUT PARAMETER : $V' \subseteq V$

Begin

1)For each $v \in V$ compute $d(v) := \#\{w \in V \mid vw \in E\}$
($n + m$ processors, O($\log n$) time by iterated addition of 1) .

2) a)Let $D_1 := \{v \mid d(v) \leq \frac{2}{3}\#V\}$; let $D_2 := \{v \mid d(v) > \frac{2}{3}\#V\}$;
compute the connected components $C_1 \ldots C_k$ of D_1.
(O($n + m$) processors, O($\log n$) time)

b)Sort (C_i): by their sizes in descending order

b1)For each i compute $\#C_i$. (O(n) processors, O($\log n$) time)

b2)Sort $(\#C_i)$: (numbers of length ($\log n$) are compared, therefore one $<$ needs O($\log n$) processors and O($\log \log n$) time).

The whole sorting procedure needs O($\log n \log \log k$) time by $k \cdot \log \frac{n}{k} \leq n$ processors.

3)If $\#C_1 > \frac{2}{3}\#V$:

a)Compute a spanning tree T_1 of C_1. (O($n + m$) processors, O($\log n$) time)

b)Pick up $r \in C_1$, s.t. $d(r)$ is maximal.
Compute the enumeration of a depth-first order on T_1 with root r, say $(v_1 \ldots v_p)$.
(O(n) processors, O($\log n$) time)

c)For $i \in \{1 \ldots p\}$ let $B_i := N(v_1 \ldots v_i)$. (O($n + m$) processors (see [KI 88]))

d)Compute for each i $\#B_i$.
Pick up a $\#B_i$, s.t.

$$\frac{1}{3}\#V \leq B_i \leq \frac{2}{3}\#B_i; \quad V' := B_i$$

STOP.
(O(n) processors, O($\log n$) time)
(*Remark.* Each B_i is an endsegment of an MEO by Theorem 5)

4)Otherwise:

a)Check wether D_2 is complete by counting the number of edges inside D_2.
(O($n + m$) processors, O($\log n$) time)

b)If YES and $\#D_2 \leq \frac{2}{3}\#V$ then $V' = D_2$.
If YES and $\#D_2 > \frac{2}{3}\#V$:
(V' is a subset of D_2, s.t. $\#V' = \frac{2}{3}\#V$;)
STOP.
(O($n + m$) processors, O($\log n$) time)

c)If NOT, find $x, y \in D_2$, $xy \notin E$.
(O($n + m$) processors, O($\log n$) time (see [KI 88]))

d)Let U be the common neighbourhood of x and y.
(O($n + m$) processors, constant time)
If $\{x\} \cup U \leq \frac{2}{3}\#V$, then $V' := \{x\} \cup U$,
otherwise let $V' := \{x\} \cup U'$, s.t. $\#\{x\} \cup U' = \frac{2}{3}\#V$.

End.

By this procedure we get the following result

Theorem 7. For any graph $G = (V, E)$ we can compute an endsegment V' of an MEO, s.t. $\#V' \leq \frac{2}{3}\#V$ and for each connected component C of $V \setminus V' \quad \#C \leq \frac{2}{3}\#V$, in time $O(\log n)$ by $O(n + m)$ processors. □

We remark that the procedure Endsegment is similar to P. Klein's procedure NONE [Kl 88], which computes an endsegment of a perfect elimination ordering.

3 A simple case.

We assume in this Section that V' and $V \setminus V'$ induce complete subgraphs. There are no restrictions on edges between V' and $V \setminus V'$.

For $v' \in V \setminus V'$ let $N'(v) := N(v) \cap V'$. For the computation of a chordal extension we can prove the following

Lemma 2. (compare also [NNS 87])
G is chordal iff for $v_1, v_2 \in V \setminus V'$, $\quad N'(v_1)$ and $N'(v_2)$ are comparable with respect to inclusion.

PROOF
$\Longrightarrow$: 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.

$\Longleftarrow$: We assume that G is not chordal.
Then a chordless cycle must be of the form v_1, v_2, w_1, w_2, s.t. $v_1, v_2 \in V \setminus V'$ 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 non chordal case the computation of a minimal chordal extension G' of G means the computation of a suitable enumeration (u_i), of $V \setminus V'$. Denote by $N''(u_i)$ the neighbourhood of u_i in V' w.r.t. G'.
Then $N''(u_i) = \bigcup_{j \leq i} N'(u_j)$.

This enumeration must have the *property M*:
If $v \in N''(u_i)$, but $v \notin N'(u_i)$, then there is a u_j, $j < i$, and a $w \in N(u_i)$, s.t. $v \in N'(u_j)$ and $w \in N'(u_i) \setminus N''(u_j)$.

Then the deletion of the edge vu_i induces a 4-cycle u_i, w, v, u_j.

Lemma 3.
If, for each i, $N'(u_{i+1}) \setminus \bigcup_{j \leq i} N'(u_j)$ is inclusion minimal for

$$\{N'(u_k) \setminus \bigcup_{j \leq i} N'(u_j) \mid k > i\},$$

then $(u_i)_i$ satisfies the *property M*.

PROOF
Consider any $v \in N''(u_l) \setminus N'(u_l)$. Let i be the minimum, s.t. $v \in N'(u_{i+1})$.

Since $N'(u_{i+1}) \setminus \bigcup_{j \leq i} N'(u_j) = N'(u_{i+1}) \setminus N''(u_i)$ is minimal for $\{N'(u_k) \setminus N''(u_i) \mid k > i\}$, w.r.t inclusion and $N'(u_{i+1}) \setminus N''(u_i) \neq N'(u_l) \setminus N''(u_i)$ (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 2, is clearly equivalent to the following computation problem:

Given a set system $\xi \subset P(V)$. Compute an enumeration A_i of ξ which satisfies the following *Property I*

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

Theorem 8. Under the assumption, that ξ is presented as a bipartite graph with n vertices and m edges, an enumeration satisfying the property I can be computed in 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:

Procedure Property $I \quad (\{A_i \mid i \in I\}, V, P)$

INPUT PARAMETER : A family $\{A_i : i \in I\}$ of subsets of V
OUTPUT PARAMETER : Sequence $P := (i_1 \dots i_I)$, which enumerates I

Begin

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

a)If $\# \bigcup_{i \in I_1} A_i \geq \frac{2}{3}\#V$, then
select $I_1' \subset I_1$, s.t.

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

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

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

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

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

Let A be an $i_2 \in I_2$, s.t. $\#A_i \setminus V'$ is minimal $V'' := V \setminus A$.

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

3)*Property* I $(\{B_i : i \in J_1\}, V', P_1)$ (if $J_1 > 1$)
Property I $(\{B_i : i \in J_3\}, V \setminus V', P_1)$ (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.

The correctness of this algorithm can be shown as follows:

Since all i, s.t. $A_i \in V'$ are in J_1 and all i, s.t. B_j, $j \in J_3$, induces an inclusion minimal $A_j \setminus V'' = A_j \setminus \bigcup_{i \in J_1 \cup J_2} A_i$.

Let $P_3 =< j_1 \dots j_k >$, s.t. $B_{j_{i+1}}$ is inclusion minimal $\{B_{j_k} \setminus \bigcup_{l<i} B_{j_l} : k > i\}$. But $B_{j_k} \setminus \bigcup_{l<i} B_{j_l} = A_{j_k} \setminus \bigcup_{j \le l} A_{j_l} \cup \bigcup_{j \in J_1 \cup J_2} A_j$.

This completes the correctness proof .

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

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

Sort I_1' w.r.t. $\#A_i$ in decreasing order $I_1' := \{i_1 \dots i_p\}$;

Compute for each v the least j, say $j(v)$, s.t. $v \in A_{i_j}$ 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 $O(\log n)$ time and $O(n+m)$ processors. Therefore 1a) can be executed in $O(\log n)$ time by $O(n+m)$ processors.

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

Since V' is a fixed set, the check for each i, whether $A_i \subseteq V'$, can be done in constant time by $O(n+m)$ processors. 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 time by $O(n+m)$ processors.

Since V' and V'' are constructed in that way, 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 $O(\log^2 n)$ time and $O(n+m)$ processors (note that J_1 and J_3 are disjoint, therefore the processor number needed for *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 $(\{B_i : i \in J_3\}, V' \setminus V'', P_3)$ resp..

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

□

4 The General Case.

Let C be any connected component of $G \downarrow V \setminus V'$. We assume that MEO is just executed on $G \downarrow V \setminus V'$.

Then we know a perfect elimination ordering of a chordal extension $\hat{G}$ of $G \downarrow C$ and therefore also a description of $\hat{G}$, as intersection graph of a collection $\mathcal{S}$ of subtrees of a tree T, say $(T, \mathcal{S})$ which can be computed in $O(\log n)$ time by $O(n^2)$ processors, if we just know the perfect elimination ordering (see [GH 88], [Kl 88]). Shortly we call $(T, \mathcal{S})$ a tree description of $\hat{G}$.

Moreover we may assume, that the vertices of T correspond to the cliques of G. On the other hand we assume, that $N(C) \cap V'$ has been made complete.

We proceed as follows:

Procedure Minchord $(G, C, V', \tilde{E})$

INPUT PARAMETER : $G, V' \subseteq V$, a connected component C of $V \setminus V'$

OUTPUT PARAMETER : A set of edges $\tilde{E}$

ASSUMPTIONS : $G \downarrow C$, $G \downarrow V'$ chordal

Begin

1)Compute a tree description $(T, \mathcal{S})$ for $G \downarrow C$ $(\mathcal{S} := \{S_v\ ,\ v \in C\})$.

2)Change and direct T in a suitable way to a root r or an root edge e_r
$\vec{T} :=$ *Direct* $(T, \mathcal{S})$
(the procedure *Direct* will be described later).

3)For each $v \in C$ let r_v be the root of S_v with respect to $\vec{T}$ (R_v may be e_r);
For $t \in V_T \quad t \neq r$, t not incident to e_r, let $SUC(t)$ be the unique successor (parent) of t in $\vec{T}$;
if t is incident to e_r let $SUC(t) := e_r$.

4)Let $N'(v) := N(v) \cap V'$ for $v \in C$;
let $\xrightarrow{*}$ be the reflexive and transitive closure if the directed edge set of $\vec{T}$;

$$\text{let } N''(t) := \bigcup_{r_v \xrightarrow{*} t} N'(v);$$

$$N'''(v) := N'(v) \cup \bigcup \{N''(t) \mid t \in S_v,\ t \xrightarrow{*}_{\neq} r_v\}$$
$$N'(v) \cup \bigcup \{N''(t) \mid t \in S_v,\ S(t) = r_v\}.$$

5)For each $t \in V_T$: Apply *Property I* $(\{N'''(v) : r_v = t\},\ V' \cap N(C), P_t)$;
Assume $P_t = (v_1^t \ldots v_{k_t}^t)$;
For each $i \in \{1 \ldots k_t\} : N^{(4)}(v_i^t) := \bigcup_{j \leq i} N'''(v_j^t)$;

6)$vw \in \tilde{E}$ iff $w \in N^{(4)}(v)$.

End

Also not knowing the procedure *Direct*, we can state the following:

Lemma 4. If $G \downarrow C$ is chordal, $G \downarrow V'$ is chordal and $V' \cap N(C)$ is complete, then $\hat{G} := (V \cup C, E \downarrow (V \cup C) \cup \tilde{E})$ is chordal.

PROOF Since $V' \cap N(C)$ is complete, at most two (but at least one) vertices of a chordless cycle of $\hat{G}$ belong to V'.

Let $(v_1 \dots v_k)$ be the sequence of vertices of C belonging to the chordless cycle d. Then the edges $v_i v_{i+1}$ belong all to different cliques C_i, which correspond to vertices t_v of $\vec{T}$. We may assume that the vertices t_i are consequentively on a path of T.

It is clear, that $t_1 \xrightarrow{*}_{\neq} r_{v_2}$ or $t_{k-1} \xrightarrow{*} r_{v_2}$ if $k > 2$. But then $v_2 w_2$ or $v_2 w_1$ is a chord of d.

If $k = 2$ and $r_v = r_{v_2}$, there is by the application of Property I a chord in d. If $r_{v_1} \xrightarrow{*} r_{v_2}$ then $N^{(4)}(v_1) \leq N^{(4)}(v_2)$ by construction, that means $v_2 w_1$ is a chord of d. □

We continue with a complexity analysis of *Minchord*

1) can be executed in $O(\log^2 n)$ time by $O(n^2)$ processors [Kl 88]

3) can be done by standard techniques in $O(\log n)$ time by $O(n^2)$ processors

4) $\xrightarrow{*}$ can be computed in $O(\log n)$ time by $O(n^2)$ processors using standard techniques.
   $N''(t)$ and $N'''(v)$ for each v can be computed in constant time by $O(nm)$ processors using standard techniques.

5) by Theorem 8 we can compute the sequence P_t by $O(n^2)$ processors in time $O(\log^2 n)$. $N^{(4)}$ can be computed in constant time by $O(n^2)$ processors .

Therefore up to step 2) the procedure *Minchord* needs $O(\log^2 n)$ time and $O(nm)$ processors (we ever can assume that $n \leq m + 1$). Since *Minchord* operates in its recursive application on disjoint subgraphs of G at the same time, the whole MEO-procedure needs $O(nm)$ processors and $O(\log^3 n)$ time under the assumtion that the procedure *Direct* needs this processor bound and $O(\log^2 n)$ time.

It remains to develop the procedure *Direct*

Procedure Direct $(T, \mathcal{S}, \vec{T})$

INPUT PARAMETERS : $T, \mathcal{S}$

OUTPUT PARAMETER : A directed tree $\vec{T}$

Begin

1)For each edge e of T let S_e be the set of subtrees passing e ;
Let $Sub := \{(e_1, e_2) : \mathcal{S}_{e_1} \subseteq \mathcal{S}_{e_2}\}$.
Let $Sub' := \{(e_1, e_2) : \mathcal{S}_{e_1} \subset_{\neq} \mathcal{S}_{e_2}\}$

2)For each edge $e := t_1 t_2$ of T, let $T^e_{t_1}$ be the connected component of $T \setminus (\{e\} \cup \{e' : (e', e) \in Sub'\})$, t_i belongs to;
let $\mathcal{S}^e_{t_1} := \{S_v \in \mathcal{S} | S_v$ does not pass e and $S_{v_1} \cap T^e_{t_i} \neq \emptyset\}$
$\tilde{N}^e_{t_i} := \bigcup\{N'(v) \,|\, S_v \in \mathcal{S}^1_{\cup}\}$;

3)Set $t_1 \rightarrow t_2 = (t_1, t_2) \in T' \Longleftrightarrow \#N^{t_1 t_2}_{t_1} < \#N^{t_1 t_2}_{t_2}$; let $T' \subseteq \vec{T}$;
direct each connected component $\tilde{T}$ of undirected edges to a root $r_{\tilde{T}}$ and add it to $\vec{T}$;

4)For $\mathcal{S}_{t_1 t_2} \subset_{\neq} \mathcal{S}_{t_3 t_4}$ let $t_1 \rightarrow t_2 \prec_i t_3 \rightarrow t_4$ iff $t_3 \rightarrow t_4$ in the connected component of t_i $(i := 1, 2)$ of $T \setminus \{t_1 t_2\}$;

5)For all $\vec{e} \in \vec{T}$ let $f(\vec{e})$ be the unique $\#\mathcal{S}_e$ maximal $\vec{e'}$, s.t. $\vec{e'} \prec_1 \vec{e}$ if $\vec{e'}$ exists, s.t. $\vec{e'} \prec_1 \vec{e}$ and let $M := \{\vec{e} : \nexists \hat{e}\, \vec{e} \prec_2 \hat{e}\}$;

6)For all $\vec{e} = t_1 \rightarrow t_2$ and for $i = 1, 2$, let $M^i_e := \{e' \in M : \mathcal{S}_{f(\vec{e'})} \not\supseteq \mathcal{S}_e, (e, f(\vec{e'})) \notin Sub)$, and $\mathcal{S}_e \subseteq \mathcal{S}_{e'}, ((e, e') \in Sub')\}$ (We shall see that M^i_e is connected.)
Compute the root r_e of M^i_e;

7)Replace $t_1 \rightarrow t_2$ by $r^1_e \rightarrow r^2_e$;

End.

A first result is the following

Lemma 5. Let T' be defined as in the procedure *Direct*. If $t \rightarrow t', t \rightarrow t'' \in T'$, then $\mathcal{S}_{tt'}$ and $\mathcal{S}_{tt''}$ are comparable by inclusion and unequal.

PROOF Assume that $\mathcal{S}_{tt'}$ and $\mathcal{S}_{tt''}$ are incomparable or equal by inclusion. Denote tt' by e' and tt'' by e''.

Assume that $\tilde{t}$ belongs to $T^{e'}_{t'}$. Then there is a path from $\tilde{t}$ to t', s.t. all its edges $\tilde{e}$ do not belong to $Sub'(e') := \{\tilde{e};\ Sub'(e', e)\}$.

Since between the edges $\tilde{e}$ and e'' ever is e', $\tilde{e}$ does not belong to $Sub'(e'')$. Since $\mathcal{S}_{e'}$ and $\mathcal{S}_{e''}$ are incomparable by inclusion or equal, e' does not belong to $Sub'(e'')$. Therefore each edge $\tilde{e}$ as above belongs to $S_t^{e''}$.

Hence $N_{t'}^{e'} \subseteq N_t^{e''}$. Therefore $\#N_t'^{e'} \leq \#N_t^{e''} < \#N_{t''}^{e''}$.

By the same arguments we also can prove the unequation via versa. That is a contradiction.

□

Remark. For the case, that for one of the edges e', e'' is undirected, we also get a contradiction.

Changings in step 7) do not destroy connectedness of the subtrees S_v by construction of the roots r_v^i.

M_e^i is connected: otherwise there would exist an $\vec{e'} \in M_e^i$, s.t. $\mathcal{S}_{f(\vec{e'})}$ contains e. This is a contradiction. No $\vec{e}_1, \vec{e}_2 \in M_e^i$ is of the form $t_1 \to t_2$, $t_1 \to t_2'$. Otherwise we can assume, that $\mathcal{S}_{e_1} \subseteq \mathcal{S}_{e_2}$ and therefore $\mathcal{S}_e \subseteq \mathcal{S}_{e_1} \subseteq \mathcal{S}_{f(\vec{e'})} \subseteq \mathcal{S}_{e_2}$.

This is a contradiction. All $\mathcal{S}_e$ in M_e^i are pairwise incomparable, since $M_e^i \subset M$. Therefore M_e^i is directed to a root r_v^i and $\vec{T}$ restricted to M_e^i is not changed in the whole procedure.

The procedure guarantees for $\mathcal{S}_{t_1 \to t_2} \subseteq \mathcal{S}_{t_1 \to t_3} = \mathcal{S}_e$, that $e = t_1 \to t_2$ is shifted to the t_3-side of $\vec{T}$, w.r.t. $e' := t_1 \to t_3$, and therefore $\vec{T}$ becomes directed to a root.

The shift of t_2 to $r_{\vec{e}}^2$ guarantees that the target of e is shifted to the root of $\bigcap\{S_v : S_v \in \mathcal{S}_e\}$.

We shall show that $\vec{T}$ is a suitable directed tree to get a minimal chordal extension:

We differ between edges $vw \in E_1'$,s.t. $w \in N'''(v)$, and edges $vw \in E_2'$, which are generated by the procedure *Property I*.

We still proved in section 3, that erasing an edge e of E_2' generates an induced cycle of length four.

We have to prove the same for edges of E_1'.

Assume $vw \in E_1'$, but $vw \notin E$. That means $w \in N'''(v)$, but $w \notin N'(v)$. Then there is a t, s.t. $t \rightarrow r_v \in \vec{T}$ and $w \in N'(t)$. That means there is a V', s.t. $r_{v'} = t$ and $w \in N'''(v')$. Since we find a v', s.t. $w \in N'(v)$ or we have $w \in N''(t)$.

The shift of the target of e to $r_{\vec{e}}^2$ ensures that for no edge $\vec{e} = t_1 \rightarrow t_2$, s.t. $t_2 \xrightarrow{*} t$ we have $\mathcal{S}_e \subset_{\neq} \mathcal{S}_{tr_v}$. Therefore $N''(t) = \tilde{T}_t^{tr_v}$.

Since $t \rightarrow r_v$ we have $\#\tilde{N}_{r_v}^{tr_v} \geq \#N''(t)$.

i) Assume $w \in \tilde{N}_{r_v}^{tr_v} \cap N''(t)$.
Then we find a $\mathcal{S}_{v''} \in \mathcal{S}_{r_v}^{tr_v}$, s. t. $w \in N'(v'')$.
Since $S_{v''}$ is on a $\vec{T}$-path, s.t. none of its edges $\hat{e} \quad S_{\hat{e}} \subset S_{tr_v}$, we find a path p from v'' to v, s.t. for none of its vertices $\neq v, \hat{v}$, $S_{\hat{v}} \in S_{tr_v}$

Therefore the concatenation (v', v) with p, denoted by $v \frown p$ is **chordless**. Therefore there must be a chord vw in the cycle $v' \frown p \frown w \frown v'$, moreover there are chords between w and all vertices of $v' \frown p$.

Therefore the deletion of vw induces a chordless cycle of length four.

ii) Assume now $w \in N''(t) \setminus \tilde{N}_{r_v}^{tr_v}$.
Since $\#N''(t) \leq \#\tilde{N}_{r_v}^{tr_v}$, there is a vertex $w' \in \tilde{N}_{r_v}^{tr_v} \setminus N''(t)$, and therefore a $v'' \in \mathcal{S}_{r_v}^{tr_v}$, s.t. $w' \in N'(v'')$. By the same arguments as in i) we get a chordless path $v' \frown p$ from v' to v'', s.t. p begins at v, since $w' \notin N''(t)$.

We have **no** edge $v'w'$. But also here we have a cycle $cyc := v' \frown p \frown w' \frown w \frown v'$.

Let $p = v, v_1, \ldots, v_k, v''$. In a chordal extension of cyc there is an edge vw' or an edge wv_1.

In both cases the deletion of vw induces a chordless cycle of length four.

Therefore we have the following

Theorem I. *Minchord* together with the subprocedure *Direct* as above computes for each graph an inclusion minimal chordal extension and therefore an MEO of G.

The last to do is a complexity analysis of *Direct*.

The number of T-edges and vertices is bounded by n.

We can therefore execute step 1) on constant time by $O(n^3)$ processors. It can also be done by $O(nm)$ processors in $O(\log n)$ time, computing the connected components of $G \setminus \{v \ : \ S_v \in \mathcal{S}_e\}$ and setting $(e', e) \in Sub$ iff e' joins two different connected components. In step 2) the computation of a connected component of one tree needs $O(n)$ processors and $O(\log n)$ time. The computation of all $T^e_{t_i}$ needs $O(n^2)$ processors and $O(\log n)$ time. The computation of $S^e_{t_i}$ needs constant time and $O(n^2)$ processors. The computation of all $\tilde{N}^e_{t_i}$ needs $O(nm)$ processors and constant time.

The comparison in step 3) needs $O(\log n)$ time and $O(n)$ processors. The computation of the connected components $\vec{T}$ needs $O(n)$ processors and $O(\log n)$ time. The direction of these connected components needs $O(\log n)$ time and $O(n)$ processors.

Step 4) needs $O(\log n)$ time and $O(n^2)$ processors for the computations of $\prec_i$.

In step 5) the computation of $f(\vec{e})$ can be done in $O(\log n)$ time by $O(n^2)$ processors. M can be computed in $O(\log n)$ time by $O(n^2)$ processors.

In step 6) the computation of M^i_e can be done in $O(\log n)$ time by $O(n^2)$ processors. The computation of r^i_e needs $O(n^2)$ processors and $O(\log n)$ time.

Step 7) can be done in constant time by $O(n)$ processors.

Therefore we can state our main Theorem:

Theorem II. Given an arbitrary n-vertex, m-edge graph G, there exists a parallel algorithm for computing an MEO of G working in $O(\log^3 n)$ parallel time and $O(nm)$ processors on a CRCW PRAM.

PROOF: We recall the beginning of the global strategy in Section 2. The first step could be done in time $O(\log n)$ by $O(nm)$ processors. The second step was a recursion step on disjoint subgraphs of G. The third step was the application of the procedure *Minchord*. This can be done in $O(\log^2 n)$ time and $O(nm)$ processors. Therefore the whole algorithm needs $O(\log^3 n)$ time. We know by the remark on step 1) that the processor bound of $O(nm)$ for the recursive application of step 1) is also true. In the procedure *Minchord* the only step using explicitely $O(nm)$ processors is step 1) of the subprocedure *Direct* (in all other steps we have a processor bound of $O(n^2)$). But n connected component problems on subgraphs of $(V', (\tilde{E}) \cup E)$ can be handeled also as n connected component problems on subgraphs of (V, E), then the processor bound of this step of $O(nm)$ is also preserved. □

5 Applications.

We summarize some applications of our parallel MEO-algorithm. We refer to [Ro 73], [Ta 85], [Kl 88], [DK 88a] for fundamentals, and to [GH 88]. By sparse Gaussian elimination we mean a Gaussian Elimination where the set of nonzero entries in the whole elimination procedure is inclusion minimal [Ro 73], [OCF 76].

Theorem III. There exist parallel algorithms working in $O(\log^3 n)$ parallel time and $O(nm)$ processors on a CRCW PRAM for the following problems.

1. Constructing an MEO and a Minimal Fill-In of an arbitrary graph.
2. Constructing a Minimal Fill-In for arbitrary graph.
3. Breadth-First Search Problem (alternative to [GM 87] for sparse graphs).
4. Sparse Gaussian Elimination on Symmetric Matrices.

PROOF OF THEOREM III: The first statement has been proved in Theorem II. A Minimal Fill-In problem is reducible to the MEO problem in $O(\log^2 n)$ parallel time and $O(n^2)$ processors on a CRCW PRAM ([Ta 85], [GH 88]). In (3) we proceed similar as in in the case of chordal graphs [Kl 88]:
We can assume that an MEO $<$, is known. For each vertex $x \neq max_< V$ there is a vertex y, s.t. $x < y$ and $xy \in E$. We choose for each vertex x a largest vertex $f(x)$, s.t. $x\,f(x) \in E$. Setting $T = \{(x, f(x)) \,|\, x \in V \setminus \{max_< V\}\}$, we have a breath first search tree, because by construction there is no edge $xy \in E$, s.t. $y = F^n(x)$ for some $n \in I\!N$. Hereby the third statement is proved.

By application of the algorithm of Gilbert and Hafsteinsson [GH 88] to the constructed MEO we obtain the fourth statement. □

6 Further Research.

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

1. Is there a way to improve our algorithm in the number of processors ($O(nm)$) giving the MEO - this way - 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 has asked us 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 III 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 it 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^{th} ACM STOC (1987), pp. 325–334.

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

[Co 86] Cole, R., *Parallel Merge Sort*, 27^{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. and Vishkin, U. *A Complexity Theory for Unbounded Fan-In Parallelism*, 23^{th} FOCS (1982), pp. 1-13.

[DK 86] Dahlhaus, E., and 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., and 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., and Karpinski, M. *Fast Parallel Decomposition by Clique Separators*, Research Report No. 8525-CS, University of Bonn (May 1988).

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

[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., and Wyllie, S. *Parallelism in Random Access Machines*, Proc. 10th 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.

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

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

[GRE 84] Gilbert, J., Rose D. and 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. and 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 and 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 (1988).

[Kl] Klein, Ph. *Personal communication*

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

[NNS 87] Naor, J., Naor, M., and Schäffer, A. *Fast Parallel Algorithms for Chordal Graphs*, Proc. 19th 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. and 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., and 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. and 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.

[TY 84] Tarjan, R.E., and Yannakakis, M., *Simple Linear Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Graphs, Test Acyclicity 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 Acyclicity of Graphs, Test Acyclicity 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.