EFFICIENT PARALLEL RECOGNITION OF COGRAPHS

Stavros D. Nikolopoulos and Leonidas Palios

11– 2003

Preprint, no 11 – 10 / 2003

Department of Computer Science
University of Ioannina
45110 Ioannina, Greece

Efficient Parallel Recognition of Cographs

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina
P.O.Box 1186, GR-45110 Ioannina, Greece
{stavros, palios}@cs.uoi.gr

Abstract: In this paper, we establish structural properties for the class of complement reducible graphs or cographs, which enable us to describe efficient parallel algorithms for recognizing cographs and for constructing the cotree of a cograph. For a graph on n vertices and m edges, both our cograph recognition and cotree construction algorithms run in $O(\log^2 n)$ time and require $O((n+m)/\log n)$ processors on the EREW PRAM model of computation. Our algorithms are motivated by the work of Dahlhaus [9] and take advantage of the optimal $O(\log n)$-time computation of the co-connected components of a general graph [6] and of an optimal $O(\log n)$-time parallel algorithm for computing the connected components of a cograph, which we present. Our results improve upon the previously known linear-processor parallel algorithms for the problems [9, 11]; we achieve a better time-processor product using a weaker model of computation.

Keywords: Perfect graph, cograph, cotree, connected and co-connected component, parallel algorithm, parallel recognition.

1 Introduction

The *complement reducible graphs*, also known as *cographs*, are defined as the class of graphs formed from a single vertex under the closure of the operations of union and complement. More precisely, the class of cographs is defined recursively as follows: (i) a single-vertex graph is a cograph; (ii) the disjoint union of cographs is a cograph; (iii) the complement of a cograph is a cograph.

The cographs have arisen in many disparate areas of applied mathematics and computer science and have been independently rediscovered by various researchers under various names such as D^*-graphs [15], P_4 restricted graphs [7, 8], 2-parity graphs and Hereditary Dacey graphs or HD-graphs [23]. The cographs are perfect and in fact form a proper subclass of permutation graphs and distance hereditary graphs; they contain the class of quasi-threshold graphs and, thus, the class of threshold graphs [5, 10]. Furthermore, cographs are precisely the graphs which contain no induced subgraph isomorphic to a P_4 (chordless path on four vertices), and are also the underlying undirected graphs of the serial-parallel digraphs [24].

The cographs were introduced in the early 1970s by Lerchs [17] who studied their structural and algorithmic properties. Along with other properties, Lerchs has shown that the class of cographs coincides with the class of P_4 restricted graphs, and that the cographs admit a unique tree representation, up to isomorphism, called a *cotree*. The cotree of a cograph G is a rooted tree such that:

(i) each internal node, except possibly for the root, has at least two children;

(ii) the internal nodes are labeled by either 0 (*0-nodes*) or 1 (*1-nodes*); the children of a 1-node (0-node resp.) are 0-nodes (1-nodes resp.), i.e., 1-nodes and 0-nodes alternate along every path from the root to any node of the cotree;

(iii) the leaves of the cotree are in a 1-to-1 correspondence with the vertices of G, and two vertices v_i, v_j are adjacent in G if and only if the least common ancestor of the leaves corresponding to v_i and v_j is a 1-node.

Lerchs' definition required that the root of a cotree be a 1-node; if however we relax this condition and allow the root to be a 0-node as well, then we obtain cotrees whose internal nodes all have at least two children, and whose root is a 1-node if and only if the corresponding cograph is connected.

There are several recognition algorithms for the class of cographs. Sequentially, a linear-time algorithm for recognizing cographs was given in [8]. In a parallel setting, cographs can be efficiently (but not optimally) recognized in polylogarithmic time using a polynomial number of processors. Adhar and Peng [1] described a parallel algorithm for this problem which, on a graph on n vertices and m edges, runs in $O(\log^2 n)$ time and uses $O(nm)$ processors on the CRCW PRAM model of computation. Another recognition algorithm was developed by Kirkpatrick and Przytycka [16], which requires $O(\log^2 n)$ time with $O(n^3/\log^2 n)$ processors on the CREW PRAM model. Lin and Olariu [18] proposed an algorithm for the recognition and cotree construction problem which requires $O(\log n)$ time and $O((n^2+nm)/\log n))$ processors on the EREW PRAM model. Recently, Dahlhaus [9] proposed a nearly optimal parallel algorithm for the same problem which runs in $O(\log^2 n)$ time with $O(n+m)$ processors on the CREW PRAM model. Another cograph recognition and cotree construction algorithm was presented by He [11]; it requires $O(\log^2 n)$ time and $O(n+m)$ processors on a CRCW PRAM model.

Since the cographs are perfect, many interesting optimization problems in graph theory, which are NP-complete in general graphs, have polynomial sequential solutions and admit efficient or even optimal parallel algorithms in the case of cographs. Such problems, with a large spectrum of practical applications, include the maximum clique, minimum coloring, minimum domination, Hamiltonian path (cycle), minimum path cover, and isomorphism testing [5, 10]. In particular, for the problem of determining the minimum path cover for a cograph, Lin *et al.* [20] presented an optimal sequential algorithm, which can be used to produce a Hamiltonian cycle or path, if such a structure exists. Bodlaender and Möhring [4] proved that the pathwidth of a cograph equals its treewidth and proposed a linear-time algorithm to determine the pathwidth of a cograph. In a parallel environment, many of the above problems are solved in polylogarithmic time with a linear number of processors for cographs, assuming that the cotree of the cograph is given as input [1, 2, 16]; for example, the minimum path cover problem is solved in $O(\log n)$ time with $O(n/\log n)$ processors [21].

The cotree of a cograph is constructed in $O(\log^2 n)$ time with $O(n+m)$ processors [9, 11], or in $O(\log n)$ time with $O((n^2+nm)/\log n))$ processors [18], and, thus, the cotree construction dominates the time and/or processor complexity of the parallel algorithms for solving all the previously stated optimization problems on cographs. It follows that these parallel algorithms need, in total, either $O(\log^2 n)$ time or $O((n^2+nm)/\log n))$ processors, since they require the cotree as input instead of the standard adjacency-list representation of the input cograph.

In this paper, we establish structural properties of cographs (based on the fact that a cograph contains no induced subgraph isomorphic to a P_4 [17]), which enable us to obtain efficient parallel algorithms for recognizing whether a given graph is a cograph and for constructing the cotree of a cograph. More precisely, for a graph on n vertices and m edges, our algorithms run in $O(\log^2 n)$ time

using $O((n+m)/\log n)$ processors on the EREW PRAM model of computation, an improvement on both the time-processor product and the model of computation over the previously known parallel algorithms for these problems. The algorithms work in a fashion similar to that used in [9] and take advantage of the optimal parallel algorithm for computing the connected components of the complement of a graph described in [6] and an optimal $O(\log n)$-time and $O((n+m)/\log n)$-processor EREW-algorithm which computes the connected components of a graph or detects that it contains a P_4; the latter algorithm is interesting in its own right as it constitutes an optimal parallel connectivity algorithm for cographs, and can be extended to yield an optimal parallel connectivity algorithm for graphs with constant diameter (note that no optimal parallel connectivity algorithm is currently available for general graphs).

The paper is organized as follows. In Section 2, we present the notation and related terminology and we establish results which are the basis of our algorithms. In Section 3, we present the optimal parallel algorithm that either computes the connected components of the input graph or detects that the graph contains a P_4 as an induced subgraph. The cograph recognition and the cotree construction algorithms are described and analyzed in Sections 4 and 5 respectively. Finally, Section 6 concludes the paper with a summary of our results and some open problems.

2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. Let S be a subset of the vertex set $V(G)$ of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$.

A *path* in the graph G is a sequence of vertices $v_1v_2\ldots v_k$ such that $v_iv_{i+1} \in E(G)$ for $i = 1, 2, \ldots, k-1$; we say that this is a path from v_1 to v_k and that its *length* is k. A path is called *simple* if none of its vertices occurs more than once; it is called *trivial* if its length is equal to 0. A simple path $v_1v_2\ldots v_k$ is *chordless* if $v_iv_j \notin E(G)$ for any two non-consecutive vertices v_i, v_j in the path. Throughout the paper, the chordless path on k vertices is denoted by P_k; in particular, a chordless path on 4 vertices is denoted by P_4. If the graph G contains a path from a vertex x to a vertex y, we say that x *is connected to* y. The graph G is *connected* if x is connected to y for every pair of vertices $x, y \in V(G)$. The *connected components* (or *components*) of G are the equivalence classes of the "is connected to" relation on the vertex set $V(G)$ of G. The *co-connected components* (or *co-components*) of G are the connected components of the complement $\overline{G}$ of G.

The *neighborhood* $N(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N[x] := N(x) \cup \{x\}$. The neighborhood of a subset S of vertices is defined as $N(S) := \left(\bigcup_{x\in S} N(x)\right) - S$ and its closed neighborhood as $N[S] := N(S) \cup S$. If two vertices x and y are adjacent in G, we say that x *sees* y; otherwise we say that x *misses* y. We extend this notion to vertex sets: $V_i \subseteq V(G)$ sees (misses) $V_j \subseteq V(G)$ if and only if every vertex $x \in V_i$ sees (misses) every vertex $y \in V_j$.

The parallel cograph recognition algorithm relies on the result stated in the following lemma.

Lemma 2.1. *Let G be a graph, v a vertex of G, $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ the co-connected components of $G[N(v)]$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ the connected components of $G[V(G) - N[v]]$. Then, G contains no P_4 with vertices in both $N[v]$ and $V(G) - N[v]$ if and only if*

(i) every co-component $\widehat{\mathcal{C}}_i$ either sees or misses every component $\mathcal{C}_j$, and

(ii) for every pair of components $\mathcal{C}_i$ and $\mathcal{C}_j$, either $N(\mathcal{C}_i) \subseteq N(\mathcal{C}_j)$ or $N(\mathcal{C}_i) \supseteq N(\mathcal{C}_j)$.

Proof: ($\Longrightarrow$) We assume that the graph G does not contain a P_4 as described; we will show that conditions (i) and (ii) hold. If condition (i) did not hold, then there would be a vertex x of some $\widehat{\mathcal{C}}_i$ which would be adjacent to a vertex y in some $\mathcal{C}_j$ but non-adjacent to a vertex z of $\mathcal{C}_j$; then, the path $vxyz$ would be a P_4 with vertices in both $N[v]$ and $V(G) - N[v]$, a contradiction. Therefore, condition (i) must hold. Suppose now that condition (ii) does not hold; then, there would exist two components $\mathcal{C}_i$ and $\mathcal{C}_j$ and two vertices $a, b \in N(v)$ such that $a \in N(\mathcal{C}_i) - N(\mathcal{C}_j)$ and $b \in N(\mathcal{C}_j) - N(\mathcal{C}_i)$. The vertices a, b are adjacent in G; otherwise, they would belong to the same co-component of $N(v)$, and then, in accordance with condition (i), they would have to either both see or both miss $\mathcal{C}_i$. But then, for any vertex $x \in N(a) \cap \mathcal{C}_i$ and any vertex $y \in N(b) \cap \mathcal{C}_j$, the path $xaby$ is a P_4 and contains vertices from both $N[v]$ and $V(G) - N[v]$; a contradiction again.

($\Longleftarrow$) We assume that the conditions (i) and (ii) hold; we will show that the graph G does not contain a P_4 with vertices in both $N[v]$ and $V(G) - N[v]$. Suppose for contradiction that G contained such a P_4. We distinguish the following cases:

a) *v participates in the P_4:* Since v is adjacent to all the vertices in $N(v)$, such a P_4 can either be of the form $vabc$ with $a \in N(v)$ and $b, c \in V(G) - N[v]$, or of the form $xvyz$ with $x, y \in N(v)$ and $z \in V(G) - N[v]$. In the former case, b, c belong to the same connected component of $G[V(G) - N[v]]$ and a sees exactly one of them, while, in the latter, x, y belong to the same co-component of $G[N(v)]$ and z sees exactly one of them; in either case, condition (i) does not hold, which leads to a contradiction.

b) *v does not participate in the P_4:* Then, the P_4 contains vertices from $V(G) - \{v\}$ and at least one edge, say, xy, with $x \in N(v)$ and $y \in V(G) - N[v]$. The edge xy cannot extend to a P_3 xyz of the P_4: if it did, then $z \in N(v)$, for otherwise y, z would belong to the same connected component of $G[V(G) - N[v]]$ and x would see exactly one of them, in contradiction to condition (i); since $x, z \in N(v)$, the P_4 would be (without loss of generality) $xyzw$ which violates condition (i) no matter whether $w \in N(v)$ (then, x, w belong to the same co-component and $y \in N(x) - N(w)$) or $w \in V(G) - N[v]$ (then, x, z belong to the same co-component and $w \in N(z) - N(x)$). Hence, if a vertex of the P_4 which belongs to $V(G) - N[v]$ is adjacent in the P_4 to a vertex in $N(v)$, it cannot be a midpoint of the P_4. This implies that no vertex in $V(G) - N[v]$ is a midpoint of the P_4; thus, the only possible cases are:

   - the P_4 is $abxy$ where $a, b \in N(v)$: then, condition (i) is violated; a, x belong to the same co-component of $G[N(v)]$ and y sees exactly one of them.
   - the P_4 is $zuxy$ where $u \in N(v)$ and $z \in V(G) - N[v]$: then, if y, z belong to the same component of $G[V(G) - N[v]]$, condition (i) is violated, otherwise, condition (ii) is violated.

   In all cases, we reached a contradiction; therefore, the graph G cannot contain a P_4 with vertices in both $N[v]$ and $V(G) - N[v]$. ∎

It follows that if conditions (i) and (ii) of Lemma 2.1 hold and the graph G contains a P_4 as an induced subgraph, then this P_4 entirely belongs either to one of the co-components $\widehat{\mathcal{C}}_i$ $(1 \leq i \leq \ell)$ of the subgraph $G[N(v)]$ or to one of the components $\mathcal{C}_j$ $(1 \leq j \leq k)$ of the subgraph $G[V(G) - N[v]]$; clearly, no P_4 in $G[N(v)]$ has vertices belonging to two or more co-components of $G[N(v)]$, and no P_4 in $G[V(G) - N[v]]$ has vertices belonging to two or more components of $G[V(G) - N[v]]$.

Additionally, for any vertex v of a graph G, the following observation holds for the number of co-connected components of the subgraph $G[N(v)]$:

Observation 2.1. *Let G be a graph on n vertices and m edges, v a vertex of G, and $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ the co-connected components of $G[N(v)]$. Then, $\ell < \sqrt{2m}$.*

Proof: The definition of $\widehat{\mathcal{C}}_i$s ($1 \leq i \leq \ell$) implies that every vertex of $\widehat{\mathcal{C}}_i$ sees every vertex of $\widehat{\mathcal{C}}_j$, for every $j \neq i$. Thus, there exist at least

$$1/2 \sum_i |\widehat{\mathcal{C}}_i| \cdot \sum_{j \neq i} |\widehat{\mathcal{C}}_j| \;\geq\; 1/2 \sum_i |\widehat{\mathcal{C}}_i| \cdot (\ell - 1) \;\geq\; \ell(\ell-1)/2$$

edges connecting vertices in different co-components of $G[N(v)]$. Since G contains a total of m edges and there are at least ℓ edges connecting v to its neighbors, we conclude that $m \geq \ell + \ell(\ell-1)/2 > \ell^2/2$, from which the observation follows. ∎

Let G be a graph on n vertices. Then, we define the sets L, M, and H of the low-, middle-, and high-degree vertices of G, respectively, as follows:

$$\begin{aligned} L &= \{\, x \in V(G) \mid \text{degree of } x \text{ in } G \;< \frac{1}{4}n \,\} \\ M &= \{\, x \in V(G) \mid \frac{1}{4}n \leq \text{degree of } x \text{ in } G \;\leq \frac{3}{4}n \,\} \\ H &= \{\, x \in V(G) \mid \text{degree of } x \text{ in } G \;> \frac{3}{4}n \,\} \end{aligned}$$

Clearly, the sets L, M, and H partition the vertex set $V(G)$ of G. Then, we can show the following results:

Observation 2.2. *Let G be a graph on n vertices and let $v \in V(G)$. If $v \in M$, then the cardinality of each co-component $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, of the subgraph $G[N(v)]$ and of each connected component $\mathcal{C}_j$, $1 \leq j \leq k$, of the subgraph $G[V(G) - N[v]]$ does not exceed $\frac{3}{4}n$.*

Proof: The definition of the set M implies that $\frac{1}{4}n \leq |N(v)| \leq \frac{3}{4}n$, from which the observation follows. ∎

Lemma 2.2. *Let G be a cograph on n vertices such that the degree of every vertex of G exceeds $\frac{3}{4}n$. If $\widehat{\mathcal{A}}_1, \widehat{\mathcal{A}}_2, \ldots, \widehat{\mathcal{A}}_p$ are the co-components of G, then $|\widehat{\mathcal{A}}_i| \leq \frac{3}{4}n$, for all $i = 1, 2, \ldots, p$.*

Proof: Suppose for contradiction that there exists some i such that $|\widehat{\mathcal{A}}_i| > \frac{3}{4}n$. Then, $\sum_{j \neq i} |\widehat{\mathcal{A}}_j| = n - |\widehat{\mathcal{A}}_i| < \frac{1}{4}n$. This implies that every vertex in $\widehat{\mathcal{A}}_i$ is adjacent to more than $n/2$ vertices in $\widehat{\mathcal{A}}_i$; recall that the degree of each vertex exceeds $3n/4$. Since $\widehat{\mathcal{A}}_i$ is a co-component of a cograph, it consists of at least two connected components, one of which has cardinality at most $n/2$; but then, the vertices of this component cannot be adjacent to more than $n/2$ vertices of $\widehat{\mathcal{A}}_i$. ∎

It is worth noting that any graph G meeting the conditions of Lemma 2.2 (i.e., each of G's vertices has degree exceeding $\frac{3}{4}|V(G)|$) is connected: if G were disconnected, then at least one of its connected components would be of size at most $|V(G)|/2$, which implies that the degrees of the vertices in this component would be less than $|V(G)|/2$, a contradiction.

Dahlhaus [9] has shown the following lemma:

Lemma 2.3. ([9], Lemma 6) *Let G be a cograph on n vertices and let*

$$L_a = \{\, v \in V(G) \mid \text{degree of } v \text{ in } G \;< \frac{1}{a}n \,\}.$$

Then, the cardinality of each connected component of the graph $G[L_a]$ does not exceed $\frac{2}{a}n$.

Finally, we will also take advantage of the following result, which can be found in [9]; to make the exposition more self-contained, below we include a proof, which is simpler than that in [9].

Lemma 2.4. *Let G be a cograph on n vertices such that the set M is empty. Let v be the vertex in L which has the maximum number of neighbors in H, and let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ be the connected components of $G[V(G) - N[v]]$. If there exists a component $\mathcal{C}_i$ such that $|\mathcal{C}_i| > \frac{3}{4}n$, then the cardinality of each co-component of the graph $G[\mathcal{C}_i]$ does not exceed $\frac{1}{2}n$.*

Proof: Observe that every vertex $x \in H - \mathcal{C}_i$ is adjacent to at least one vertex of $\mathcal{C}_i$; if not, then the degree of x would be at most equal to $n - |\mathcal{C}_i| < \frac{1}{4}n$, which leads to a contradiction. In fact, such a vertex x sees the entire $\mathcal{C}_i$; this follows from Lemma 2.1, taking into account that x belongs to a co-component of $G[N(v)]$, since x is adjacent to a vertex in $\mathcal{C}_i$ and it does not belong to $\mathcal{C}_i$. Additionally, $\mathcal{C}_i$ contains at least one high-degree vertex; otherwise, it would be (a subset of) a connected component of $G[L]$ and then, by Lemma 2.3, its cardinality would be at most $\frac{2}{4}n$ (note that L is equal to the set L_a in Lemma 2.3 for $a = 4$). Finally, we show that $\mathcal{C}_i$ contains no low-degree vertices. Suppose that there existed such a vertex z. Since $\mathcal{C}_i$ is connected and contains a high-degree vertex, there would exist a path from z to that high-degree vertex in $G[\mathcal{C}_i]$; clearly, such a path would contain an edge connecting a low-degree vertex, say, w, to a high-degree vertex. Then, w is adjacent to all the high-degree vertices in $H - \mathcal{C}_i$ and to at least one high-degree vertex in $\mathcal{C}_i$. Since $H \cap N(v) \subseteq H - \mathcal{C}_i$, this contradicts the choice of v as the low-degree vertex that has the maximum number of neighbors in H. Therefore, $\mathcal{C}_i$ contains only high-degree vertices. Then, in the complement of G, the vertices of $\mathcal{C}_i$ belong to the low-degree vertex set L' of $\overline{G}$ and the co-components of $G[\mathcal{C}_i]$ would be subsets of the connected components of $\overline{G}[L']$; Lemma 2.3 implies that the cardinality of any such co-component would not exceed $\frac{2}{4}n$. ∎

The following lemma will be useful both in our cograph recognition algorithm as well as in our algorithm for constructing the cotree of a cograph.

Lemma 2.5. *Let G be a cograph, v a vertex of G, $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ the co-connected components of $G[N(v)]$, and $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ the connected components of $G[V(G) - N[v]]$. For each co-component $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, we define the set $\widehat{I}_i = \{ j \mid \widehat{\mathcal{C}}_i \text{ sees } \mathcal{C}_j \}$. Then, the following hold:*

(i) The co-components of $G[N(v)]$ have the following monotonicity property: $|\widehat{I}_i| \leq |\widehat{I}_j|$ implies that $\widehat{I}_i \subseteq \widehat{I}_j$.

(ii) Suppose that the ordering of the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ corresponds to their ordering by non-decreasing $|\widehat{I}_i|$. If we associate each component $\mathcal{C}_i$, $1 \leq i \leq k$, with the set $I_i = \{ j \mid \mathcal{C}_i \text{ sees } \widehat{\mathcal{C}}_j \}$, then the components of $G[V(G) - N[v]]$ have the following property: if $|I_i| \neq \emptyset$ and h is the minimum element of I_i, then $I_i = \{h, h+1, \ldots, \ell\}$.

Proof: (i) Suppose for contradiction that there exist co-components $\widehat{\mathcal{C}}_i$ and $\widehat{\mathcal{C}}_j$ such that $|\widehat{I}_i| \leq |\widehat{I}_j|$ and $\widehat{I}_i \not\subseteq \widehat{I}_j$. Then, there exists $t \in \widehat{I}_i - \widehat{I}_j$, which implies that $\widehat{\mathcal{C}}_i$ sees $\mathcal{C}_t$ whereas $\widehat{\mathcal{C}}_j$ misses $\mathcal{C}_t$. Additionally, since $|\widehat{I}_i| \leq |\widehat{I}_j|$ and $t \in \widehat{I}_i - \widehat{I}_j$, there exists $t' \in \widehat{I}_j - \widehat{I}_i$, which in turn implies that $\widehat{\mathcal{C}}_j$ sees $\mathcal{C}_{t'}$ whereas $\widehat{\mathcal{C}}_i$ misses $\mathcal{C}_{t'}$. But then, any four vertices a, b, c, d, such that $a \in \mathcal{C}_t$, $b \in \widehat{\mathcal{C}}_i$, $c \in \widehat{\mathcal{C}}_j$, and $d \in \mathcal{C}_{t'}$, induce a P_4 $abcd$ in G; a contradiction.

(ii) For any component $\mathcal{C}_i$ such that $I_i \neq \emptyset$, it suffices to show that $\forall j > h$, $\mathcal{C}_i$ sees $\widehat{\mathcal{C}}_j$. Consider any such j; since $j > h$, it holds that $|\widehat{I}_h| \leq |\widehat{I}_j|$, which according to statement (i) yields that $\widehat{I}_h \subseteq \widehat{I}_j$. From the definition of h, we have that $\mathcal{C}_i$ sees $\widehat{\mathcal{C}}_h$, or equivalently that $\widehat{\mathcal{C}}_h$ sees $\mathcal{C}_i$; that is, $i \in \widehat{I}_h$. Since $\widehat{I}_h \subseteq \widehat{I}_j$, then $i \in \widehat{I}_j$, i.e., $\mathcal{C}_i$ sees $\widehat{\mathcal{C}}_j$. ∎

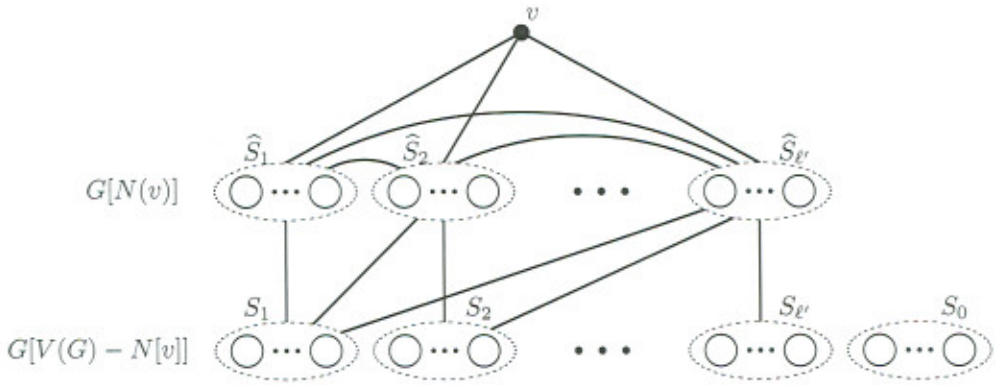

Figure 1

Consider the partition of the set of co-components $\{\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell\}$ of the subgraph $G[N(v)]$ into a collection of sets where any two co-components $\widehat{\mathcal{C}}_i, \widehat{\mathcal{C}}_j$ belong to the same set if and only if $\widehat{I}_i = \widehat{I}_j$, i.e., $\widehat{\mathcal{C}}_i$ and $\widehat{\mathcal{C}}_j$ see the same components of the subgraph $G[V(G) - N[v]]$. Let us denote these partition sets $\widehat{S}_1, \widehat{S}_2, \ldots, \widehat{S}_{\ell'}$, where, for every i, j such that $1 \leq i < j \leq \ell'$, and every $\widehat{\mathcal{C}}_r \in \widehat{S}_i$ and $\widehat{\mathcal{C}}_s \in \widehat{S}_j$, $\widehat{I}_r \subset \widehat{I}_s$; the value ℓ' is equal to the distinct values of the $\widehat{I}_i$s, and thus each set $\widehat{S}_j$ is nonempty.

In light of the above partition and due to Lemma 2.5 (statement (ii)), we can partition the set of connected components $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ of the subgraph $G[V(G) - N[v]]$ into sets $S_0, S_1, \ldots, S_{\ell'}$ as follows:

$$
\begin{aligned}
S_1 &= \{\, \mathcal{C}_j \mid \forall \widehat{\mathcal{C}} \in \widehat{S}_1,\ \mathcal{C}_j \text{ sees } \widehat{\mathcal{C}} \,\} \\
S_i &= \{\, \mathcal{C}_j \mid \forall \widehat{\mathcal{C}} \in \widehat{S}_i \text{ and } \widehat{\mathcal{C}}' \in \widehat{S}_{i-1},\ \mathcal{C}_j \text{ sees } \widehat{\mathcal{C}} \text{ but does not see } \widehat{\mathcal{C}}' \,\} \qquad (2 \leq i \leq \ell') \\
S_0 &= \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\} - \textstyle\bigcup_{i=1,\ldots,\ell'} S_i
\end{aligned}
$$

The definition of the sets $\widehat{S}_j$, $j = 1, 2, \ldots, \ell'$, implies that $S_i \neq \emptyset$ for all $i = 2, 3, \ldots, \ell'$. However, S_0 and S_1 may be empty; in particular, S_0 is empty if and only if the graph G is connected. The partitions of the co-components and components described above are illustrated in Figure 1; the dotted ovals indicate the partition sets, and the circles inside the ovals indicate the components or co-components belonging to the partition set. In terms of these partitions, the cotree of a cograph G has a very special structure, which is described in the following lemma.

Lemma 2.6. *Let G be a cograph, v a vertex of G, and $\widehat{S}_1, \widehat{S}_2, \ldots, \widehat{S}_{\ell'}$ and $S_0, S_1, \ldots, S_{\ell'}$ respectively the partitions of the co-connected components of $G[N(v)]$ and of the connected components of $G[V(G) - N[v]]$ as described above. Then,*

(i) if $S_1 = \emptyset$, the cotree of G has the general form depicted in Figure 2(a);

(ii) if $S_1 \neq \emptyset$, the cotree of G has the general form depicted in Figure 2(b).

In either case, the dashed part appears in the tree if and only if $S_0 \neq \emptyset$.

The circular nodes labeled with a 0 or a 1 in Figure 2 are 0-nodes and 1-nodes respectively, whereas the shaded node is a leaf node; the triangles denote the cotrees of the corresponding connected components or co-components. Lemma 2.6 gives us a way of constructing the cotree of an input cograph G: we compute the partitions $\widehat{S}_1, \ldots, \widehat{S}_{\ell'}$ and $S_0, S_1, \ldots, S_{\ell'}$; we recursively construct the

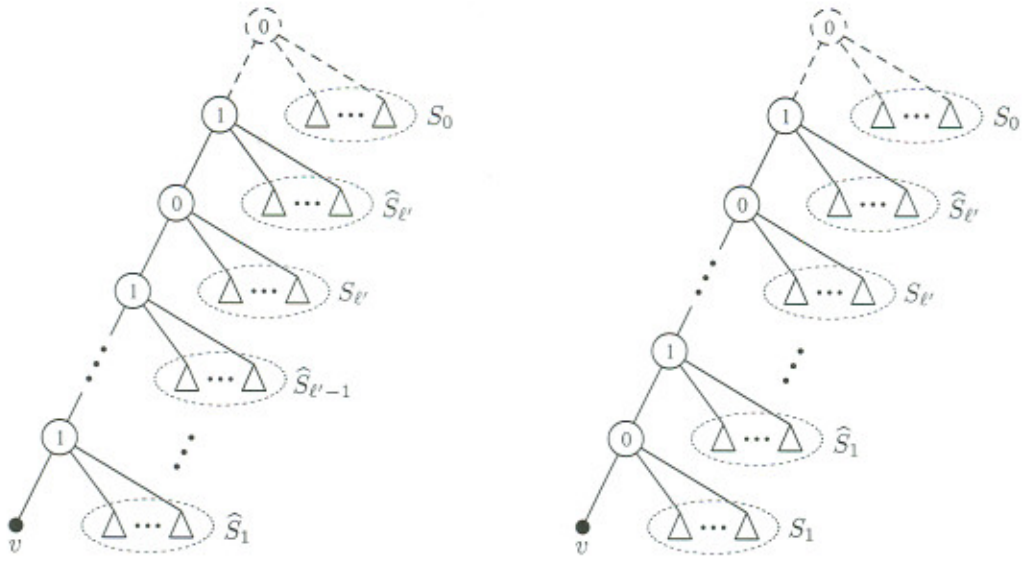

Figure 2

cotrees of the elements of each of the above partition sets; we link these cotrees as indicated in Figure 2. By carefully selecting the vertex v, we can guarantee that the cotree construction takes $O(\log^2 n)$ time, where n is the number of vertices of G.

3 Finding Connected Components or Detecting a P_4

In this section, we present a parallel algorithm which takes as input a graph and computes its connected components or detects that the graph contains a P_4 as an induced subgraph.

Let G be an undirected graph on n vertices and m edges, and suppose without loss of generality that $V(G) = \{1, 2, \ldots, n\}$. We define the function $f : V(G) \to V(G)$ as follows: $f(v) = \min\{u \mid u \in N[v]\}$. The function f is well defined since, for any vertex v, $N[v] \neq \emptyset$; additionally, the following properties hold:

(P1) For any vertex $v \in V(G)$, $f(v)$ is the minimum-index vertex at distance at most 1 from v.

(P2) Let us define $f^{(k)}(v)$ as follows: $f^{(1)}(v) = f(v)$, $f^{(k)}(v) = f(f^{(k-1)}(v))$. Then, for any vertex $v \in V(G)$, $f^{(k)}(v)$ is the minimum-index vertex at distance at most k from v, or equivalently $f^{(k)}(v) = \min\{u \mid u \in \underbrace{N[N[\ldots N[}_{k} v] \ldots]]\}$.

(P3) Any two vertices $u, v \in V(G)$, for which $f(u) = f(v)$, belong to the same connected component of G.

(P4) If u, v, w are distinct vertices of G such that $f(u) = v$ and $f(v) = w$, then the vertices u, v and w induce a P_3 uvw in G.

Property P1 follows trivially from the definition of $f(v)$; Property P2 is easily established by induction on k. Property P3 is a consequence of Property P2, whereas Property P4 follows from Property P1 and the fact that in such a case $v < u < w$.

Lemma 3.1. *Let G be an undirected graph, f the function defined above, and $V_1, V_2, \ldots, V_k$ the partition of $V(G)$ such that any two vertices x, y belong to the same partition set iff $f(f(x)) = f(f(y))$. Then, the following statements hold:*

(i) All the vertices in each V_i belong to the same connected component.

(ii) If there exists an edge $xy \in E(G)$ such that $x \in V_i$, $y \in V_j$, and $i \neq j$, then G contains a P_4 as an induced subgraph.

(iii) If the length of every induced path in G does not exceed 2, the sets $V_1, V_2, \ldots, V_k$ are the connected components of G.

Proof: (i) Clearly true, since, by Property P2, for all vertices $x, y \in V(G)$ such that $f(f(x)) = f(f(y)) = z$, G contains paths (of length at most 2) from x to z and from y to z.
(ii) Suppose that there exists such an edge xy, and assume without loss of generality that $f(f(x)) > f(f(y)) = z$. Then, Property P2 implies that $z \in N[N[y]]$ and Property P1 that $z \notin N[N[x]]$, which in turn implies that $z \notin N[y]$. Since $z \in N[N[y]]$ and $z \notin N[y]$, there exists a vertex $w \in N(y)$ such that y, w, z induce a P_3 ywz in G. Then, the fact that neither z nor w are adjacent to x (otherwise, $z \in N[N[x]]$) implies that the graph G contains the P_4 $xywz$ as an induced subgraph.
(iii) If every induced path in G has length at most 2, then, for every vertex $x \in V(G)$, the set $N[N[x]]$ coincides with the vertex set of the connected component of G to which x belongs. That is, for every vertex x in a connected component C_i of G, $f(f(x)) = \min\{ u \mid u \in C_i \}$; the truth of statement (iii) follows. ∎

Our connected components algorithm relies on Lemma 3.1. It computes, for each vertex v of the input graph, the value of $f(f(v))$, and then checks whether there exist two adjacent vertices v and u such that $f(f(v)) \neq f(f(u))$; if yes, it reports that the graph contains a P_4, otherwise, based on the values of $f(f())$, it generates an output array `comp[]` of size n such that `comp[v]` is equal to a representative of the connected component containing v. The algorithm uses two auxiliary arrays $A[]$ and $B[]$ of size equal to the number of vertices of the input graph which store the values of $f()$ and $f(f())$ respectively.

Algorithm Components-or-P4

Input: an undirected graph G on n vertices and m edges.

Output: either a message that G contains a P_4 as an induced subgraph or an array `comp[]`.

1. For each vertex $v \in V(G)$ do in parallel
   $A[v] \leftarrow v$;

2. For each vertex $v \in V(G)$ do in parallel
   $A[v] \leftarrow \min\{A[u] \mid u \in N[v]\}$;
   $B[v] \leftarrow \min\{A[u] \mid u \in N[v]\}$;

3. For each edge $uv \in E(G)$ do in parallel
   if $B[u] \neq B[v]$
   then print that G contains a P_4 as an induced subgraph; exit;

4. For each vertex $v \in V(G)$ do in parallel
   `comp[v]` $\leftarrow B[v]$;

The correctness of the algorithm is a direct consequence of Lemma 3.1.

Time and Processor Complexity. Next, we analyze the time and processor complexity of the algorithm; for details on the PRAM techniques mentioned below, see [3, 12, 22]. We assume that the input graph G is given in adjacency list representation.

Step 1: Clearly, the assignment operation performed in Step 1 can be executed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

Step 2: In order to compute the new value of $A[v]$ for each vertex $v \in V(G)$ avoiding concurrent read operations, we use for each vertex v an auxiliary array $A_v[\,]$ of size equal to the degree $deg(v)$ of v in G. We also use another auxiliary array $W[\,]$ of size $n \times n$; it must be noted that, although $W[\,]$ has n^2 entries, only $O(m)$ of these will be processed. Then, the computation of $A[v]$ is carried out as follows:

▷ For each vertex $v \in V(G)$ do in parallel

2.1 for each vertex u in the adjacency list $List(v)$ of v do in parallel
compute the rank $r_v(u)$ of the record of u in $List(v)$;
$deg(v) \leftarrow \max_u\{r_v(u)\}$;

2.2 copy the value $A[v]$ (as initialized in Step 1) to each of the $deg(v)$ entries of $A_v[\,]$;

2.3 for each vertex u in the adjacency list $List(v)$ of v do in parallel
$W[v,u] \leftarrow A_v[r_v(u)]$;
$A_v[r_v(u)] \leftarrow \min\{W[v,u], W[u,v]\}$;

2.4 $A[v] \leftarrow \min\{A_v[i] \mid 1 \leq i \leq deg(v)\}$;

Clearly, by taking advantage of the "twin" entries $W[v,u]$ and $W[u,v]$ in Substep 2.3, we ensure that $A[v]$ is correctly updated. In Substep 2.1, the ranks of the elements of $List(v)$ and their maximum can be optimally computed in $O(\log deg(v))$ time using $O(deg(v)/\log deg(v))$ processors, or in $O(\log n)$ time using $O(deg(v)/\log n)$ processors, on the EREW PRAM model. Substeps 2.2, 2.3, and 2.4 can also be executed without concurrent read or write operations in $O(\log n)$ time with $O(deg(v)/\log n)$ processors. Thus, the computation of the values $A[v]$ for all vertices $v \in V(G)$ can be done in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model. Since the rest of Step 2, i.e., the updating of the array $B[\,]$, is executed in the very same way, the entire step takes $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 3: Here, we have to check whether there exists an edge uv of G such that $B[u] \neq B[v]$. For an EREW execution, we use the $n \times n$ array $W[\,]$ mentioned in the analysis of Step 2, and for each vertex $v \in V(G)$, an auxiliary array $B_v[\,]$ of size equal to the degree $deg(v)$ of v.

▷ For each vertex $v \in V(G)$ do in parallel

3.1 copy the value $B[v]$ (as computed in Step 2) to each of the $deg(v)$ entries of $B_v[\,]$;

3.2 for each vertex u in the adjacency list $List(v)$ of v do in parallel
$W[v,u] \leftarrow B_v[r_v(u)]$, where $r_v(u)$ is the rank of the record of u in $List(v)$;
if $W[v,u] \neq W[u,v]$
then $B_v[r_v(u)] \leftarrow 0$;

3.3 if $\min\{B_v[i] \mid 1 \leq i \leq deg(v)\} = 0$
then print that G contains a P_4 as an induced subgraph;

Note that $W[v,u] \neq W[u,v]$ iff $B_v[r_v(u)] \neq B_u[r_u(v)]$, or equivalently, $B[v] \neq B[u]$. Using parallel techniques similar to those used in Step 2, it is easy to see that the entire step for all vertices $v \in V(G)$ can be executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 4: The assignment operations performed in this step are executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model.

Taking into consideration Lemma 3.1 and the time and processor complexity of each step of the algorithm, we obtain the following result.

Theorem 3.1. *When applied on a graph G on n vertices and m edges, Algorithm Components-or-P4 either detects that G contains a P_4 as an induced subgraph or computes G's connected components in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.*

It must be noted that the goal of Algorithm Components-or-P4 is not to detect whether the input graph contains a P_4. So, in some cases, it terminates without reporting that the graph contains a P_4 even if this is so; in any such case, however, it correctly reports the connected components of the given graph.

Finally, it is worth mentioning that the main idea employed by the Algorithm Components-or-P4 can be used to yield an optimal parallel computation of the connected components of any graph with constant diameter. For any graph with diameter at most some constant d, it suffices to replace the body of the for-loop in Step 2 of the algorithm by the sequential execution of d computations of the form "$A[v] \leftarrow \min\{A[u] \mid u \in N[v]\}$" and ignore Step 3. The resulting algorithm clearly runs in $O(d \log n) = O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM.

Remark 3.1: Computing the representatives of the connected components. Let G be a graph on n vertices and let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ be its connected components. If the Algorithm Components-or-P4 does not report the existence of a P_4 in G, it computes G's connected components and stores the information in the array `comp[]` of size n so that for each $v \in \mathcal{C}_i$, `comp[v]` is equal to the representative of the connected component $\mathcal{C}_i$; in fact, the representatives $v_1, v_2, \ldots, v_k$ of the connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ are such that $v_i = \min\{v \in \mathcal{C}_i\}$, $1 \leq i \leq k$. The representatives can be isolated in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model as follows: we use an array $R[\,]$ of size n such that $R[v] = v$ if `comp[v]` $= v$ and $R[v] = 0$ otherwise; then, by using prefix computation and array packing techniques on $R[\,]$, we can collect the representatives $v_1, v_2, \ldots, v_k$ into the first k positions of the array $R[\,]$ —that is, $R[i] = v_i$ for $1 \leq i \leq k$.

Remark 3.2: Collecting the vertices of each connected component. Let $v_1, v_2, \ldots, v_k$ be the representatives of the connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of the input graph G, which have been computed by Algorithm Components-or-P4. We are interested in collecting the vertices of each connected component.

First, it is important to observe that if the Algorithm Components-or-P4 has terminated and reported that it has computed the connected components of G, then every pair of adjacent vertices of G have the same value of $B[\,]$. Additionally, in order to ensure that each vertex will be collected exactly once, during the computation of $B[v]$ in Step 2 of the algorithm, we keep track of the vertex that has contributed the minimum in the computation of $B[v]$, and we break ties in favor of the lowest-index vertex; let us denote this vertex by $p(v)$. Then, the definition of the quantity $p(\,)$ implies that the following hold:

- for each representative v_i, it holds that $p(v_i) = v_i$; for any other vertex v, $p(v) \neq v$;
- if the quantity $p(v)$ is interpreted as the "parent" of vertex v, then, the pairs $(v, p(v))$ form a tree in parent-pointer representation.

As in the description of the Algorithm Components-or-P4, we assume that the input graph G is given in adjacency-list representation, and that $List(v)$ denotes the adjacency list of vertex v. We use an auxiliary array $W[\,]$ of size $n \times n$ (as in Step 2 of the Algorithm Components-or-P4), and, for each vertex v, an array $T_v[\,]$ of size equal to the degree $deg(v)$ of v in G. Then, the vertices of each of the connected components $\mathcal{C}_i$, $1 \leq i \leq k$, can be collected as follows:

1. For each vertex $v \in V(G)$ do in parallel

   1.1 for each vertex u in the adjacency list $List(v)$ of v do in parallel
   compute the rank $r_v(u)$ of the record of u in $List(v)$;
   $deg(v) \leftarrow \max_u\{r_v(u)\}$;

   1.2 copy the value $p(v)$ to each of the $deg(v)$ entries of $T_v[\,]$;

   1.3 for each vertex u in the adjacency list $List(v)$ of v do in parallel
   $W[v,u] \leftarrow T_v[r_v(u)]$;
   $p \leftarrow W[u,v]$; $\{p = p(u)\}$
   if $p \neq v$
   then mark the record of u as useless;
   else insert the adjacency list $List(u)$ of u right after the record of u in $List(v)$;

2. for each vertex representative v_i, $1 \leq i \leq k$, do in parallel
   compute the ranks of the vertex records in the (augmented) adjacency list of v_i;
   copy the contents of the adjacency list to an array;
   pack the array while ignoring vertices that have been marked as useless;

For $1 \leq i \leq k$, the resulting packed array associated with vertex v_i contains each of the vertices in $\mathcal{C}_i - \{v_i\}$ exactly once; adding an entry for v_i yields the entire set of vertices of the connected component $\mathcal{C}_i$. It is easy to see that the above computation can be carried out using standard and simple parallel techniques in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Having computed the vertices of each connected component $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of the graph G, we can also compute the adjacency-list representation of each induced subgraph $G[\mathcal{C}_1], G[\mathcal{C}_2], \ldots, G[\mathcal{C}_k]$ within the same time and processor bounds using the same model of computation.

4 The Recognition Algorithm

In this section, we present a parallel algorithm for recognizing whether a given graph G is a cograph. As in the description of the parallel connectivity algorithm of the previous section, we assume that G is given in adjacency-list representation. We also assume that for each edge uv of G, the two records in the adjacency lists of u and v are linked together; this helps us re-index the vertices in subgraphs of the given graph fast.

Algorithm Recognize-Cograph

Input: an undirected graph.
Output: a message either that the input graph is a cograph or that it is not.

1. Call the subroutine *Process-Graph* on the input graph;

2. Print that the input graph is a cograph.

Subroutine Process-Graph

Input: an undirected graph G on n vertices and m edges.

1. Compute the sets L, M and H containing the low-, middle-, and high-degree vertices of G, respectively;
2. If $L = \emptyset$ and $M = \emptyset$ then $\{$*each* $v \in V(G)$ *has degree* $> \frac{3}{4}n\}$
   (a) compute the co-components of G;
   (b) if any of these co-components has cardinality exceeding $\frac{3}{4}n$
   then $\{G$ *is not a cograph*$\}$
   print that the input graph is not a cograph; exit;
   else compute the subgraphs induced by each of the co-components and
   call recursively the subroutine *Process-Graph* on each of these subgraphs;
   return;
3. If $M \neq \emptyset$
   then $v \leftarrow$ an arbitrary vertex of M;
   else $v \leftarrow$ the vertex in L with the maximum number of neighbors in H; $\{$*note:* $L \neq \emptyset\}$
4. Compute the following induced subgraphs G_1 and G_2 of the graph G:
   (a) $G_1 = G[N(v)]$;
   (b) $G_2 = G[V(G) - N[v]]$;
5. Compute the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ of the graph G_1;
6. Use the algorithm Components-or-P4 on the graph G_2 in order either to detect that it contains an induced P_4 or to compute its connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$;
   if G_2 contains an induced P_4
   then print that the input graph is not a cograph; exit;
7. If there exists a pair $\widehat{\mathcal{C}}_i, \mathcal{C}_j$ such that the co-component $\widehat{\mathcal{C}}_i$ neither sees nor misses the connected component $\mathcal{C}_j$
   then $\{G$ *contains an induced* P_4 *with vertices in* G_1 *and* $G_2\}$
   print that the input graph is not a cograph; exit;
8. Sort the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ of the graph G_1 in non-decreasing number of the connected components of the graph G_2 that each co-component sees;
   let $\widehat{S} = (\widehat{\mathcal{C}}_{\pi(1)}, \widehat{\mathcal{C}}_{\pi(2)}, \ldots, \widehat{\mathcal{C}}_{\pi(\ell)})$ be the sorted list;
9. If there exist two consecutive co-components $\widehat{\mathcal{C}}_{\pi(i)}$ and $\widehat{\mathcal{C}}_{\pi(i+1)}$ in $\widehat{S}$, where $1 \leq i < \ell$, such that $\widehat{\mathcal{C}}_{\pi(i)}$ sees a connected component of the graph G_2 which $\widehat{\mathcal{C}}_{\pi(i+1)}$ misses
   then $\{G$ *contains an induced* P_4 *with its midpoints in* G_1 *and its endpoints in* $G_2\}$
   print that the input graph is not a cograph; exit;
10. Compute the induced subgraphs $G[\widehat{\mathcal{C}}_i]$, $1 \leq i \leq \ell$, and $G[\mathcal{C}_j]$, $1 \leq j \leq k$;
11. Call recursively the subroutine *Process-Graph* on each $G[\widehat{\mathcal{C}}_i]$ $(1 \leq i \leq \ell)$, and on each $G[\mathcal{C}_j]$ $(1 \leq j \leq k)$ such that $|\mathcal{C}_j| \leq \frac{3}{4}n$;
    if there exists a $\mathcal{C}_j$ such that $|\mathcal{C}_j| > \frac{3}{4}n$ then call recursively the subroutine *Process-Graph* on each of the subgraphs induced by the co-components of $G[\mathcal{C}_j]$;

Observe that if at any point any of the subgraphs on which we apply the subroutine *Process-Graph* is found not to be a cograph, then the subroutine immediately prints that the input graph is not a cograph and exits. Additionally, it is important to note that the following lemma holds.

Lemma 4.1. *If during the execution of the subroutine Process-Graph on a graph G on n vertices a recursive call is made on a graph G', then G' is a subgraph of G and the number of vertices of G' does not exceed $\frac{3}{4}n$.*

Proof: Recursive calls are executed in Steps 2 and 11 of the subroutine Process-Graph. In either case, the graphs on which the calls are executed are subgraphs of G. Moreover, when Process-Graph is called on a subgraph induced by a co-component of the input graph G in Step 2, then the number of vertices of the subgraph does not exceed $\frac{3}{4}n$. The same clearly holds when it is called on a subgraph $G[\mathcal{C}_j]$ $(1 \leq j \leq k)$ such that $|\mathcal{C}_j| \leq \frac{3}{4}n$ in Step 11. For the case when Process-Graph is called on a subgraph $G[\widehat{\mathcal{C}}_i]$ $(1 \leq i \leq \ell)$ in Step 11, we note that $|\widehat{\mathcal{C}}_i| \leq |N(v)| \leq \frac{3}{4}n$, since v belongs to the set M or the set L. Finally, if in Step 11 there exists a component $\mathcal{C}_j$ such that $|\mathcal{C}_j| > \frac{3}{4}n$, then, in light of Observation 2.2, $v \notin M$, which implies that $M = \emptyset$; but then, by Lemma 2.4, the cardinality of each co-component of $G[\widehat{\mathcal{C}}_j]$ does not exceed $\frac{1}{2}n$. ∎

Correctness. The correctness of the messages printed by Subroutine Process-Graph in Steps 2, 6, and 7 readily follows from Lemma 2.2, from the definition of the cographs, and from Lemma 2.1, respectively. The correctness of the message printed in Step 9 also follows from Lemma 2.1: if there exists a co-component $\widehat{\mathcal{C}}_{\pi(i)}$ which sees a connected component, say, $\mathcal{C}$, of the graph G_2 which the co-component $\widehat{\mathcal{C}}_{\pi(i+1)}$ misses, then because the co-components have been ordered by non-decreasing number of components they see, the co-component $\widehat{\mathcal{C}}_{\pi(i+1)}$ must see another component of G_2, say, $\mathcal{C}'$, that $\widehat{\mathcal{C}}_{\pi(i)}$ misses; then, neither $N(\mathcal{C}) \subseteq N(\mathcal{C}')$ nor $N(\mathcal{C}) \supseteq N(\mathcal{C}')$, which implies that the input graph is not a cograph according to Lemma 2.1, statement (ii). On the other hand, both in Steps 2 and 11 where recursive calls are executed, the graphs on which the recursive calls are made are components or co-components of the graph G and their vertex sets form a partition of the vertex set of G, so that G is a cograph iff each of these graphs is a cograph.

Time and Processor Complexity. It suffices to analyze the time and processor complexity of the subroutine Process-Graph; clearly, the time and processor complexities of the application of the algorithm Recognize-Cograph on a graph G are identical to the respective complexities of the application of the subroutine Process-Graph on G. For details on the PRAM techniques mentioned below, see [3, 12, 22].

Step 1: The computation of the degree $deg(v)$ of a vertex v of the graph G can be done by applying list ranking on the adjacency list of v and by taking the maximum rank; this can be done in $O(\log n)$ time using $O(deg(v)/\log n)$ processors on the EREW PRAM. The computation for all the vertices takes $O(\log n)$ time and $O(m/\log n)$ processors on the same model of computation. Locating the low-degree vertices of the graph G, i.e., all the vertices $v \in V(G)$ such that $deg(v) < \frac{1}{4}n$, can be done in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model: we use an auxiliary array $Low[\,]$ of size n and we set $Low[v] = v$ if the vertex v has degree $deg(v) < \frac{1}{4}n$ and $Low[v] = 0$ otherwise; then, the low-degree vertices of G can be collected by means of array packing on $Low[\,]$ using prefix computation. The middle- and the high-degree vertices of G can be collected in a similar fashion within the same time-processor bound.

Step 2: Since we use an array representation for each of the vertex sets L, M, and H, we can check whether such a set contains a vertex (or it is an empty set) in constant sequential time. The co-components of G can be computed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model [6], and so can the subgraphs of G induced by each of these co-components.

Step 3: Since each of the vertex sets L, M, and H is given in array representation, this step is clearly executed in constant sequential time if $M \neq \emptyset$: we take $v \leftarrow M[1]$. If $M = \emptyset$, it is executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model: for each vertex w in L, we mark the high-degree vertices in w's adjacency list and compute the number of marked vertices; then, we compute the maximum of these numbers over all vertices in L and select as v a vertex whose number of marked vertices in its adjacency list equals the maximum.

Step 4: Let $List(v)$ be the adjacency list of the vertex v, and let $r_v(u)$ denote the rank of the vertex u in the list $List(v)$. For each vertex $v \in V(G)$, we use two auxiliary arrays $A_v[\,]$ and $B_v[\,]$, each of size equal to the degree $deg(v)$ of v. Then, the adjacency-list representation of the graph $G_1 = G[N(v)]$ is computed, as follows:

▷ For each vertex $x \in V(G)$ do in parallel

4.1 for each vertex y in the adjacency list $List(x)$ of x do in parallel
$A_x[r_x(y)] \leftarrow y$;

4.2 if the vertex x belongs to $N(v)$
then copy the value 1 to each of the $deg(x)$ entries of $B_x[\,]$;
else copy the value 0 to each of the $deg(x)$ entries of $B_x[\,]$;

▷ For each vertex w in the adjacency list $List(v)$ of v do in parallel

4.3 for $i = 1, 2, \ldots, deg(w)$ do in parallel
$u \leftarrow A_w[i]$;
if $B_u[r_u(w)] = 0$, then mark the entry $A_w[r_w(u)]$;

4.4 store the unmarked elements of the array $A_w[\,]$ in consecutive locations, and, then, construct a list of these vertices and associate it with vertex $w \in V(G_1)$;

Since $B_u[r_u(w)] = 0$ if and only if $u \notin N(v)$, it is not difficult to see that the resulting lists for all the vertices $w \in N(v)$ form an adjacency-list representation of the induced subgraph G_1. The computation of the list representation of the induced subgraph G_2 is done in a fashion similar to that previously described. Using standard and simple parallel techniques, such as interval broadcasting and array packing, it is easy to see that the linked list representation of both graphs G_1 and G_2 can be computed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 5: The computation of the co-components of a graph on n vertices and m edges can be optimally done in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model [6].

Step 6: Here, we use the algorithm Components-or-P4 that we have presented in Section 3, and either detect that the graph G_2 contains a P_4 as an induced subgraph or compute the connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of G_2. Thus, the step is executed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 7: In this step, we check whether for each pair $\widehat{\mathcal{C}}_i, \mathcal{C}_j$, the co-component $\widehat{\mathcal{C}}_i$ either sees or misses the connected component $\mathcal{C}_j$, where $1 \leq i \leq \ell$ and $1 \leq j \leq k$. To do that, we first construct a subgraph G^* of the graph G as follows:

$$V(G^*) = V(G) - \{v\}$$
$$E(G^*) = \{xy \mid x \in V(G_1), y \in V(G_2)\};$$

recall that the induced graphs $G_1 = G[N(v)]$ and $G_2 = G[V(G) - N[v]]$, for $v \in V(G)$, have been computed in Step 4. It is easy to see that an adjacency-list representation of G^* can be constructed from the graph G in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model. By taking advantage of the graph G^*, Step 7 is equivalent to performing the following two steps:

- for each co-component $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, check whether all the vertices of $\widehat{\mathcal{C}}_i$ have identical neighborhoods in G^*;
- for each connected component $\mathcal{C}_j$, $1 \leq j \leq k$, check whether all the vertices of $\mathcal{C}_j$ have identical neighborhoods in G^*.

Next, we show how to do the former step; the latter is done in a similar fashion. Let $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\ell$ and $\hat{n}_1, \hat{n}_2, \ldots, \hat{n}_\ell$ be the representatives and the number of vertices, respectively, of the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ of the graph G_1. For each co-component $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, we use an auxiliary array $D_i[\,]$ of size $\hat{n}_i - 1$, and arrays $P_i[\,]$, $P_{i,j}[\,]$ $(1 \leq j \leq \hat{n}_i - 1)$, and $M_{i,x}[\,]$ $(x \in \widehat{\mathcal{C}}_i - \{\hat{v}_i\})$, each of size equal to the degree $deg^*(\hat{v}_i)$ of the representative $\hat{v}_i$ in G^*. Then, the processing of the co-components is as follows:

▷ For each co-component $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, do in parallel

7.1 compute a linked list LC_i containing the vertices in $\widehat{\mathcal{C}}_i - \{\hat{v}_i\}$;

7.2 compute the degree $deg^*(\hat{v}_i)$ of $\hat{v}_i$ in G^*;

7.3 make $\hat{n}_i - 1$ copies of the degree $deg^*(\hat{v}_i)$ in an array $D_i[1..\hat{n}_i - 1]$;

7.4 For each vertex $x \in LC_i$, do in parallel
compute the degree $deg^*(x)$ of x in G^*;
if $deg^*(x) \neq D_i[r_i(x)]$, where $r_i(x)$ is the rank of x in LC_i,
then print that the input graph is not a cograph; exit;

7.5 copy the neighbors of $\hat{v}_i$ in G^* in the array $P_i[1..deg^*(\hat{v}_i)]$;

7.6 make $\hat{n}_i - 1$ copies $P_{i,1}[\,], \ldots, P_{i,\hat{n}_i-1}[\,]$ of the array $P_i[\,]$;

7.7 For each vertex $x \in LC_i$, do in parallel
copy the neighbors of x in G^* in the array $M_{i,x}[\,]$;
if $M_{i,x}[\,] \neq P_{i,r_i(x)}[\,]$, where $r_i(x)$ is the rank of x in LC_i,
then print that the input graph is not a cograph; exit;

The correctness of the computation follows from the fact that the input graph is reported as not being a cograph iff there exists a vertex x such that either the condition "$deg^*(x) \neq D_i[r_i(x)]$" in Substep 7.4 or the condition "$M_{i,x}[\,] \neq P_{i,r_i(x)}[\,]$" in Substep 7.7 is found true; the former condition is true iff the degrees of x and $\hat{v}_i$ in G^* differ, the latter iff the vertices x and $\hat{v}_i$ have different neighborhoods in G^*.

Substep 7.1 can be completed for all the co-components in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model in a fashion similar to the one described for the connected components of a graph in Remark 3.2. Moreover, since the graph G^* is given in adjacency-list representation and contains $n - 1$ vertices and $m^* \leq m$ edges, where n and m are the numbers of vertices and edges of the input graph G, all the operations in Substeps 7.2 through 7.4 can be executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

If the algorithm does not abort in Substep 7.4, then, for every vertex $x \in \widehat{\mathcal{C}}_i - \{\hat{v}_i\}$, $deg^*(x) = deg^*(\hat{v}_i)$. Since $deg^*(x)$ is less than the degree $deg(x)$ of x in G, it follows that Substeps 7.5 and 7.6 can be executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

The size of both the array $P_{i,r_i(x)}[\,]$ and the array $M_{i,x}[\,]$ is equal to $deg^*(x)$ and the if-statement can be easily checked in $O(\log deg^*(x))$ time with $O(deg^*(x)/\log deg^*(x))$ processors, or in $O(\log n)$ time with $O(deg^*(x)/\log n)$ processors, on the EREW PRAM model by means of an auxiliary array $B_{i,x}[\,]$ of size $deg^*(x)$ as well: for $j = 1, 2, \ldots, deg^*(x)$, if $M_{i,x}[j] \neq P_{i,r_i(x)}[j]$ then $B_{i,x}[j] \leftarrow 1$ else $B_{i,x}[j] \leftarrow 0$; next, we compute the maximum element of $B_{i,x}[\,]$, and $M_{i,x}[\,] \neq P_{i,r_i(x)}[\,]$ iff the maximum is equal to 1. Thus, Substeps 7.7 and 7.8 are executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

From the above time-processor analysis, we conclude that we can check whether all the vertices of each co-component have identical neighborhoods in G^* in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model. Processing the vertices of the connected components $\mathcal{C}_j$, $1 \leq j \leq k$, is done in a similar fashion; thus, the execution of Step 7 of the algorithm takes $O(\log n)$ time and requires $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 8: Here, we sort the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ in non-decreasing number of the connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ that each co-component sees. Let $(a_1, a_2, \ldots, a_\ell)$ be the list such that a_i is the number of the connected components that $\widehat{\mathcal{C}}_i$ sees; then, a_i is equal to the degree of the representative $\hat{v}_i$ of $\widehat{\mathcal{C}}_i$ in the subgraph of the graph G^* induced by the representatives of the co-components $\widehat{\mathcal{C}}_i$, $1 \leq i \leq \ell$, and the components $\mathcal{C}_j$, $1 \leq j \leq k$. Thus, the a_is can be computed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model. Since the number ℓ of co-components is $O(\sqrt{m})$ (Observation 2.1), sorting the a_is can be executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model; note that $\log \ell = O(\log m) = O(\log n)$, and that if $\sqrt{m} < \log n$ then $\sqrt{m} = O(n/\log n)$, whereas if $\sqrt{m} \geq \log n$ then $\sqrt{m} = O(m/\log n)$.

Step 9: For simplicity, we assume that $(\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell)$ is the sorted list of the co-components of the graph G_1, i.e, $\pi(i) = i$ for $i = 1, 2, \ldots, \ell$ (see Step 5). In a way similar to the one we used in order to compute the list $(a_1, a_2, \ldots, a_\ell)$ in the previous step, we compute the list $(b_1, b_2, \ldots, b_k)$ where b_i is the number of the co-components of the graph G_1 that the connected component $\mathcal{C}_i$ sees, $1 \leq i \leq k$. Then, we implement Step 9 as follows:

▷ For each connected component $\mathcal{C}_i$, $1 \leq i \leq k$, do in parallel

9.1 find the co-component $\widehat{\mathcal{C}}_h$ with the minimum index that the representative v_i of $\mathcal{C}_i$ sees;

9.2 if $b_i \neq \ell - h + 1$
then print that the input graph is not a cograph; exit;

The correctness of the computation follows from Lemma 2.5, statement (ii): note that if G is a cograph and the minimum-index co-component that the representative v_i of $\mathcal{C}_i$ sees is $\widehat{\mathcal{C}}_h$, then $\mathcal{C}_i$ sees each of the co-components $\widehat{\mathcal{C}}_h, \widehat{\mathcal{C}}_{h+1}, \ldots \widehat{\mathcal{C}}_\ell$, that is, it sees exactly $\ell - h + 1$ co-components. The computation of the list $(b_1, b_2, \ldots, b_k)$ takes $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model. Moreover, it is easy to see that the representative of the minimum-index co-component that each component $\mathcal{C}_i$ sees, can also be computed within the same time and processor bounds. Thus, Step 9 is executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 10: The induced subgraphs $G[\widehat{\mathcal{C}}_i]$, $1 \leq i \leq \ell$, and $G[\mathcal{C}_j]$, $1 \leq j \leq k$, can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 11: The co-components of any connected component $\mathcal{C}_j$ such that $|\mathcal{C}_j| > \frac{3}{4}n$ can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model [6], and as in Step 10 the subgraphs induced by them can be constructed within the same time and processor complexity.

Taking into consideration the time and processor complexity of each step of the subroutine *Process-Graph* and the recursive calls, we have that the time complexity $T(n,m)$ and processor complexity $P(n,m)$ of the algorithm *Recognize-Cograph* when applied on a graph on n vertices and m edges satisfy the following equalities:

$$\begin{aligned} T(n,m) &= O(\log n) + \max_i\{T(n_i,m_i)\} \\ P(n,m) &= \max\{O((n+m)/\log n), \sum_i P(n_i,m_i)\} \end{aligned}$$

where n_i and m_i are the numbers of vertices and edges of the subgraphs on which the subroutine *Process-Graph* is recursively called. Since $\sum_i n_i \leq n$, $\sum_i m_i \leq m$, and for each i, $n_i \leq 3n/4$ (see Lemma 4.1), the equalities for $T(n,m)$ and $P(n,m)$ admit the solution: $T(n,m) = O(\log^2 n)$, $P(n,m) = O((n+m)/\log n)$. Thus, we obtain the following result.

Theorem 4.1. *Algorithm Recognize-Cograph runs in $O(\log^2 n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.*

Corollary 4.1. *Cographs can be recognized in $O(\log^2 n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model of computation.*

5 Construction of the Cotree

In this section, we present a parallel algorithm for constructing the cotree of a cograph. The algorithm relies on Lemma 2.6 which gives the structure of the cotree of a cograph G in terms of the co-components of the subgraph $G[N(v)]$ and the connected components of the subgraph $G[V(G) - N[v]]$ for any vertex v of G. The algorithm selects an appropriate vertex v of the input graph G, recursively computes the cotrees of the subgraphs induced by the co-components of $G[N(v)]$ and the connected components of $G[V(G) - N[v]]$, and then uses Lemma 2.6 to link these cotrees in order to form the cotree of G. As in the case of the cograph recognition algorithm, we assume that the input graph is given in adjacency-list representation.

Algorithm Construct-Cotree

Input: a cograph G on n vertices and m edges.

Output: the root-node of the cotree $T(G)$ of the graph G.

1. Compute the sets L, M and H containing the low-, middle-, and high-degree vertices of the input graph G, respectively;
2. If $L = \emptyset$ and $M = \emptyset$ then $\quad$ {each $v \in V(G)$ has degree $deg(v) > \frac{3}{4}n$}
   (a) compute the co-components $\widehat{\mathcal{A}}_1, \widehat{\mathcal{A}}_2, \ldots, \widehat{\mathcal{A}}_p$ of the graph G;
   (b) construct a 1-node r;
   (c) for $i = 1, 2, \ldots, p$ do in parallel
   compute the induced subgraph $G[\widehat{\mathcal{A}}_i]$;
   apply recursively the algorithm *Construct-Cotree* on $G[\widehat{\mathcal{A}}_i]$; let $\hat{s}_i$ be the root-node of the returned tree;
   $parent(\hat{s}_i) \leftarrow r$;
   (d) return(r);

3. If $M \neq \emptyset$
   then $v \leftarrow$ an arbitrary vertex of M;
   else $v \leftarrow$ the vertex in L with the maximum number of neighbors in H; {*note:* $L \neq \emptyset$}

4. Compute the co-components $\widehat{\mathcal{C}}_1, \widehat{\mathcal{C}}_2, \ldots, \widehat{\mathcal{C}}_\ell$ of the graph $G_1 = G[N(v)]$;
   for $i = 1, 2, \ldots, \ell$ do in parallel
      compute the induced subgraph $G[\widehat{\mathcal{C}}_i]$;
      apply recursively the algorithm *Construct-Cotree* on $G[\widehat{\mathcal{C}}_i]$; let $\hat{r}_i$ be the root-node of the returned tree;

5. Compute the connected components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of the graph $G_2 = G[V(G) - N[v]]$;
   for $i = 1, 2, \ldots, k$ do in parallel
      compute the induced subgraph $G[\mathcal{C}_i]$;
      if $|\mathcal{C}_i| \leq \frac{3}{4}n$
      then apply recursively the algorithm *Construct-Cotree* on $G[\mathcal{C}_i]$; let r_i be the root-node of the returned tree;
      else construct a 1-node r_i;
         compute the co-components $\widehat{\mathcal{C}}_{i_j}$, $1 \leq j \leq h$, of the graph $G[\mathcal{C}_i]$ and the induced subgraphs $G[\widehat{\mathcal{C}}_{i_j}]$;
         for $j = 1, 2, \ldots, h$ do in parallel
            apply recursively the algorithm *Construct-Cotree* on $G[\widehat{\mathcal{C}}_{i_j}]$; let $\hat{t}_{i_j}$ be the root-node of the returned tree;
            $parent(\hat{t}_{i_j}) \leftarrow r_i$;

6. Compute the subgraph $\widetilde{G}$ of G spanned by the edges incident upon a co-component representative $\hat{v}_i$ $(1 \leq i \leq \ell)$ and a component representative v_j $(1 \leq j \leq k)$;
   compute the degrees of the $\hat{v}_i$s, $1 \leq i \leq \ell$, in $\widetilde{G}$, sort them in non-decreasing order, and locate the distinct values; let $\delta[i]$, $1 \leq i \leq \ell'$, be the resulting ordered sequence;

7. Compute the entries of an array $pos[i]$, $1 \leq i \leq \ell$, such that $pos[i] = j$ if and only if the degree of $\hat{v}_i$ in $\widetilde{G}$ is equal to $\delta[j]$;

8. Construct a tree-path of alternating 1- and 0-nodes as follows:

   (a) construct ℓ' 1-nodes $\hat{t}_i$, $1 \leq i \leq \ell'$, and ℓ' 0-nodes t_j, $1 \leq j \leq \ell'$;
      construct a leaf-node t storing v;

   (b) for $i = 1, \ldots, \ell' - 1$ do in parallel
         $parent(t_i) \leftarrow \hat{t}_i$;
         $parent(\hat{t}_i) \leftarrow t_{i+1}$;
      $parent(t_{\ell'}) \leftarrow \hat{t}_{\ell'}$;
      if $\delta[1] \neq 0$
      then $parent(t) \leftarrow t_1$;
      else $parent(t) \leftarrow \hat{t}_1$; delete node t_1;

9. Construct and return the following tree:

   (a) for $i = 1, 2, \ldots, \ell$ do in parallel
         $parent(\hat{r}_i) \leftarrow \hat{t}_{pos[i]}$;

   (b) for $i = 1, 2, \ldots, k$ do in parallel
         $parent(r_i) \leftarrow t_{p_i}$, where $p_i \leftarrow \min\{\, pos[j] \mid v_i$ is adjacent to $\hat{v}_j$ in $\widetilde{G}\}$;

(c) if there exist component representatives v_i in $\widetilde{G}$ of degree equal to 0
then construct a 0-node r;
for each component representative v_i of degree equal to 0 in $\widetilde{G}$
$parent(r_i) \leftarrow r$;
$parent(\hat{t}_{\ell'}) \leftarrow r$;
else $r \leftarrow \hat{t}_{\ell'}$;

(d) return(r);

The correctness of Steps 2 and 5 follows as in the case of the cograph recognition algorithm in Section 4, and from the fact that any two co-components of a graph see each other. The correctness of the rest of the algorithm directly follows from Lemma 2.6: note that, for $i = 1, 2, \ldots, \ell'$, the tree node $\hat{t}_i$ corresponds to the 1-node that is the parent of the roots of the cotrees of the co-components in the set $\widehat{S}_i$, and the tree node t_i corresponds to the 0-node that is the parent of the roots of the cotrees of the components in S_i (see Figure 2); additionally, $\delta[1] \neq 0$ if and only if $S_1 \neq \emptyset$ (Step 8(b)), while Step 9(c) takes care of the case when $S_0 \neq \emptyset$.

Time and Processor Complexity. We shall use a step-by-step analysis for computing the time and processor complexities of each step of Algorithm Construct-Cotree.

Steps 1–5: All the operations performed in these steps are also performed in Steps 1–3, 5, 6, and 10 of Algorithm Recognize-Cograph. Thus, it is easy to see that, if we ignore the time taken by the recursive calls, the execution of Steps 1–5 of Algorithm Construct-Cotree take $O(\log n)$ time and require $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 6: The subgraph $\widetilde{G}$ coincides with the subgraph of the graph G^* (see the analysis of Step 7 of Algorithm Recognize-Cograph) induced by the vertex set $\{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\ell, v_1, v_2, \ldots, v_k\}$, and can be constructed from G^* in a way similar to the one used to obtain the subgraphs G_1 and G_2 from G in Step 4 of Algorithm Recognize-Cograph. Thus, $\widetilde{G}$'s construction takes $O(\log n)$ time and requires $O((n+m)/\log n)$ processors on the EREW PRAM model. The computation of the degrees of the vertices $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_\ell$ in $\widetilde{G}$ can be done within the same time and processor bounds (see Step 1 of Algorithm Recognize-Cograph).

In order to compute the array $\delta[\,]$, we use an auxiliary array $d[\,]$ of size ℓ, which we initialize by assigning to the entry $d[i]$ the degree of $\hat{v}_i$ in $\widetilde{G}$, $1 \leq i \leq \ell$. Since the number of co-components is $O(\sqrt{m})$ according to Observation 2.1, the array $d[\,]$ can be sorted in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model. Then, it is easy to see that we can locate the distinct values of the sorted array $d[\,]$ using prefix sums and array packing techniques. Thus, the array $\delta[i]$, $1 \leq i \leq \ell'$, can be computed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM.

Step 7: Let $d[\,]$ be the sorted array of size ℓ computed in Step 6, and let $\pi = [\pi(1), \pi(2), \ldots, \pi(\ell)]$ be a permutation of the integers $1, 2, \ldots, \ell$ such that $d[\pi(i)] \leq d[\pi(j)]$ for every $1 \leq i < j \leq \ell$. In order to avoid concurrent read operations while computing the array $pos[\,]$, we use an auxiliary array $d'[\,]$ of size ℓ; we initialize it by setting $d'[i] = 1$ if $i = 1$ or $d[i] \neq d[i-1]$, and $d'[i] = 0$ otherwise, and we subsequently compute prefix sums on it. Then, $pos[i] \leftarrow d'[\pi(i)]$, for $i = 1, 2, \ldots, \ell$. Thus, the array $pos[\,]$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

Step 8: This step involves the construction of $O(\ell')$ nodes and $O(n)$ pointer assignments. Since $\ell' = O(\sqrt{m})$, it is easy to see that the execution of the step takes $O(\log n)$ time and requires $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 9: The only operations performed in Substeps 9(a) and 9(c) are construction of at most n nodes and pointer assignments (the degrees of the vertices v_i have been computed in Step 6). Thus, both substeps can be executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model.

Let us now analyze the time-processor complexity of Step 9(b). Here, $k = O(n)$ pointer assignments are performed on the root-nodes r_i, where r_i is the root-node of the cotree of the graph $G[\mathcal{C}_i]$, $1 \leq i \leq k$. In particular, the node r_i gets attached as a child of the tree node t_{p_i}, where p_i is such that v_i is adjacent to the co-component representative $\hat{v}_{p_i}$ in the graph $\widetilde{G}$, and it is not adjacent to any $\hat{v}_j$ with $j < p_i$. By using an auxiliary array $A_v[\,]$ for each vertex $v \in V(\widetilde{G})$ (of size equal to the degree of v in $\widetilde{G}$), and the array $pos[\,]$ computed in Step 7, we can compute the index p_i for each representative v_i $(1 \leq i \leq k)$ avoiding concurrent-read operations as follows:

▷ For each co-component representative $\hat{v}_i$, $1 \leq i \leq \ell$, do in parallel

9.1 copy the value $pos[i]$ to each entry of $A_{\hat{v}_i}[\,]$;

▷ For each component representative v_i, $1 \leq i \leq k$, do in parallel

9.2 for each vertex u adjacent to v_i in $\widetilde{G}$ do in parallel
{*u is a co-component representative*}
$A_{v_i}[r_{v_i}(u)] \leftarrow A_u[r_u(v_i)]$, where $r_x(y)$ denotes the rank of y in the adj. list of x in $\widetilde{G}$;

9.3 $p_i \leftarrow$ the minimum element of the array $A_{v_i}[\,]$;

It is easy to see that the above Substeps 9.1 through 9.3 can be completed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model. Thus, the entire Step 9 is completed within the same time and processor bounds.

If we take into consideration the time and processor complexity of each step of the algorithm Construct-Cotree and the recursive calls, and work in a fashion similar to the one used in the analysis of Algorithm Recognize-Cograph, we obtain the following result.

Theorem 4.1. *Algorithm Construct-Cotree runs in $O(\log^2 n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.*

Corollary 4.1. *Let G be a cograph on n vertices and m edges. The cotree of the graph G can be constructed in $O(\log^2 n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.*

6 Concluding Remarks

In this paper, we have presented parallel algorithms for recognizing cographs and for constructing the cotree of a cograph. When applied on a graph on n vertices and m edges, both algorithms run in $O(\log^2 n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model of computation. Thus, our results improve upon the previously known linear-processor parallel algorithms for the same problems [9, 11]. Instrumental in our work is an optimal parallel algorithm which computes the connected components of a graph or detects that it contains a P_4; this algorithm is interesting in its own right as it provides an optimal parallel connectivity algorithm for cographs and can be extended to yield an optimal connectivity algorithm for graphs with constant diameter.

An interesting open question is whether the class of cographs can be optimally recognized on the EREW PRAM model of computation, i.e., whether there exists an $O(\log n)$-time cograph recognition algorithm which runs on the EREW PRAM model and requires $O((n+m)/\log n)$ processors. Moreover, since cographs form a proper subclass of permutation graphs, a direction for further research would be to investigate whether a similar technique applies for the purpose of recognizing the class of permutation graphs within the same time-processor bounds.

More general classes of perfect graphs, such as the classes of P_4-reducible and P_4-sparse graphs, also admit unique tree representations up to isomorphism [13, 14]. Recently, Lin and Olariu presented parallel recognition and tree construction algorithms for P_4-sparse graphs [19]; for an input graph on n vertices and m edges, both the recognition and the tree construction algorithms run in $O(\log n)$ time using $O((n^2+nm)/\log n)$ processors on the EREW PRAM model of computation. Thus, it would be interesting to see whether the approach and algorithmic techniques used in this paper can help develop efficient parallel recognition and tree construction algorithms for these two classes of graphs.

References

[1] G.S. Adhar and S. Peng, Parallel algorithms for cographs and parity graphs with applications, *J. Algorithms* **11**, 252–284, 1990.

[2] G.S. Adhar and S. Peng, Parallel algorithms for path covering, Hamiltonian path, Hamiltonian cycle in cographs, *Proc. Inter. Conf. on Parallel Processing*, Vol. III, 364–365, 1990.

[3] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, 1997.

[4] H.L. Bodlaender and R.H. Möhring, The pathwidth and treewidth of cographs, *SIAM J. Discrete Math.* **6**, 181–188, 1993.

[5] A. Brandstädt, V.B. Le, and J.P. Spinrad, *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics and Applications, 1999.

[6] K.W. Chong, S.D. Nikolopoulos, and L. Palios, An optimal parallel co-connectivity algorithm, *Theory of Computing Systems* (to appear).

[7] D.G. Corneil, H. Lerchs, and L. Stewart-Burlingham, Complement reducible graphs, *Discrete Appl. Math.* **3**, 163–174, 1981.

[8] D.G. Corneil, Y. Perl, and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **14**, 926–934, 1985.

[9] E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance hereditary graphs, *Discrete Appl. Math.* **57**, 29–44, 1995.

[10] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., 1980.

[11] X. He, Parallel algorithm for cograph recognition with applications, *J. Algorithms* **15**, 284–313, 1993.

[12] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[13] B. Jamison and S. Olariu, Recognizing P_4-sparse graphs in linear time, *SIAM J. Comput.* **21**, 381–406, 1992.

[14] B. Jamison and S. Olariu, A linear-time recognition algorithm for P_4-reducible graphs, *Theoret. Comput. Sci.* **145**, 329–344, 1995.

[15] H.A. Jung, On a class of posets and the corresponding comparability graphs, *J. Combin. Theory Ser. B* **24**, 125–133, 1978.

[16] D.G. Kirkpatrick and T. Przytycka, Parallel recognition of complement reducible graphs and cotree construction, *Discrete Appl. Math.* **29**, 79–96, 1990.

[17] H. Lerchs, On cliques and kernels, *Technical Report*, Department of Computer Science, University of Toronto, March 1971.

[18] R. Lin and S. Olariu, An NC recognition algorithm for cographs, *J. Parallel and Distributed Comput.* **13**, 76–90, 1991.

[19] R. Lin and S. Olariu, A fast parallel algorithm to recognize P4-sparse graphs, *Discrete Appl. Math.* **81**, 191–215, 1998.

[20] R. Lin, S. Olariu, and G. Pruesse, An optimal path cover algorithm for cographs, *Computers Math. Applic.* **30**, 75–83, 1995.

[21] K. Nakano, S. Olariu, and A.Y. Zomaya, A time-optimal solution for the path cover problem on cographs, *Theoret. Comput. Sci.* **290**, 1541–1556, 2003.

[22] J. Reif (editor), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, Inc., 1993.

[23] D.P. Sumner, Dacey graphs, *J. Austral. Math. Soc.* **18**, 492–502, 1974.

[24] J. Valdes, R.E. Tarjan, and E.L. Lawler, The recognition of serial parallel digraphs, *SIAM J. Comput.* **11**, 298–313, 1982.