Fast Parallel Computation of Perfect and Strongly Perfect Elimination Schemes

ELIAS DAHLHAUS

AND

MAREK KARPINSKI

DEPT. OF COMPUTER SCIENCE
UNIVERSITY OF BONN

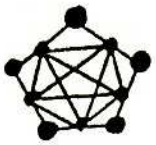
Abstract.

We design fast parallel algorithms for the construction of perfect and strongly perfect elimination schemes (orderings) for chordal and strongly chordal graphs. In the case of chordal graphs the algorithm works in $O(\log^2 n)$ parallel time and $O(n^4)$ processors on a CREW PRAM, an improvement over the recent algorithms of [NNS] and [DK]. The parallel algorithm for strongly perfect elimination schemes works in $(\log^4 n)$ time and $O(n^4)$ processors.

Section 1: Introduction

Chordal graphs are graphs which have no induced cycle greater than 3. *Strongly chordal graphs* have additionally no *trampoline* or *sun* as an induced subgraph. A *sun* or *trampoline* consists of an independent set S, a clique T, both of cardinality k, and a $2 \cdot k$-cycle C which alternates between S and T.

Trampoline:



Several authors ([Go], [Fa 1]) gave characterizations of (strongly) chordal graphs by the existence of so-called *(strongly) perfect elimination orderings.*

Definition: Let $G = (V, E)$ be a graph.

i) A total ordering $<$ is a *perfect elimination ordering* or a *perfect elimination scheme* on G iff for any x, y, z, s.t. $[x, y], [x, z] \in E$ and $x < y, x < z$, we have $[y, z] \in E$.

ii) A perfect elimination ordering $<$ is called a *strongly perfect elimination ordering* or *scheme* iff for any x, y, x', y', s.t. $[x, y], [x, y'], [y, x'] \in E$ and $x < x', y < y'$, we have $[x', y'] \in E$.

Theorem 1 (Elimination Theorem): ([Go], [Fa 1]) G is (strongly) chordal iff G has a (strongly) perfect elimination scheme.

It is well known that (strongly) perfect elimination orderings can be found in polynomial time. Our aim is to find such orderings in parallel. For chordal graphs the NC^2-algorithms have just been published ([NNS], [DK]). The important subroutines of these algorithms were to determine the set of all the cliques. We shall give a new parallel algorithm, computing all the cliques of a chordal graph, which needs less processors than that of [NNS] and [DK], and a parallel algorithm computing a strongly perfect elimination ordering of a strongly chordal graph. In section 2, we give some foundations on chordal and strongly chordal graphs. We also introduce parallel complexity classes. Section 3 presents the new parallel algorithm for the computation of all the cliques of a chordal graph. For strongly chordal graphs, we can speed up and simplify this algorithm to AC^0. Section 4 describes the algorithm and the subroutines determining strongly perfect elimination orderings.

2. Basic Definitions and Results

2.1. Parallel complexity

We use two models of parallel computation which are equivalent in time and space up to a polylog factor. The first model is that of (logspace-) uniform sequences of switching circuits with bounded fan-in [Co]: NC^k is the class of all problems computable by a uniform sequence of switching circuits of depth $O(\log^k n)$ (it corresponds to the time) and of polynomial size (the size corresponds to the number of processors). $NC = \bigcup_{k=1}^{\infty} NC^k$.

AC^0 is the class of all problems computable by a uniform sequence of *unbounded* fan-in circuits of polynomial size and constant depth. Most reductions to NP-complete or P-complete or NC-complete problems are not only polynomial. They are local replacement reductions. Most local replacements are AC^0. Clearly $AC^0 \subseteq NC^1$.

The second model of parallel computation is that of concurrent read/exclusive write parallel random access machines (CREW-PRAM). A tape can be read by many proces-

sors at the same time, but only one processor can write into the same tape. For more details on parallel random access machines we refer to [Gl].

2.2. Additional features of chordal graphs

Here we give a characterization of chordal graphs which is quite important for the whole paper.

Theorem 2 (Tree Characterization Theorem): [Bu],[Ga] A graph G is chordal iff there is a tree T and a collection S of subtrees of T, s.t. G is isomorphic to the vertex intersection graph of S. Moreover, any vertex t of T corresponds to a clique of G in the following sense: The clique represented by t contains exactly all $s \in S$, s.t. $t \in s$.

[NNS] and [DK] presented a parallel algorithm to compute such a tree representation.

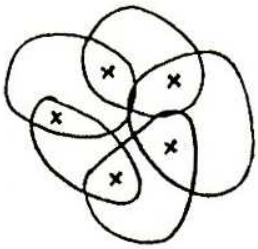
Theorem 3 (Tree Computation Theorem): [NNS] If the set C of cliques of a chordal graph G is a part of the input, then a subtree representation (T, S) of G can be computed by a CREW-PRAM in time $O(\log^2 n)$ by $O(n^3)$ processors.

2.3. Foundations on Strongly Chordal Graphs

Let G be a graph. Instead of G we can also consider the ***hypergraph*** H_G of its ***cliques*** (maximal complete subgraphs). Now the following result is known (See also [BDS] and [BP]):

Theorem 4 (Acyclicity Theorem): A graph G is strongly chordal if and only if its clique hypergraph H_G is *β-acyclic* [BFMY]. A hypergraph H is *β-acyclic* iff there is no *β-cycle* $v_0, h_1, v_2, h_2, \cdots, h_{i-1}, v_i = v_1$, that means:

i) the v_i are vertices
ii) the h_i are hyperedges
iii) $v_j \in h_{j-1 (\mathrm{mod}\ i)} \cap h_j$ and $v_j \notin h_k, k \neq j, j-1 (\mathrm{mod}\ i)$.



β-cycle (see [BMFY])

In the case of chordal and strongly chordal graphs we have at most as many cliques as vertices. H_G can be determined from G in polynomial time (see [Go]). It is also

known that it can be determined in NC^2 [NNS]. For strongly chordal graphs we shall see that there is a much simpler solution in AC^0.

Now we want to translate strongly perfect elimination orderings on strongly chordal graphs to elimination orderings on the cliques. First, we state a simple auxiliary result:

Lemma 1 (Intersection Lemma).: Let H be a β-acyclic hypergraph. Then each hyperedge intersection is the intersection of at most two hyperedges.

PROOF OF THE INTERSECTION LEMMA: Consider cliques C_0, C_1, C_2. Assume the statement of the intersection lemma is not true for these cliques. Then there are x_0, x_1, x_2, s.t. $x_i \in C_i \cap C_{i+1(mod3)} \backslash C_{i+2(mod3)}$. This defines a β-cycle. Second, we need the following result:

Lemma 2 (Clique Lemma). [Di] Let $<$ be a (strong) perfect elimination scheme; then $\{y | [y,x] \in E|, x < y\}$ is complete.

Therefore

Lemma 3 (First Element Lemma). No two cliques of a chordal graph have the same first element with respect to a perfect elimination scheme. Therefore we can now define an ordering on the hyperedges:

$h_1 <_H h_2$ iff the first element of $h_1$is smaller than the first element of h_2.

Let $N(x) := \{y | y = x \text{ or } [y,x] \in E\}$ be the *neighborhood* of x. The following result is due to Farber [Fa 1]:

Lemma 4 (Neighborhood Comparability Lemma). Let G be strongly chordal and $<$ be a strong perfect elimination scheme.

Let $N_x(y) := \{v \in N(y) | x \leq v\}$.

Then $\{N_x(y) | y \in N_x(x)\}$ is comparable by inclusion. For the hyperedges we get the following

Corollary 1: Let $<_H$ be the corresponding ordering on the cliques to a strong perfect elimination scheme.

Then $\{h \cap h' | h' >_H h\}$ is pairwise comparable by inclusion.

PROOF: Assume $h_1, h_2 >_H h$. Assume $h_1 \cap h$ and $h_2 \cap h$ are incomparable. We can find an $x_1 \in h_1 \backslash h$ and a $y_2 \in h_2 \cap h$, s.t. x_1 and y_2 are not in a common clique (or joined by an edge). Otherwise h_1 would not be a maximal complete set. By the same argument we also find an $x_2 \in h_2 \backslash h$ and a $y_1 \in h_1 \backslash h$, s.t. x_2 and x_1 are not joined by an edge. Clearly $y_2 \notin h_1$ and $y_1 \notin h_2$. Let x be the least element of h. Clearly $x_1, x_2, y_1, y_2 > x$. But $N_x(y_1)$ and $N_x(y_2)$ are incomparable because $x_1 \in N_x(y_1) \backslash N_x(y_2)$ and $x_2 \in N_x(y_2) \backslash N_x(y_1)$. □

Therefore $<^H$ satisfies the following *strong elimination property* for hyperedges $h_1 <_H h_2, h_3 \Longrightarrow h_1 \cap h_2 \subset h_1 \cap h_3$ or $h_1 \cap h_3 \subset h_1 \cap h_2$. We call such orderings *strongly perfect elimination* schemes as well. Now we have to show that a strongly perfect elimination scheme on the cliques also induces a strongly perfect elimination scheme on the vertices.

Let $<$ be any ordering on the vertices and $<_H$ be a strongly perfect elimination schemes on the cliques. Let h_x be the greatest clique h with respect to $<_H$, s.t. $x \in h$. Define $x <_G y$ iff $h_x < h_y$ or $h_x = h_y$ and $x < y$.

Lemma 5 (Correspondence Lemma). $<_G$ is a strongly perfect elimination scheme.

PROOF: trivial.

Corollary 2: For each strongly perfect elimination ordering on the cliques we can compute a strongly perfect elimination ordering on the vertices in AC^0 (and therefore in NC^1) and vice versa.

PROOF: This can be checked from the fact that each first-order-statement on finite models can be checked in AC^0.

3. Computing the Set of Cliques of a Chordal Graph

Consider any chordal graph $G = (V, E)$ and its tree representation s.t. the vertices of the tree T correspond to the cliques.

Lemma 6.

i) Each clique represented by a leaf of T is the corresponding clique to a simplicial vertex.

ii) Each clique c corresponding to a T-vertex of degree 2 (shortly degree 2-cliques) is generated by two vertices of c.

PROOF:

i) trivial;

ii) Let c_1, c_2 be the immediate neighbours of c in T. Then we find $x_1 \in c$, s.t. $x_1 \notin c_1$ and $x_2 \in c$, s.t.$x_2 \notin c_2$. But then $\{y | [y, x_1], [y, x_2] \in E\} = c$. □

Another property of degree-2-cliques is the following

Observation 1: Each degree-2-clique has only two maximal intersections with other cliques. To check the property without knowing all the cliques, we obtain the following

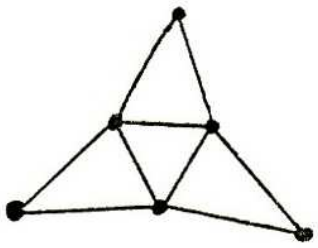
Observation 2: For each clique c_1 and each $x \in c_1 \backslash c, c_1 \cap c \subseteq N(x) \cap c$. Therefore each maximal clique intersection is of the form $N(x) \cap c$.

Two 1-generated or 2-generated cliques can be compared easily:

Lemma 7. Let $c(x, y)$ be the clique generated by x and y. Then $c(x_1, y_1) = c(x_2, y_2)$ iff $x_1, y_1 \in c(x_2, y_2)$.

PROOF: trivial.

Definition: The *Hajos graph* is nothing else than the trampoline, consisting of an independent set and a clique both of size 3:



Historically it is the first known chordal graph not being an interval graph.

Theorem 5: Each chordal graph not having the Hajos graph as an induced subgraph only has cliques generated by one or two vertices.

PROOF: Consider any clique c not generated by any pair of vertices and a clique c_1 of maximal intersection with c. Consider $x_1 \in c \cap c_1$, $x_2 \in c \backslash c_1$. Consider any clique c_2, s.t.$x_1, x_2 \in c_2$, s.t. $c_2 \cap c$ maximal and $c_1 \cap c_2 \cap c$ maximal. $c_1 \cap c_2 \cap c \neq c_1 \cap c$ because of maximality conditions. Pick up any $x_3 \in c_1 \cap c \backslash c_2$ and consider a maximal intersecting clique c_3, s.t. $x_2, x_3 \in c_3$. Because of maximality conditions on c_1 and c_2, we have $c_1 \cap c_2 \cap c \notin c_3 \cap c$. Let $x_1' \in c_1 \cap c_2 \cap c \backslash c_3$. Then x_1', x_2, x_3 together with c_1, c_2, c_3 form a β-cycle of length three. One can find $y_i \in c_i$, s.t. $[y_i, y_j] \notin E$ for $i \neq j$. Let $y_i \in c_i \backslash c$ for $i = 1, 2, 3$. Let $[y_i, y_j] \in E$. Consider any $\tilde{x_i} \in c_i$, $\tilde{x_j} \in c_j$ and $x_i \notin c_j, x_j \notin c_i$. Then y_i, y_j, $\tilde{x_j}$, $\tilde{x_i}$ form a cycle. But then there is an edge $[x_i, y_j]$ or $[x_j, y_i]$. W.l.o.g. $[x_i, y_j] \in E$. But then $c_j \cap c \subsetneq N(y_j) \cap c$ and therefore c_j is not maximally intersecting. The induced subgraph of $\{x_1', x_2, x_3, y_1, y_2, y_3\}$is a Hajos graph. □

Corollary 3: It is possible to compute all the cliques of a Hajos-graph-free chordal graph by a CREW-PRAM in $O(\log n)$ time by $O(n^4)$ processors. This is especially true for strongly chordal graphs.

PROOF: We state an algorithm:

1. Compute for each $[x, y] \in E$ $N(x, y) := \{u | [u, x], [u, y] \in E\}$ and check whether $N(x, y)$ is complete ($O(n^4)$ processors and $O(\log n)$ time).
   Say
$$c(x, y) = \begin{cases} N(x, y), \text{ if } N(x, y) \text{ is complete} \\ 0 \text{ otherwise} \end{cases}$$

2) For each $[x, y], [z, w] \in E$ check whether $C[x, y] = C[z, w](O(n^4)$ by the last lemma and $O(\log n)$ time).

3) Eliminate duplicates ($O(n^3)$ processors and $O(\log n)$ time or $O(n^2)$ processors and $O(\log^2 n)$ time by sorting ([Hi],[Cl]). □

The corollary means that we can extend the result of [NNS] for the special case of Hajos-graph-free chordal graphs in time and in processors. Now we want to have a closer look at the general case of chordal graphs.

Theorem 6: It is possible to compute the collection of all cliques of any chordal graph by a CREW-PRAM by $O(n^4)$ processors in $O(log^2 n)$ time.

Before we state an algorithm, we have a look at the structure of vertices appearing in any non-1-or 2-generated clique: A first observation is the following

Lemma 8. Each vertex of a chordal graph $G = (V, E)$ appears in any 1- or 2-generated clique.

PROOF OF LEMMA: Consider any vertex $x \in V$ and a tree representation $T_x \subseteq T$ of x. Let c be a clique which is a leaf of T_x. We can assume that there are at least two

cliques in which x appears. Let c_1 be the immediate neighbour of c in T, s.t. $x \in c_1$ and $y \in c \backslash c_1$. Then c is generated by y and x. □

The key lemma for an algorithm is the following

Lemma 9. If $x \in c$, s.t. c is not generated by one or two vertices, then

i) there is an edge $[x, y] \in E$, s.t. there is no 1- or 2-generated clique $c(u_1, u_2)$, s.t. $x, y \in c(u_1, u_2)$, or

ii) there are y, z, s.t. $[x, y], [x, z], [y, z] \in E$ but there is no 1- or 2-generated clique $c(u_1, u_2)$ s.t. $x, y, z \in c(u_1, u_2)$, or

iii) there are 1- or 2-generated cliques c_1, c_2, c_3, s.t.
   1) $c_i \cap c_j$ are pairwise incomparable by inclusion
   2) $x \in c_1 \cap c_2 \cap c_3$.
   We call such a triple a 3-cycle.

PROOF OF LEMMA: We assume that i) and ii) are not the case. Assume $x \in c_1$, s.t. c_1 is 1- or 2-generated and $c_1 \cap c$ is a maximal by inclusion. Assume $y' \in c \backslash c_1$ Then by the assumption that i) is not satisfied, that there is a 1- or 2-generated c_2', s.t. $x, y' \in c_2'$. We can assume that $c_2' \cap c$ is maximal. Moreover, we can find a maximal $c_2 \cap c$, s.t. $c_1 \cap c_2 \cap c$ is a maximal intersection of 1- or 2-generated c_1, c_2 and the non-1- or 2-generated clique c, s.t. $c_1 \cap c$, $c_2 \cap c$ are maximal. Assume $y \in c_1 \backslash c_2$, $z \in c_2 \backslash c_1$. Then we find a 1- or 2-generated clique c_3, s.t. $x, y, z \in c_3$ by the assumption that ii) is not satisfied. Assume that also $c_3 \cap c$ is maximal. But by the maximality of $c_1 \cap c_2 \cap c$ we find a $w \in c_1 \cap c_2 \cap c \backslash c_3$. Therefore $c_i \cap c_j$ are pairwise incomparable by inclusion. □

A last useful result for the complexity analysis is the following

Lemma 10. Let x be a vertex, s.t. for all cliques c, s.t. $x \in c$, c has in T a degree of at most two. Then x does not satisfy the consequences of the previous lemma.

PROOF: The cliques of T_x form an interval and are therefore α-acyclic. Therefore there cannot be a 3-cycle in T_x and all cliques of T_x are 1- or 2-generated by Observation 1.

Now we can state an algorithm:

Input: a chordal graph $G = (V, E)$ $(V', E') := G$, $C := 0$

Repeat:

1) Compute the set $D := \{c(x_1, x_2) | c(x_1, x_2)$ is the clique generated by x_1, x_2; $x_1, x_2 \in V\}$ and identify equal cliques (this requires $O(n^4)$ processors and $O(\log n)$ time on a CREW-PRAM).

2) Set

$$\text{Dua} := \{[x, y] \in E | \forall c \in D(x \notin c \vee y \notin c)\}$$

Tria $:= \{(x,y,z) | [x,y],[y,z],[z,y] \in E$ and $\forall c \in D(x \notin c \vee y \notin c \vee z \notin c)\}$
(this requires $O(n^4)$ processors and $O(\log n)$ time on a CREW-PRAM).

3) 3-cyc $:= \{(c_1,c_2,c_3) \in D^3 : \exists z \in c_1 \cap c_2 (z \notin c_3) \vee \exists z \in c_2 \cap c_3 (z \notin c_1) \vee \exists z \in c_3 \cap c_1 (z \notin c_2)\}$
(this can also be computed by $O(n^4)$ processors and $O(\log n)$ time on a CREW-PRAM).

4) $V := \{x \in V' : \exists y \in V [y,x] \in$ Dua $\vee \exists y,z \in V((x,y,z) \in$ Tria $) \vee \exists c_1,c_2,c_3 \in D((c_1,c_2,c_3) \in$ 3-cyc $\wedge x \in c_1 \cap c_2 \cap c_3)\}$
(This can be computed by $O(n^4)$ processors and in $O(\log n)$ time by a CREW-PRAM).

$G' := (V', E' \restriction V')$

5) $D' := \{c \in D_i \not\exists c' \in c (c \subseteq c')\} : C := C \cup D'$
until $V' := 0$.

Clearly one step of the loop can be executed by $O(n^4)$ processors in $O(\log n)$ time.

Claim: The loop is repeated at most $O(\log n)$ time.

PROOF OF THE CLAIM: Consider again the tree representation of G. Then at each step all vertices x, s.t. T_x appears only on branches of T, vanish and therefore all branches of T do so. But one only needs $O(\log n)$ branch deletions until one reaches the empty set. This completes the proof of Theorem 6. □

Conclusion: To compute a perfect elimination scheme we continue in the same way as [NNS] did. We compute a collection of subtrees of a tree representing the chordal graph G and from that we compute the perfect elimination scheme.

4. Testing Strong Chordality

At first we shall give a parallel algorithm which tests the β-acyclicity of a hypergraph. Afterwards we present an algorithm which tests the strong chordality of a graph which uses the first algorithm as a subroutine. We begin with a lemma which is related to the chordality test in [TY 2] or [NNS].

Lemma 11 (Lemma of β- Acyclicity Test).

A hypergraph is β-acyclic iff for each $x_1, x_2 \in H$, s.t. $x_1 \cap x_2 \neq 0$, after the deletion of all vertices of $x_1 \cap x_2$ and all hyperedges $h \neq x_1, x_2$, s.t. $x_1 \cap x_2 \subseteq h$, x_1 and x_2 are in different connected components.

PROOF: "$\Longleftarrow$" Suppose (V,H) is not β-acyclic. Then there is a cycle $x_1, x_2, x_k = x_1$, s.t. all $x_i \cap x_{i+1}$ are pairwise incomparable by inclusion. We assume that the cycle is of minimal legth. Then no intersection $x_i \cap x_{i+1} \subset x_j$, $j \neq i, i+1$. But then x_i, x_{i+1}

remain in the same connected component also after the deletion of all $v \in x_i \cap x_{i+1}$ and all y, s.t. $x_i \cap x_{i+1} \subset y \neq x_i, x_{i+1}$.

"$\Longleftarrow$": Suppose x_1 and x_2 are in the same component after the deletion of $x_1 \cap x_2$ and all y, s.t. $x_1 \cap x_2 \subseteq y \neq x_1, x_2$. Consider a shortest path $x_1 = y_1, y_2, \cdots, y_k = x_2$. Then the intersections $y_i \cap y_{i+1}$ are pairwise incomparable. $y_i \cap y_{i+1} \not\subseteq x_1 \cap x_2$ because all vertices of $x_1 \cap x_2$ are deleted, $x_1 \cap x_2 \not\subseteq y_i \cap y_{i+1}$, because $x_1 \cap x_2 \not\subseteq y_i$ for $i \neq 1, k$.

□

Theorem 7 (β-Acyclicity Test Theorem):

i) β-acyclicity is in $CoNL \subseteq NC^2$.

ii) β-acyclicity can be tested by a CREW-PRAM in time $O(\log^2 n)$ using $O(n^4)$ processors, where $n = max$ (number of vertices, number of hyperedges)

PROOF:

i) follows from the previous theorem.

ii) Let $h_1 \; E' h_2 :\Longleftrightarrow h_1 \cap h_2 \; eq0$. To test the acyclicity is nothing else than m^2 times to compute connected components of an indirected subgraph $G'_{x,y}$ of (H, E'). Knowing V_1, V_2, all $G'_{x,y}$ can be generated simultaneously in constant time by n^4 processors. To compute connected components on one $G'_{x,y}$, we need $O(\log^2 n)$ time and $O(n^2)$ processors ([Hi],[CV]). Therefore we need $O(n^4)$ processors and $O(\log^2 n)$ time to compute all connected components of all $G'_{x,y}$ simultaneously. Therefore β-acyclicity can be tested in $O(\log^2 n)$ time by $O(n^4)$ processors.

To test the strong chordality of a graph and to build up the corresponding clique-hypergraph, we use the following algorithm:

Input: $G = (V, E)$ and an order $\lesssim$ on E

Step 1) Select all pairs $[x, y]$, generating a clique. $C := \{[x, y] | [x, y]$ generates a clique $\}$.

Step 2) If $[x, y], [z, w]$ both generate a clique and $[x, z], [x, w], [y, z], [y, w] \in E$, then identify the clique generated by $[x, y]$ and the clique generated by $[z, w]$, say $[x, y] \sim [z, w]$.

Step 3) For each $[x, y] \sim [z, w]$, s.t. $[x, y] \lesssim [z, w]$, delete $[z, w]$ from C.

Step 4) If $|C| > |V|$ then *reject.*

Step 5) If there is an edge $[x, y]$, s.t. for all $[z, w] \in C$ $\{x, y, z, w\}$ is not complete, then *reject.*

Step 6) Test $(V, H := \{\{x | [x, u], [x, v] \in E\} | [u, v] \in C\})$ on $\beta-$ acyclicity.

Processor and Time Analysis:

Step 1): $C := \{[x, y] | \forall z, w([z, w(], [z, y], [w, x], [w, y]) \in E \rightarrow [z, w] \in E\}$ can be computed in $O(\log n)$ time by $O(n^4)$ processors.

Step 2): can be executed in constant time by $O(n^4)$ processors.

Step 3): can be done by $O(n^4)$ processors in $O(\log n)$ time.

Step 4): $|C|$ can be computed from C in $O(\log n)$ time by $O(n^2 \cdot \log n)$ processors.

Step 5): can be executed by $O(n^4)$ processors in $O(\log n)$ time.

Step 6): β-Acyclicity can be tested in $O(\log n)$ time by n^4 processors (see β-Acyclicity Test Theorem).

Therefore strong chordality can be tested in $O(\log^2 n)$ time by $O(n^4)$ processors.

5. An Algorithm for the Computation of a Strongly Perfect Elimination Scheme

After we stated an NC-algorithm for the recognition of strongly chordal graphs, it is straightforward to look for the problem of the computation of a strongly perfect elimination scheme. We cannot repeat the program of [NNS], for we don't have tree structures which represent exactly the class of strongly chordal graphs. But such a construction problem looks similar to the problem of the construction of a transitive orientation [HM 2]. An important subroutine is finding a maximal independent set (MIS). A first parallel solution of the MIS-problem is due to Karp, Upfal and Wigderson [KUW]. An NC^2-solution of that problem is due to Luby [Lu]. A solution which needs only a linear number of processors in time $O(\log^4 n)$ on a CREW-PRAM is due to Goldberg and Spencer [GS].

Our algorithm to find a strongly perfect elimination scheme is arranged as follows:

Input: a strongly chordal graph G

1) Compute the set of all cliques of G.
2) Compute a strongly perfect elimination scheme on the set of cliques
3) Translate the strongly perfect elimination scheme on the cliques to a strongly perfect elimination ordering on the vertices. This will be done by the algorithm SPE.

The first step is already done. It remains to consider the steps 2) and 3). For the step 2) we have to introduce some notions:

A *chain* is a sequence $(x_1, \cdots, x_n)$ s.t. $x_i \cap x_{i+1}$ and $x_i \cap x_{i-1}$ are incomparable by inclusion. We denote by $x — y — z$ the fact that there is a chain from x to z via y.

Lemma 12. Let $<$ be a strongly perfect elimination scheme on the hyperedges of a β-acyclic hypergraph H and x, y, z be hyperedges of H. If $y < z$ and $x — y — z$, then $x < y$.

The proof can be done by a trivial induction.

5.1. Algorithm SPE (for the computation of a strongly perfect elimination ordering on the cliques):

Input a β-acyclic hypergraph (V, H) and an ordering $\lesssim$ on H.

Step 1) Compute for each $x, y, z \in H$ the relation $x - y - z$, which means y is on a chain between x and z.

Step 2) Using the algorithm of Luby [Lu] or Goldberg and Spencer [GS], compute a relation $\xrightarrow{A}$, s.t. for each $x, y, z \in H$:

1. If $x - y - z$, then $\neg(y \xrightarrow{A} x) \vee \neg(y \xrightarrow{A} z)$ and $(x \xrightarrow{A} z) \vee \neg(y \xrightarrow{A} z)$ and $(z \xrightarrow{A} x) \vee \neg(y \xrightarrow{A} x)$
2. $(x \xrightarrow{A} y) \vee (y \xrightarrow{A} x)$ for each intersecting pair x, y
3. $\neg(x \xrightarrow{A} y) \vee \neg(y \xrightarrow{A} x)$.

Here one has Krom formulas with variables of the form $x \xrightarrow{A} y$. To find such a relation is to find a suitable replacement of the variables making these Krom formulas true.

Step 3) Check for each x, y, s.t. $x \xrightarrow{A} y$, if $x \underset{\sim}{\xrightarrow{A}} y : x \underset{\sim}{\xrightarrow{A}} y$ iff there is a v, s.t. $x \xrightarrow{A} v \xrightarrow{A} y$, and $x \cap v, v \cap y$ incomparable or $x \xrightarrow{A} y \xrightarrow{A} v$ and $x \cap y, y \cap v$ incomparable.

Step 4) Compute the transitive closure $\xrightarrow{A}{}^{*}$ of $\underset{\sim}{\xrightarrow{A}}$.

Step 5)

If i) $x \xrightarrow{A} y$, $y \xrightarrow{A} z$

ii) not ($z \xrightarrow{A}{}^{*} x$ or $z \xrightarrow{A} y$ or $x \xrightarrow{A}{}^{*} z$ or $y \xrightarrow{A}{}^{*} z$

and

iii) $x \cap z$, $y \cap z$ are incomparable, then set $x \xrightarrow{\wedge} z$, $y \xrightarrow{\wedge} z$

Step 6) Compute the transitive closure $<'$ of $\xrightarrow{\wedge} \cup \xrightarrow{A}{}^{*}$

Step 7) $<$ is any total extension of $<'$. Output $<$.

5.2. Time and Processor Analysis for SPE:

Each step can be done in NC^2. Therefore the algorithm can be interpreted as an NC^2-algorithm. But the processor analysis is more important: Let n be the number of vertices and m be the number of hyperedges. We assume that the hypergraph is implemented as a binary relation. $\{(v, h) : v \in h\}$ We also assume that for no hyperedges h_1, h_2 we have $h_1 \subseteq h_2$. We will now do the processor analysis for each step:

Step 1): To compute $\{x - y - z|, x, y, z \in H\}$ we have to compute

a) $V_1 = \{[x, y]|x \cap y \neq 0\} := \{[x, y]|\exists v : v \in x \wedge v \in y\}$. Set $V' := \{[x, y, v]|v \in x \cap y\}$ and $V_1 := \{[x, y] : \exists v[x, y, v] \in V'\}$.
V_1 and V' can be computed in $\log n$ time with $O(n \cdot m^2) \leq O(n^3)$ processors on a CREW-PRAM.

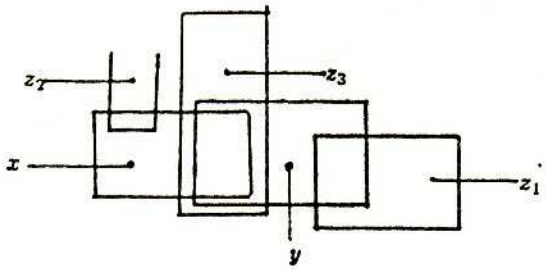
b) $V_2 := \{[x, y, z,]|x \cap y \subset z\}$. Let V' be defined as above.
Let $V'' := \{[x, y, v, z]|[x, y, v] \in V'$ and $v \notin z\}$
and $V''' := \{[x, y, v, z,]|\exists v\,[x, y, v, z,] \in V''\}$

$V_2 := \{[x,y,z,]|[x,y,z] \notin V'''\}$.
V_2 can be computed in $\log n$ time with $\leq O(n^4)$ processors.

c) $V_3 := \{[x,y,z] : x \cap y$ and $y \cap z$ are incomparable by inclusion $\}$
$= \{[x,y,z] : [x,y,z] \notin V_2$ and $[y,z,x] \notin V_2$ and $[x,y],[y,z] \in V_1\}$.
This can be computed by $O(n^3)$ processors in $O(\log n)$ time.

d) Let
$V_4 := \{[x,y,z]|x \cap z \subseteq x \cap y \wedge x \cap z \neq 0\}$
$= \{[z,y,z]|x \cap z \subseteq y\}$.
V_4 can be computed by $O(n^3)$ processors in $O(\log n)$ time.

e) Let
$V_5 := \{[x,y,z]\,|x \cap z \subsetneq x \cap y$ or $z = y\}$
$=$ $\{[x,y,z]|x \cap y, x \cap z$ are not incomparable $([y,x,z,] \notin V_3)$ and $(x \cap z \not\subseteq x \cap z\,([x,z,y] \notin V_4)$ or $z = y)\}$



$[x,y,z_1] \in V_5$, but $[x,y,z_2],[x,y,z_3] \notin V_5$

V_5 can be computed by n^3 processors in $O(\log n)$ time. Using V_5, we want to construct all elements that can be reached by a chain from x via y.

f) Let $(\tilde{V}_{x,y}, \tilde{E}_{x,y} := (\{z|[x,y,z] \in V_5\}|\{[z_1,z_2], z_1 \cap z_2 \not\subseteq x\})$
Say $V_6 := \{[x,y,z_1,z_2]||[z_1,z_2] \in \tilde{E}_{x,y}$ and $z_1,z_2 \in \tilde{V}_{x,y}\}$, ($z_1$ intersects z_2 after the deletion of the vertices of x)
$= \{[x,y,z_1,z_2]||[x,y,z_1],[x,y,z_2] \in V_5$ and $[z_1,z_2,x] \notin V_2\}$.

g) Let $H_{x,y} := \{z|z$ is in the connected component of $\tilde{E}_{x,y}$ and $y\}$
Say $V_7 := \{[x,y,z] : z \in H_{x,y}\}$
For each x,y we need $\leq n^3$ processors and $O(\log n^2)$ time to compute $H_{x,y}$. Therefore we need $\leq O(n^5)$ processors to compute V_7.

h) We can make the following statement:

Lemma 13. x—y—z iff there are x', z', s.t. $x' \cap y$ and $z' \cap y$ are incomparable and $x \in H_{y,x'}$ and $z \in H_{y,z'}$.

PROOF: If $(x = x_1, \cdots, x_i, y, x_{i+2}, \cdots, x_k = z)$ is a chain, then $x_i \cap y$ and $x_{i+2} \cap y$ are clearly incomparable. For $j < i$ we have $x_j \cap x_{j+1} \backslash y \neq 0$ because of the incomparability of the intersections $x_j \cap x_{j+1}$ and $x_j \cap y \subseteq x_{j+1} \cap y$ and $x_j \cap y \subseteq x_i \cap y$. Analogously for $j \geq i+2$ $x_j \cap x_{j+1} \backslash y \neq \emptyset$ and $x_{j+1} \cap y \subseteq x_{i+2} \cap y$. Therefore x is in the connected component of x_i in E_{y,x_i} and x_i and z is in the connected component of x_{i+2} in $E_{y,x_{i+2}}$. Vice versa we consider shortest paths $P_x = (x = x_1, \cdots, x_k = x')$ from x to z' in $E_{y,x'}$ and $P_2 = (z = z_1, \cdots, z_l = z')$ from z to z' in $E_{y,z'}$. As in the proof of the acyclicity test lemma, we can check that these shortest paths form chains. By the definition of $E_{y,x'}$ and $E_{y,z'}$, all intersections $x_i \cap x_{i+1}$ and $z_j \cap z_{j+1}$ contain elements not being in y. Therefore they are incomparable with $x' \cap y$ or $z' \cap y$, respectively. We can paste these two chains together and the lemma is proved. □

$V_8 := \{[x,y,z] | x — y — z\}$
$= \{[x,y,z] | \exists x', z' \, V_7[y,x',x] \text{ and } V_7[y,z',z] \text{ and } V_3[x',y,z']\}$.

To compute V_8, we first compute
$\tilde{V}' := \{[x',y,z',z] | x' \cap y,\ y \cap z' \text{ incomparable and } z \in H_{y,z'}\}$ ($0(n^4)$ processors and $O(\log n)$ time)
$\tilde{V}'' := \{[x',y,z] | \exists z' [x',y,z',z] \in \tilde{V}'\}$
$\tilde{V}''' := \{[x,x',y,z] | [x',y,z] \in \tilde{V}'' \text{ and } x \in H_{y,x'}\}$ ($O(n^4)$ processors and $)O(\log n)$ time)
$V_8 := \{[x,y,z] | \exists x' [x,x',y,z] \in V'''$.
Therefore we can compute V_8 by $O(n^4)$ processors in $O(\log n)$ time. Therefore

Theorem 8: $\{[x,y,z] | x — y — z\}$ can be computed by $O(n^4)$ processors in $O(\log n^2)$ time.

Step 2):

a) The computation of the Krom formulas does not need more processors than the size of the output; that means $O(n^4)$ processors.

b) The deductive closure is not computed in the usual way: The deductive closure is $D := \{(x \longrightarrow y) \vee (y \longrightarrow x) | x \cap y \neq 0\} \cup \{\neg(x \longrightarrow y) \vee \neg(y \longrightarrow x) | x, y \in H\} \cup \{\neg(y \stackrel{A}{\longrightarrow} x) \vee (z \longrightarrow w) | x — y — w \text{ and } x — w — z\}$.
Therefore the deductive closure can be computed by n^4 processors.

c) To compute $\stackrel{A}{\longrightarrow}$, we need $O(\log^2 n)$ time and $O((n^2)^2 \cdot (n^2)^2) = O(n^8)$ processors [Lu] or $O(\log^4 n)$ time and $O(n^2 + n^4) = O(n^4)$ processors [GS]. That means Step 2) needs $O(\log^2 n)$ time and $O(n^8)$ processors or $O(\log^4 n)$ time and $O(n^4)$ processors.

Step 3): $\stackrel{A}{\longrightarrow}$ can be computed from $\stackrel{A}{\longrightarrow}$ and V_3 in Step 1) in $O(\log n)$ time and by $O(n^3)$ processors.

Step 4): The transitive closure $\stackrel{A}{\longrightarrow}^*$ of $\stackrel{A}{\sim}$ can be computed in $O(\log^2 n)$ time by $O(n^3)$ processors.

Step 5): $\{(x,y,z) | x \stackrel{\wedge}{\longrightarrow} z \text{ and } y \stackrel{\wedge}{\longrightarrow} z\}$ can be computed from the previous data in constant time by $O(n^3)$ processors. Therefore $\stackrel{\wedge}{\longrightarrow}$ can be computed in $O(\log n)$ time by $O(n^3)$ processors.

Step 6): can be computed in $O(\log^2 n)$ time by $O(n^3)$ processors.

Step 7): The completion of a partial ordering $<$ to a total ordering will be done in two steps:

a) We will construct a set of levels H_i, s.t.

i) $x < y \Longrightarrow \exists i > j$, s.t. $x \in H_i$, $y \in H_j$ and

ii) $H_i \cap H_j = \emptyset$ for $i \neq j$.

Define $H_i := \{x|$ the maximal length of any ascending chain $x < x_1 < x_2 < \cdots < x_i$ is $i\}$

b) Let $x \prec y$ iff $x \in H_i$, $y \in H_j$, $i > j$ or $x, y \in H_i$ and $x \lesssim y$ w.r.t. the ordering $\overset{<}{\sim}$ of the input.

To compute step a), we state an algorithm which needs $\leq O(n^4)$ processors and $O(\log^2 n)$ time.

$A_{2^0} :=<$
For $i = 1$ **until** $\log n$
$\lceil A_{2^{i+1}} = A_{2^i} \circ A_{2^i} = \{(x,y)|\exists z : A_i(x,z) \cap A_i(z,y)\}$
$\lfloor A_{2^i} := A_{2_i} \backslash A_{2^{i+1}}$
(The sets A_{2^i} are pairwise disjoint)
For $j = 1$ **until** $\log n$
For each number $\nu < m$ of arity j and **each** $k \leq \log n - j$
begin $A_{\nu \cdot 2^k + 2^{k-1}} := \{(x,y) \in A_{\nu \cdot 2^k} | x \in A_{\nu \cdot 2^k} \circ A_{2^k}\}$
$A_{\nu \cdot 2^k} := A_{\nu \cdot 2^k} \backslash A_{\nu \cdot 2^k + 2^{k-1}}$
end loop
(The sets $A_{\nu \cdot 2^k}$, s.t. ν of arity $j+1$ are pairwise disjoint.)
For each $x : h(x) = \max\{k|\exists y (y,x) \in A_k\}$. $x \in H_k :\Longleftrightarrow h(x) = k$.
Step b) can be executed in $O(\log n)$ time by n^3 processors. Therefore:

Theorem 9: A strong perfect elimination ordering can be constructed by $O(n^4)$ processors in $O(\log^4 n)$ time or by $O(n^8)$ processors in $O(\log^2 n)$ time.

5.3. Proof of the Correctness of the Algorithm SPE:

Step 5) guarantees that if $y \underset{\sim}{\overset{A}{\rightharpoonup}} z$, but not $y \underset{\sim}{\overset{A}{\rightharpoonup}} z$ (this means $z < y$ may be possible) and $x \wedge z$, $y \wedge z$ are incomparable, then $x < z$ is preserved in form of $x \overset{\wedge}{\rightharpoonup} z$ Therefore the transitive closure of $\underset{\sim}{\overset{A}{\rightharpoonup}} \cup \overset{\wedge}{\rightharpoonup}$ computed in Step 6) suffices the axioms of a strong elimination ordering. We have only to show that $\underset{\sim}{\overset{A}{\rightharpoonup}} \cup \overset{\wedge}{\rightharpoonup}$ has no directed cycles.

Assume $\underset{\sim}{\overset{A}{\rightharpoonup}} \cup \overset{\wedge}{\rightharpoonup}$ has a cycle C with minimal length. Consider three consecutive hyperedges x, y, z of C, $x \longrightarrow y \longrightarrow z$, $\longrightarrow = \overset{\wedge}{\rightharpoonup} \cup \underset{\sim}{\overset{A}{\rightharpoonup}}$.

Claim: $x \cap y \subsetneq y \cap z$.
The claim leads to a contradiction:

PROOF OF THE CLAIM: At first $x \cap y$ and $y \cap z$ are comparable by inclusion. Otherwise $x \underset{\sim}{\overset{A}{\rightharpoonup}} z$. We now assume $y \cap z \subseteq x \wedge y$. That means $x \cap z = y \cap z$.

1) Assume $y \overset{\wedge}{\underset{\sim}{A}} z$. Then this is a w, s.t. $y \cap z$, $w \cap z$ are incomparable and $w \overset{A}{\underset{\sim}{\rightarrow}} z$ but not $w \overset{A}{\underset{\sim}{\rightarrow}} z$. But then we cannot have $z \overset{A}{\underset{\sim}{\rightarrow}} x$, otherwise $w \overset{A}{\underset{\sim}{\rightarrow}} z$. Therefore $x \overset{\wedge}{\underset{\sim}{A}} z$. We have a contradiction to the minimality of the cycle C.

2) Assume $y \cap z \subsetneqq x \cap y$. Because of 1) we may assume $y \overset{A}{\underset{\sim}{\rightarrow}} y$. Let $d = (y, x_1, \cdots, x_k, z, (u))$ be a minimal chain verifying $y \overset{A}{\underset{\sim}{\rightarrow}} z$. Then x_1 is the only element of the chain touching $y \backslash z$.
Assume x_1 also touches $x \backslash z$, then $(x, x_1, \cdots, x_k, z, (u))$ is a chain verifying $x \overset{A}{\underset{\sim}{\rightarrow}} z$ and we have a contradiction to the minimality of the cycle C.
Assume x_1 does not touch $x \backslash z$. Then $y \cap x_1$ and $x \cap y$ are incomparable and $(x, y, x_1, \cdots, x_k, z, (u))$ is a chain and therefore $x \overset{A}{\underset{\sim}{\rightarrow}} z$. Therefore here we have again a contradiction to the minimality of the cycle C.

3) We may now assume $x \cap y = y \cap z = x \cap z$. We cannot have $x \overset{\wedge}{\underset{\sim}{A}} y$. Otherwise by the same argument as in 1) we must have $x \overset{A}{\underset{\sim}{\rightarrow}} y$ and $z \overset{A}{\underset{\sim}{\rightarrow}} y$, which is a contradiction to $y \overset{A}{\underset{\sim}{\rightarrow}} z$. Also we cannot have a chain (x, y, z), s.t. $x \overset{A}{\underset{\sim}{\rightarrow}} y \overset{A}{\underset{\sim}{\rightarrow}} w$. We had also $z \overset{A}{\underset{\sim}{\rightarrow}} y$. Therefore there must be a chain $(x, x_1, \cdots, x_k, y)$, s.t. $x \overset{A}{\underset{\sim}{\rightarrow}} x_1 \overset{A}{\underset{\sim}{\rightarrow}} x_2 \cdots \overset{A}{\underset{\sim}{\rightarrow}} x_k \overset{A}{\underset{\sim}{\rightarrow}} y$. Let $x_0 = x$ and $x_{k+1} := y$.

Clearly $y \cap z = x \cap y \subsetneqq x_i \cap x_{i+1}$ for each $i = 0, \cdots, k$. The subset property follows from the β-acyclicity and properness from the incomparability of the intersections.

Subclaim: $x_i \cap z = y \cap z$

PROOF OF THE SUBCLAIM: We will prove it by induction.
$i := k + 1$; trivial.

Assume $x_i \cap z = y \cap z$. We know $x_{i+1} \cap x_i$ contains elements not being in $y \cap z$. By $x_{i+1} \cap z = y \cap z$, $x_i \cap x_{i+1} \backslash z = (x_i + 1 \backslash z) \cap x_i = (x_{i+1} \backslash (y \cap z)) \cap x_i = x_i \cap x_{i+1} \backslash (y \cap z) \neq 0$.

But if $y \cap z \subsetneqq x_i \cap z$, then $x_i \cap z$ and $x_i \cap x_{i+1}$ are incomparable by inclusion. That would force $z \overset{A}{\underset{\sim}{\rightarrow}} y$ because the x_i form a chain. This is a contradiction. □

But now, using the same arguments as in 2), we get $x_i \overset{A}{\underset{\sim}{\rightarrow}} z$ for each $i = 0, \cdots, k+1$ and therefore $x \overset{A}{\underset{\sim}{\rightarrow}} z$. This is a contradiction to the minimality of the cycle C. □

5.4. Translation to a Strongly Perfect Elimination Ordering on the Vertices

Now we have to consider the translation of a strongly perfect elimination ordering on the cliques into a strongly perfect elimination ordering on the vertices.

Input: A β-acyclic hypergraph (V, H), a strongly perfect elimination ordering $<$ on H and an ordering $\prec$ on V.

1) For each x, h_1, h_2, s.t. $h_1 < h_2$ and $x \in h_1, h_2$, delete x from h_1. (After the execution of this step, each x is in a unique hyperedge.)

2) For each x, y, h_x, h_y, s.t. $x \in h_x, y \in h_y$
$x < y$ iff ($h_x < h_y$ or $h_1 = h_2$ and $x \prec y$).

Step 1) needs $O(\log n)$ time and $O(n^3)$ processors.

Step 2) needs $O(\log n)$ time and $O(n^2)$ processors.

Therefore the whole algorithm can be executed in $O(\log n)$ time by $O(n^3)$ processors.

From this we conclude the following

Theorem 10 (Complexity of Strongly Perfect Elimination):

For any strongly chordal graph we can compute a strongly perfect elimination ordering by a CREW-PRAM in $O(\log^2 n)$ time using $O(n^8)$ processors or in $O(\log^4 n)$ time using $O(n^4)$ processors.

Section 6. Conclusions:

One might be interested to continue the whole program to Ptolemäic graphs (i.e., distance hereditary chordal graphs) or to path graphs. The recognition problem of Ptolemäic graphs is trivially in NC, because they are chordal graphs which have additionally a finite set of graphs as forbidden induced subgraphs [BM]. Only the number of processors seems to be too high. But the decomposition property of this graph class may lead to an alternating logspace algorithm with a polynomial tree size. This should imply the existence of a parallel recognition algorithm of less processors [Ru]. We shall deal with the whole topic in a later paper.

References

[BDS] Brouwer, A., Duchet, P., and Schrijver, A.,
Graphs Whose Neighborhoods Have no Special Cycle
Discrete Math. 47 (1983), pp. 177-182

[BFMY] Beeri, C., Fagin, R., Maier, D., and Yannakakis, M.,
On the Desirability of Acyclic Data Base Schemes
JACM 30 (1983), pp. 479-513

[BK] Brouwer, A., and Kolen, A.,
A Super-Balanced Hypergraph Has a Nest Point
Mathematisch Centrum, Report ZW 146, Amsterdam (1980)

[BM] Bandelt, H.J., and Mulder, H.,
Distance-Hereditary Graphs
Journal of Combinational Theory Ser. B 41 (1986), pp. 182-208

[BP] Bandelt, H.J., and Prisner, E.,
Clique Graphs and Helly Graphs
(preprint)

[Bu] Buneman, A.,
A Characterization of Rigid Circuit Graphs
Discrete Math. 9 (1974), pp. 205-212

[Cl] Cole, R.,
Parallel Merge Sort
27^{th} IEEE FOCS (1986), pp. 511-516

[CV] Cole, R., and Vishkin, U.,
Approximate and exact parallel scheduling with applications to list, tree and graph problems
27^{th} IEEE FOCS (1986), pp. 478-491

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

[Di] Dirac, G.,
On Rigid Circuit Graphs
Abhandlungen Mathematischer Seminare der Universität Hamburg 25 (1961), pp. 71-76

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

[Fa 1] Farber, M.,
Characterizations of Strongly Chordal Graphs
Discrete Math. 43 (1983), pp. 173-189

[Fa 2] Farber, M.,
Applications of LP-Duality to Problems Involving Independence and Domination
Ph.D. Thesis, Computer Science Department, Rutgers University, New Brunswick, NJ (1982)

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

[Ga] Gavril, F.,
The Intersection Graphs of Subtrees of a Tree Are Exactly the Chordal Graphs
J. Comput. Ser. B 16 (1974), pp. 47-56

[GH] Gilmore, P., and Hoffman, A.,
A Characterization of Comparability Graphs and of Interval Graphs
Can. J. Math. 16 (1964), pp. 539-548

[Go] Golombic, M.,
Algorithmic Graph Theory and Perfect Graphs
Academic Press, New York (1980)

[Gl] Goldschlager, L.,
Synchroneous Parallel Computation
Journal of the ACM 29 (1982), pp. 1073-1086

[GS] Goldberg, M., and Spencer, T.,
A New Parallel Algorithm for the Maximal Independent Set Problem
to appear in FOCS '87

[HCS] Hirschberg, D.S., Chandra, A.K., and Sarvate, D.V.,
Computing Connected Components on Parallel Computers
CACM 22 (1979), pp. 461-464

[HM 1] Hembold, D. and Mayr, E.,
Two Processor Scheduling is in NC, in: *VLSI Algorithms and Architectures (ed. Makedon et. al.)*
LNCS 227, pp. 12-15

[HM 2] Hembold, D., and Mayr, E.,
Applications of Parallel Scheduling to Perfect Graphs
Graph-Theoretic Concepts in Computer Science, LNCS 246 (1987), pp. 188-203

[Jo] Johnson, D.S.,
NP-Completeness Column
Journal of Algorithms 6 (1985), pp. 434-451

[KUW] Karp, R., Upfal, E., and Wigderson, A.,
Finding a Maximum Matching in NC
17^{th} ACM STOC (1985), pp. 22-32

[KVV] Kozen, D. Vazirani, U., and Vazirani, V.,
NC-Algorithms for Comparability Graphs, Interval Graphs and Testing Unique Perfect Matching
to appear

[LB] Lekkerkerker, C., and Boland, D.,
Representation of Finite Graphs by a Set of Intervals on the Real Line
Fund. Math. 51 (1962), pp. 45-64

[Lu] Luby, M.,
A Simple Parallel Algorithm for the Maximal Independent Set Problem
17^{th} ACM STOC (1984), pp. 1-10

[MVV] Mulmulay, K., Vazirani, U., and Vazirani, V.,
Matching is as Easy as Matrix Inversion
Proc. 19^{th} ACM STOC (1987), pp. 345-354

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

[Ru] Ruzzo, W.,
Tree Size Bounded Alternation
JCSS 21 (1980), pp. 218-235

[RV] Rabin, M.O., and Vazirani, V.,
Maximum Matching in General Graphs through Randomization
Report No. 15-84, Center for Research in Computing Technology, Harvard University, Cambridge, Mass (1984)

[SV] Shiloach, Y., and Vishkin, K.,
An $O(\log n)$ Parallel Connectivity Algorithm
J. Algorithms 3 (1982), pp. 57-67

[TY 1] Tarjan, R., 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. 13 (1984), pp. 566-579

[TY 2] Tarjan, R., and 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