A Coarse-Grained Parallel Algorithm for Maximal Cliques in Circle Graphs

E. N. Cáceres[1], S. W. Song[2], and J. L. Szwarcfiter[3]

[1] Universidade Federal do Mato Grosso do Sul
Departamento de Computação e Estatística
Campo Grande, MS, 79069, Brazil
edson at dct. ufms. br
[2] Universidade de São Paulo
Departamento de Ciência da Computação - IME
São Paulo, SP - 05508-900 - Brazil
song at ime. usp. br
[3] Universidade Federal do Rio de Janeiro
Instituto de Matemática and Núcleo de Computação Eletrônica
Rio de Janeiro, RJ, 21945-970, Brazil
jayme at nce. ufrj. br

Abstract. We present a parallel algorithm for generating the maximal cliques of a circle graph with n vertices and m edges. We consider the Coarse-Grained Multicomputer Model (CGM) and show that the proposed algorithm requires $O(\log p)$ communication rounds, where p is the number of processors, independent of n. The main contribution is the use of a new technique based on the unrestricted depth search for the design of CGM algorithms.

1 Introduction

In this paper we present a new CGM/BSP parallel algorithm for finding maximal cliques in circle graphs.

Circle graphs are a special kind of intersection graphs. Recognition of circle graphs has been an open problem for many years until the mid-eighties with solutions discovered independently by several researchers [1, 8, 10]. These algorithms basically makes use of a graph decomposition technique [5, 6]. The presented algorithms are sequential polynomial algorithms. We consider the problem of generating all maximal cliques in a circle graph of n vertices and m edges. Szwarcfiter and Barroso have studied this problem and presented a sequential $O(n(m + \alpha))$ algorithm [11] where α is the number of maximal cliques of the graph. Cáceres and Szwarcfiter [3] present a PRAM algorithm of $O(\alpha \log^2 n)$ time using n^3 processors in a CREW PRAM.

To our knowledge there are no known parallel algorithms for finding maximal cliques in circle graphs under the coarse-grained parallel computing model. Based on [11] and [3] we present a CGM (Coarse-Grained Multicomputer) algorithm for this problem that requires $O(\log p)$ communication rounds. The main contribution of this paper is the introduction of the *unrestricted search* as a new technique for the design of CGM algorithms.

2 Coarse-Grained Multicomputer (CGM) Model

The PRAM model has been extensively utilized to produce many important theoretical results. Such results from PRAM algorithms, unfortunately, do not necessarily match the speedups observed on *real* parallel machines.

In this paper, we present a parallel algorithm that is based on a more practical parallel model. More precisely, we will use a version of the BSP model [13] referred to as the *Coarse Grained Multicomputer* (CGM) model [7]. In comparison to the BSP model, the CGM allows only bulk messages in order to minimize message overhead. A CGM consists of a set of p processors $P_1, \ldots, P_p$ with $O(N/p)$ local memory per processor and an arbitrary communication network (or shared memory). A CGM algorithm consists of alternating local computation and global communication rounds. Each communication round consists of routing a single h-relation with $h = O(N/p)$, i.e. each processor sends $O(N/p)$ data and receives $O(N/p)$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one long message, thereby minimizing the message overhead. A CGM computation/communication round corresponds to a BSP superstep with communication cost $g\frac{N}{p}$. Finding an optimal algorithm in the coarse grained multicomputer model is equivalent to minimizing the number of communication rounds as well as the total local computation time. The CGM model has the advantage of producing results which correspond much better to the actual performance on commercially available parallel machines.

3 Notation and Terminology

Circle graphs are intersection graphs of a family of chords of a circle. Consider a family of n chords, numbered as $1 - 1', 2 - 2', \ldots, n - n'$, in a circle C (or equivalently in a square) (see Fig. 1 on the left). Assume that any two chords do not share a same endpoint. This corresponds to a circle graph $G = (V, E)$ where each chord corresponds to a vertex: $V = \{1, 2, \ldots, n\}$ and the edge set E is formed by edges (u, v) if chord u intersects chord v. The corresponding circle graph is shown on the right of Fig. 1.

A *circular sequence* S of G is the sequence of the $2n$ distinct endpoints of the chords in circle C, by traversing C in a chosen direction, starting at a given point in C. Denote by $S_1(v)$ and $S_2(v)$ respectively the first and second instances in S of the chord corresponding to $v \in V$ in C. Denote by $S_i(v) < S_j(w)$ (for $i, j = 1, 2$) when $S_i(v)$ precedes $S_j(w)$ in S. We have $S_1(v) < S_2(v)$.

Let $G = (V, E)$ be a circle graph, S the circular sequence of G. An S_1-*orientation* $\boldsymbol{G}$ of G is an orientation in which any directed edge $(v, w) \in E$ satisfies $S_1(v) < S_1(w)$. The S_1-orientation is an acyclic digraph (directed graph).

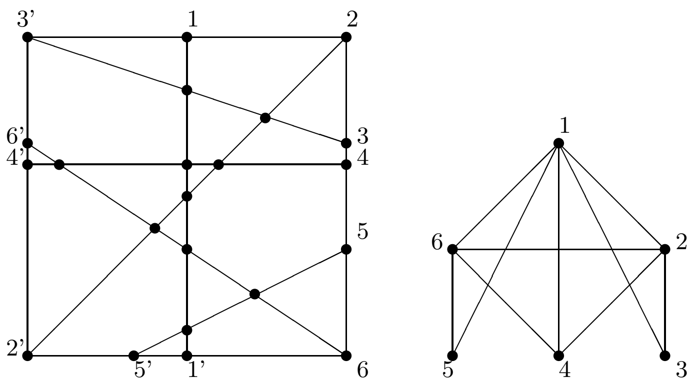

Fig. 1. Family of chords (left) and the corresponding circle graph on the right $G = (V, E)$

Let $\boldsymbol{G}$ denote an acyclic orientation of G. $A_v(\boldsymbol{G})$ and $A_v^{-1}(\boldsymbol{G})$ are the subsets of vertices that leave or enter v, respectively. For $v, w \in V$, v is an *ancestral* of w in $\boldsymbol{G}$ if the directed graph contains a path $v - w$. In this case, w is a *descendant* of v. Denote by $D_v(\boldsymbol{G})$ the set of descendants of v. If $w \in D_v(\boldsymbol{G})$ and $v \neq w$, then v is a *proper ancestral* of w and w a *proper descendant* of v. $\boldsymbol{G}$ is denominated a *transitive digraph* with respect to edges when $(v, w), (w, z) \in E$ implies $(v, z) \in E$. The *transitive reduction* $\boldsymbol{G}_R$ is the subgraph of $\boldsymbol{G}$ formed by the edges that are not motivated by transitivity. In other words, the transitive reduction of a directed graph $\boldsymbol{G}$ is the subgraph $\boldsymbol{G_R}$ with the smallest number of edges such that for every path between vertices in $\boldsymbol{G}$, $\boldsymbol{G_R}$ has a path between those vertices.

Let $G = (V, E)$ be a non-directed graph, $|V| > 1$ and $\boldsymbol{G}$ an acyclic orientation of G.

Let $v, w \in V$. We denote by $Z(v, w) \subset V$ the subset of vertices that are simultaneously descendants of v and ancestrals of w in $\boldsymbol{G}$. An edge $(v, w) \in E$ *induces a local transitivity* when $\boldsymbol{G}(Z(v, w))$ is a transitive digraph. Clearly, in this case the vertices of any path from v to w induce a clique in G. Furthermore, (v, w) *induces a maximal local transitivity* when there does not exist $(v', w') \in E$ different from (v, w) such that v' is simultaneously an ancestral of v and w' a descendant of w in $\boldsymbol{G}$. (v, w) is denominated a *maximal edge*. The orientation $\boldsymbol{G}$ is *locally transitive* when each of its edges induces local transitivity. Fig. 2 shows an example of a locally transitive orientation.

Based on the following theorem, one can use locally transitive orientations for finding maximal cliques.

Theorem 1. *Let $G = (V, E)$ be a graph, $\boldsymbol{G}$ a locally transitive orientation of G and $\boldsymbol{G}_R$ the transitive reduction of $\boldsymbol{G}$. Then there exists a one-to-one correspondence between the maximal cliques of G and paths $v - w$ in $\boldsymbol{G}_R$, for all maximal edges $(v, w) \in E$.*

The proof can be found in [11].

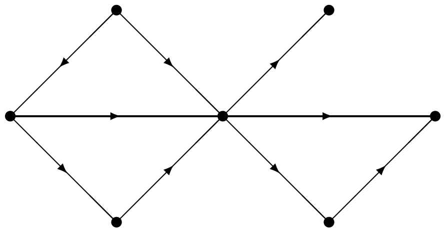

Fig. 2. A locally transitive digraph $G = (V, E)$

4 Algorithm for Unrestricted Search in Acyclic Digraphs

In a depth search of a graph, each edge is visited a constant number of times. Such a search process is known as a restricted search. An unrestricted search in a graph G is a systematic process of traversing G such that each edge is visited a finite number of times. The only restriction is that the process terminate.

The sequential unrestricted depth search of a connected graph $G = (V, E)$ is given as follows. The algorithm starts with a given vertex v chosen as root. It uses a stack Q.

Unrestricted Search (v)
Mark v
Push v onto stack Q
For each vertex w adjacent to v do
  if w is not in Q then
    visit edge (v, w)
    Unrestricted Search (w)
Pop v from Q
Unmark v

Now we will implement this algorithm in CGM. First we give some definitions. Given a digraph $G = (V, E)$, for each vertex $v \in V$ we define the *successor* of v (suc[v]) as a fixed element of the adjacency list of v. Let $e = (u, v) \in E$. We define the *successor* of e as the edge (v,suc[v]). A kl-path is a path C in G with initial edge $e = (k, l)$. An edge $e \in E$ belongs to kl-path C if e is the successor of some edge on the path C, and $e \notin C$. An edge (u, v) can belong to more than one kl-path.

For a undirected graph G, we consider each edge (u, v) as two distinct directed edges (u, v) and (v, u).

Since the successor of each vertex is fixed, all the edges on a kl-path incident with a vertex v have a same successor (v,suc[v]). A kl-path can be a simple path, a cycle, a path together with a cycle or the union of two cycles. In the case when $G = (V, E)$ is an acyclic digraph, the kl-paths are formed by simple paths, with an initial vertex k and a final vertex t.

We present below an algorithm for unrestricted depth search in acyclic digraphs. The algorithm computes all maximal simple paths from a given root r.

The algorithm initially decomposes the digraph $G = (V, E)$ into a set of kl-paths. This decomposition is obtained through the definition of the successors of vertices and edges of G. The algorithm initializes the successor of each vertex $v \in V$ as the first element of its adjacency list of v. After the decomposition, the algorithm explores the edges of the kl-path such that $k = r$, where r is the root of the search and $l = \mathrm{suc}[r]$. The visited edges form a maximal simple path C. Once we have determined a maximal simple path $C = \{v_0, \cdots, v_p\}$, for $r = v_0$ on the kl-path, we can obtain a new maximal simple path, if it exists, as follows. We determine the last vertex $v_i \in C$ that has some vertex in its adjacency list that has not been visited yet. The successors of the vertices $v_j \notin \{v_0, \cdots, v_i\}$ are modified to be the first element of the adjacency list of each v_j, and the successor of v_i is altered to be the element of the adjacency list of v_i immediately following $\mathrm{suc}[v_i]$. The successors of the vertices $\{v_0, \cdots, v_{i-1}\}$ remain unaltered. The new definition of successors determines a new decomposition of the digraph into a set of kl-paths. We determine a simple path $C = \{v_i, \cdots, v_t\}$ on a kl-path in the new decomposition, with $k = v_i$ and $l = \mathrm{suc}[v_i]$. The path $C = \{v_0, \cdots, v_i, \cdots, v_t\}$ formed by the union of the paths $\{v_0, \cdots, v_i\}$ and $\{v_i, \cdots, v_t\}$ is a new maximal simple path. The remaining maximal simple path, if they exist, are computed analogously.

All the vertices w_i that are not reachable from the root vertex r are not included in the search.

Algorithm: Unrestricted Search

1. Define a root r, construct the adjacency lists of G and initialize the maximal simple path CM.
2. Decompose the digraph G into a set of kl-paths.
3. Determine a simple path C from r on the kl-path, with $k = r$ and $l = \mathrm{suc}[r]$.
4. Compute the maximal simple path $CM = CM \cup C$. Verify the existence of any vertex $v_i \in CM$ that has in its adjacency list a vertex that has not been visited yet.
5. In case v_i exists, $CM = \{v_0, \cdots, v_{i-1}\}$. Alter the successors of the vertices $v_i \notin CM$, and unmark the edges $e \in G$ that do not belong to the path CM.
6. Apply steps 2, 3, 4, and 5 to the set of unmarked edges of G for $r = v_i$ until all possible maximal simple paths from v_0 have been explored.

First, we rank the adjacency lists of all $v \in V$ and define the successor of v to be the first element of the adjacency list of v.

We start with the root r and with the defined successors, this represents a maximal path in G. Since our graph is acyclic, using list ranking, we can rank all vertices of this path. After this, we mark the last vertex of the path and change the successor (if there is one) of the tail of this path and compute another maximal path. Otherwise, using a proper data structure, we can backtrack in this path and visit a vertex that was not visited before.

The unrestricted search in this acyclic graph can be done in $O(\log p)$ communication rounds.

5 The Maximal Cliques Algorithm

An S_1-orientation $\boldsymbol{G}$ of a given circle graph G can be easily obtained through its circular sequence. We thus assume that a S_1-orientation is given as input. It can also be observed [11] that if $\boldsymbol{G}$ is locally transitive, then $(v, w) \in E$ is a maximal edge if and only if $A_v(\boldsymbol{G}) \cap A_w(\boldsymbol{G}) = A_v^{-1}(\boldsymbol{G}) \cap A_w^{-1}(\boldsymbol{G}) = \emptyset$.

Step 3 below is the main step of this algorithm and is based on the unrestricted search.

5.1 Algorithm Description

CGM Algorithm: Maximal Cliques

1. Construct the transitive reduction $\boldsymbol{G}_R$.
2. Determine all maximal edges of $\boldsymbol{G}$.
3. For each maximal edge $(v, w) \in E$, determine all paths $v - w$ in $\boldsymbol{G}_R$.

5.2 An Illustrative Example

Consider the circle graph G of Fig. 1. Consider the S_1-orientation shown in Fig. 3 (a).

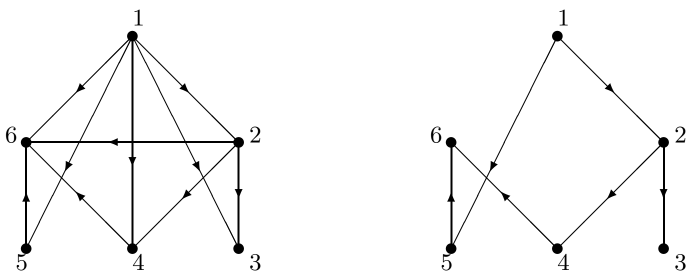

Fig. 3. (a) Graph $\boldsymbol{G}$ (S_1-orientation) and (b) Transitive Reduction Graph $\boldsymbol{G_R}$

Step 1

Determine the transitive reduction $\boldsymbol{G_R}$ of $\boldsymbol{G}$.

We can easily determine the adjacency lists $A_v^{-1}(\boldsymbol{G})$ for all the vertices of $\boldsymbol{G}$. We can also obtain the lists $A_v^{-1}(\boldsymbol{G})$ by partitioning the edge lists into sublists, one for each vertex v. We use the array of lists ADJ to store the adjacency lists. $ADJ[v_i]$ contains the list of vertices v_j that arrives in v_i. Using recursive doubling, we remove from each adjacency list $ADJ[v_i]$ all the vertices that are in adjacency lists $ADJ[v_j]$, for all $v_j \in ADJ[v_i]$. This can be done as follows.

for all $v_i \in V$ in parallel do
    $ADJ[v_i] \leftarrow A_{v_i}^{-1}(\boldsymbol{G})$
    $BADJ[v_i] \leftarrow \cup_{v_j \in ADJ[v_i]} ADJ[v_j]$
repeat $\lceil \log n \rceil$ times
    for all $v_i \in V$ in parallel do
        $BADJ[v_i] \leftarrow BADJ[v_i] \cup \{\cup_{v_j \in BADJ[v_i]} BADJ[v_j]\ \}$
for all $v_i \in V$ in parallel do
    $ADJ[v_i] \leftarrow ADJ[v_i] - BADJ[v_i]$

The adjacency lists $A_{v_i}^{-1}(\boldsymbol{G_R})$ are given by $ADJ[v_i]$.

The main step that is repeated $O(\log n)$ times amounts to using the *pointer jumping* or *recursive doubling* technique. By this technique, it can be easily shown that the CGM algorithm requires $O(\log n)$ communication rounds. We can however do better by using the p^2-ruling set technique presented in [4] for the list ranking problem. An *r-ruling set* is defined as a subset of selected list elements that has the following properties: (1) No two neighboring elements are selected. (2) The distance of any unselected element to the next selected element is at most r. Because of the characteristics of circle graphs and the S_1-orientation, we have at most $O(n^2)$ partial lists in graph G. So this step can be done with $O(\log p)$ communication rounds.

For our example $\boldsymbol{G_R}$ is

$$\boxed{(1,2)(1,5)(2,3)(2,4)(4,6)(5,6)}$$

Step 2

In this step we determine the maximal edges of $\boldsymbol{G}$. We base on the observation [11] that if $\boldsymbol{G}$ is locally transitive, then $(v,w) \in E$ is a maximal edge if and only if $A_v(\boldsymbol{G}) \cap A_w(\boldsymbol{G}) = A_v^{-1}(\boldsymbol{G}) \cap A_w^{-1}(\boldsymbol{G}) = \emptyset$.

We can either assume the adjacency lists as part of the input or easily computer them. In the example, we have

$$\begin{array}{ll}
A_1(\boldsymbol{G}) = \{2,3,4,5,6\} & A_1^{-1}(\boldsymbol{G}) = \emptyset \\
A_2(\boldsymbol{G}) = \{3,4,6\} & A_2^{-1}(\boldsymbol{G}) = \{1\} \\
A_3(\boldsymbol{G}) = \emptyset & A_3^{-1}(\boldsymbol{G}) = \{1,2\} \\
A_4(\boldsymbol{G}) = \{6\} & A_4^{-1}(\boldsymbol{G}) = \{1,2\} \\
A_5(\boldsymbol{G}) = \{6\} & A_5^{-1}(\boldsymbol{G}) = \{1\} \\
A_6(\boldsymbol{G}) = \emptyset & A_6^{-1}(\boldsymbol{G}) = \{1,2,4,5\}
\end{array}$$

We thus compute the following maximal edges.

$$\boxed{(1,3)(1,6)}$$

Step 2 can be computed with a constant number of communication rounds.

Step 3

In this step we first compute for each $v \in V$ the subsets $W(v)$ formed by vertices w such that (v, w) is a maximal edge. From the maximal edges obtained in Step 2 we get

$$\begin{array}{ll} W(1) = \{3,6\} & W(2) = \emptyset \\ W(3) = \emptyset & W(4) = \emptyset \\ W(5) = \emptyset & W(6) = \emptyset \end{array}$$

Construct now the subgraph $\boldsymbol{H}$ of $\boldsymbol{G_R}$ induced by the vertices that are simultaneously descendants of v and ancestrals of any $w \in W(v)$. This can be done by determining the transitive closure [9] $\boldsymbol{G_R}$ and through the intersection of the vertices that leave v with those that enter each of $w \in W(v)$. The paths $v - w$ in $\boldsymbol{G_R}$ taken from a certain vertex v are exactly the source-sink paths in $\boldsymbol{H}$. These paths can be obtained through the parallel unrestricted depth search algorithm (see Section 4).

For our example the subgraph $\boldsymbol{H}$ is shown in Fig. 4.

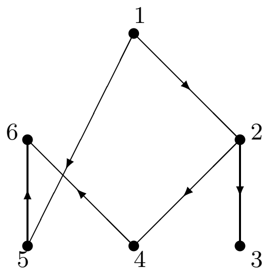

Fig. 4. Graph $\boldsymbol{H}$

$$\boxed{(1,2)(1,5)(2,3)(2,4)(4,6)(5,6)}$$

Let us now perform an unrestricted search on the digraph $\boldsymbol{H}$.

The unrestrited search in this acyclic graph can be done in $O(\log p)$ communication rounds.

Observe that digraph $\boldsymbol{H}$ is acyclic. Using the parallel unrestricted search algorithm of [2] we determine the maximal paths of $\boldsymbol{H}$. For our example the maximal paths in $\boldsymbol{H}$ are:

$$\boxed{C_1 = \{1,2,3\}}$$
$$\boxed{C_2 = \{1,2,4,6\}}$$
$$\boxed{C_3 = \{1,5,6\}}$$

The source-sink paths of $\boldsymbol{H}$ obtained in the previous step form maximal cliques of G.

6 Correctness

Lemma 1. *Let $G = (V, E)$ be a weakly connected digraph. Let $v \in V$ and a kl-path in G such that $k = v$ and $l = suc[v]$. Then the kl-path is: (i) a simple path; (ii) a cicle; or (iii) a path connected to a cycle.*

Proof: [3]

Lemma 2. *Let $G = (V, E)$ be an acyclic weakly connected digraph. Then each kl-path is a simple maximal path.*

Proof: [3]

Theorem 2. *Let $G = (V, E)$ be an acyclic weakly connected digraph. All maximal paths of G, starting at the root vertex r, will be found by the algorithm at least once.*

Proof: [3]

Theorem 3. *Let $G = (V, E)$ be an acyclic weakly connected digraph. None of the maximal paths of G, starting at the root vertex r, will be found by the algorithm more than once.*

Proof: [3]

Lemma 3. *Graph $\boldsymbol{G_R}$ obtained at the end of Step 1 of the parallel algorithm is the transitive reduction of $\boldsymbol{G}$.*

Proof: At each iteration we compute for each vertex v the set L_v formed by the union of the adjacency lists of the vertices that arrive at v. The set L_v contains the vertices for which there exists at least one path of length $\geq 2^i$ to v. At each iteration i, for each vertex v, we remove, from the adjacency lists of the vertices that arrive at v, those vertives that belong to L_v. After $\lceil \log n \rceil$ iterations, all the vertices that are ancestral of v belong to L_v. Thus the adjacency lists will contain only vertices v_j for which there does not exist any path joining u_j to v different of (u_j, v).

Theorem 4. *The parallel algorithm computes all the maximal cliques correctly.*

Proof: By the previous lemma, the algorithm computes the transitive reduction correctly. By using the unrestricted search of an acyclic digraph, we determine all the maximal paths in the digraph $\boldsymbol{H}$. The correctness result follows from the fact that the S_1-orientation $\boldsymbol{G}$ is locally transitive.

Acknowledgments

The first author is supported by PRONEX-SAI. The second author is supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) Proc. No. 98/06138-2, CNPq Proc. No. 52.3778/96-1 and 46.1230/00-3, and CNPq/NSF Collaborative Research Program Proc. No. 68.0037/99-3. The third author is partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, and Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro, FAPERJ, Brazil.

References

1. A. Bouchet.: Reducing Prime Graphs and Recognizing Circle Graphs. Combinatorica **7** (1987) 243–254
2. E.N. Cáceres.: Parallel Algorithms for Graph Problems. (In Portuguese.) PhD thesis, COPPE - UFRJ, Rio de Janeiro - RJ, (1992)
3. E. N. Cáceres and J. L. Szwarcfiter.: In preparation.
4. E. Cáceres, F. Dehne, A. Ferreira, P. Flocchini, I. Rieping, A. Roncato, N. Santoro, and S.W. Song.: Efficient Parallel Graph Algorithms For Coarse Grained Multicomputers and BSP. In: Proceedings ICALP '97 - 24th International Colloquium on Automata, Languages, and Programming. P. Degano, R. Gorrieri, A. Marchetti-Spaccamela (editors). Lecture Notes in Computer Science **1256** (1997) 390–400
5. W. H. Cunningham.: Decomposition of Directed Graphs. SIAM J. Alg. and Disc. Methods **3** (1982) 214–228
6. W. H. Cunningham and J. Edmonds.: A Combinatorial Decomposition Theory. Canad. J. Math. **32** (1980) 734–765
7. F. Dehne (Ed.), Coarse grained parallel algorithms. In: Special Issue of Algorithmica **24** (1999) 173–426
8. C. P. Gabor, W.L. Hsu, and K. J. Supowit.: Recognizing Circle Graphs in Polynomial Time. J. of Assoc. Comput. Mach. **36** (1989) 435–474
9. R. M. Karp and V. Ramachandran.: Parallel Algorithms for Shared-Memory Machines. In: J. van Leeuwen (ed.): Handbook of Theoretical Computer Science Vol. A. Chapter 17, The MIT Press/Elsevier (1990) 869–941
10. W. Naji.: Graphes des Cordes, Caractérisation et Reconnaissance. Disc. Math. **54** (1985) 329–337
11. J. L. Szwarcfiter and M. Barroso.: Enumerating the Maximal Cliques of Circle Graph. In: F.R.K. Chung, R.L. Graham, and D.F. Hsu (eds.): Graph Theory, Combinatorics, Algorithms and Applications, SIAM Publications (1991) 511–517
12. L. G. Valiant.: The Complexity of Enumeration and Reliability Problems. SIAM J. Comp. **8** (1979) 410–421
13. L. G. Valiant.: A Bridging Model for Parallel Computation. Communication of the ACM **33** (1990) 103–111