ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Discrete Applied Mathematics ■■■ (■■■■) ■■■–■■■

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Efficient parallel recognition of cographs

Stavros D. Nikolopoulos, Leonidas Palios

Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece

Received 24 October 2003; received in revised form 5 January 2005; accepted 14 February 2005

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 graph if it is a cograph; if the input graph is not a cograph, both algorithms return an induced P_4. For a graph on n vertices and m edges, both our cograph recognition and cotree construction algorithms run in $\mathrm{O}(\log^2 n)$ time and require $\mathrm{O}((n+m)/\log n)$ processors on the EREW PRAM model of computation. Our algorithms are motivated by the work of Dahlhaus (Discrete Appl. Math. 57 (1995) 29–44) and take advantage of the optimal $\mathrm{O}(\log n)$-time computation of the co-connected components of a general graph (Theory Comput. Systems 37 (2004) 527–546) and of an optimal $\mathrm{O}(\log n)$-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 (Discrete Appl. Math. 57 (1995) 29–44; J. Algorithms 15 (1993) 284–313): we achieve a better time-processor product using a weaker model of computation and we provide a certificate (an induced P_4) whenever our algorithms decide that the input graphs are not cographs.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Perfect graphs; Cographs; Cotree; Connected and co-connected components; Parallel algorithms; Parallel recognition; Certifying algorithms

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 complementation. More precisely, the class of cographs is defined recursively as follows:

E-mail addresses: stavros@cs.uoi.gr (S.D. Nikolopoulos), palios@cs.uoi.gr (L. Palios).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.02.004

(i) a single-vertex graph is a cograph, (ii) the disjoint union of cographs is a cograph and (iii) the complement of a cograph is a cograph.

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 [16], P_4 restricted graphs [8,9], 2-parity graphs and Hereditary Dacey graphs or HD-graphs [24]. 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 [5,11]. Furthermore, cographs are precisely the graphs which contain no induced subgraph isomorphic to a P_4 (chordless path on four vertices).

Cographs were introduced in the early 1970s by Lerchs [18] 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 labelled 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- 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, linear-time algorithms for recognizing cographs were given in [9,6]. 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 [17], which requires $O(\log^2 n)$ time with $O(n^3/\log^2 n)$ processors on the CREW PRAM model. Lin and Olariu [19] 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 [10] 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 [12]; it requires $O(\log^2 n)$ time and $O(n + m)$ processors on the 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 [5,11]. In particular, for the problem of determining the minimum path cover for a cograph, Lin et al. [21] 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 [1,2,17]; for example, the minimum path cover problem is solved in $O(\log n)$ time with $O(n/\log n)$ processors [22].

The cotree of a cograph is constructed in $O(\log^2 n)$ time with $O(n+m)$ processors [10,12], or in $O(\log n)$ time with $O((n^2+nm)/\log n)$ processors [19], 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 [18]), which enable us to obtain efficient parallel algorithms for recognizing whether a given graph is a cograph and for constructing the cotree of a graph if it is 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 [10] and take advantage of the optimal parallel algorithm for computing the connected components of the complement of a graph described in [7] 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). Finally, we note that all our algorithms produce an induced P_4 whenever they decide that the input graph is not a cograph, thus providing a certificate for their decision.

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 $G[S]$.

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 $N(S) := (\bigcup_{x \in S} N(x)) - S$ and its closed neighborhood as $N[S] := N(S) \cup S$. The *degree* of a vertex x in G, denoted *deg*(x), is the number of vertices adjacent to x in G; thus, *deg*$(x) = |N(x)|$. 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 $y \in V_j$.

A *path* in the graph G is a sequence of vertices $v_1 v_2 \ldots v_k$ such that $v_i v_{i+1} \in E(G)$ 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 $v_1 v_2 \ldots v_k$ is *chordless* if $v_i v_j \notin E(G)$ 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. In a P_4 *abcd*, the vertices b and c are the *midpoints* and the vertices a and d the *endpoints* of the P_4. The edge connecting the midpoints of a P_4 is its *rib*, whereas the other two edges (which are incident on the endpoints) are the *wings*; for example, the edge *bc* is the rib and the edges *ab* and *cd* are the wings of the P_4 *abcd*.

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 $\bar{G}$ of G.

An important tool in both our cograph recognition and cotree construction algorithms is to consider for a vertex v of a graph G the partition of the subgraphs $G[N(v)]$ and $G[V(G) - N[v]]$ into co-components and connected components, respectively. In particular, we define:

Definition 2.1. Let G be a graph and v a vertex of G. We define the *component-partition of G with respect to v*, denoted by $(v; \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell; \mathscr{C}_1, \mathscr{C}_2 \ldots, \mathscr{C}_k)$, as the partition of the vertex set $V(G)$

$$V(G) = \{v\} + \hat{\mathscr{C}}_1 + \hat{\mathscr{C}}_2 + \cdots + \hat{\mathscr{C}}_\ell + \mathscr{C}_1 + \mathscr{C}_2 + \cdots + \mathscr{C}_k,$$

where $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ are the co-connected components of $G[N(v)]$ and $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ are the connected components of $G[V(G) - N[v]]$.

Since the cographs do not contain P_4s, we are especially interested in component-partitions such that there is no P_4 with vertices in both $N[v]$ and $V(G) - N[v]$, which are the only type of P_4s with not all its vertices in the same co-component $\hat{\mathscr{C}}_i$ or in the same component $\mathscr{C}_j$; note that any P_4 with all its vertices in $N[v]$ has all its vertices in the same co-component of $G[N(v)]$, and any P_4 with all its vertices in $V(G) - N[v]$ has all its vertices in the same component of $G[V(G) - N[v]]$.

Definition 2.2. Let G be a graph, v a vertex of G, and $(v; \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell; \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k)$ the component-partition of G with respect to v. We say that this component-partition is *good* if and only if G contains no P_4 with vertices in both $N[v]$ and $V(G) - N[v]$.

Clearly, if the component-partition $(v; \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell; \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k)$ of a graph G with respect to a vertex v is good and if the graph G contains a P_4 as an induced subgraph, then this P_4 entirely belongs either to one of the co-components $\hat{\mathscr{C}}_i$ $(1 \leqslant i \leqslant \ell)$ of the subgraph $G[N(v)]$ or to one of the components $\mathscr{C}_j$ $(1 \leqslant j \leqslant k)$ of the subgraph $G[V(G) - N[v]]$; recall that no P_4 with its vertices in $N(v)$ has vertices belonging to two or more co-components of $G[N(v)]$, and no P_4 with its vertices in $V(G) - N[v]$ has vertices belonging to two or more components of $G[V(G) - N[v]]$.

In Lemma 2.1 we establish necessary and sufficient conditions for a component-partition to be good.

Lemma 2.1. *Let G be a graph, v a vertex of G, and $(v; \hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell; \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k)$ the component-partition of G with respect to v. Then, the component-partition of G with respect to v is good if and only if the following two conditions hold*:

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

(ii) *if, for each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, we define the set $\hat{I}_i = \{j \mid \hat{\mathscr{C}}_i \text{ sees } \mathscr{C}_j\}$, then the co-components of $G[N(v)]$ have the following monotonicity property*: $|\hat{I}_i| \leqslant |\hat{I}_j|$ *implies that $\hat{I}_i \subseteq \hat{I}_j$.*

Proof. ($\Rightarrow$) We assume that the component-partition of G with respect to v is good, i.e., the graph G does not contain a P_4 with vertices in both $N[v]$ and $V(G) - N[v]$; we will show that conditions (i) and (ii) hold. If condition (i) did not hold, then there would be a vertex x of some $\hat{\mathscr{C}}_i$ which would be adjacent to a vertex y in some $\mathscr{C}_j$ but non-adjacent to a vertex z of $\mathscr{C}_j$; 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 co-components $\hat{\mathscr{C}}_i$ and $\hat{\mathscr{C}}_j$ such that $|\hat{I}_i| \leqslant |\hat{I}_j|$ and $\hat{I}_i \nsubseteq \hat{I}_j$. Then, there exists $t \in \hat{I}_i - \hat{I}_j$, which implies that $\hat{\mathscr{C}}_i$ sees $\mathscr{C}_t$ whereas $\hat{\mathscr{C}}_j$ misses $\mathscr{C}_t$. Additionally, since $|\hat{I}_i| \leqslant |\hat{I}_j|$ and $t \in \hat{I}_i - \hat{I}_j$, there exists $t' \in \hat{I}_j - \hat{I}_i$, which in turn implies that $\hat{\mathscr{C}}_j$ sees $\mathscr{C}_{t'}$ whereas $\hat{\mathscr{C}}_i$ misses $\mathscr{C}_{t'}$. But then, any four vertices a, b, c, d, such that $a \in \mathscr{C}_t$, $b \in \hat{\mathscr{C}}_i$, $c \in \hat{\mathscr{C}}_j$, and $d \in \mathscr{C}_{t'}$, induce a P_4 $abcd$ in G; a contradiction.

($\Leftarrow$) 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

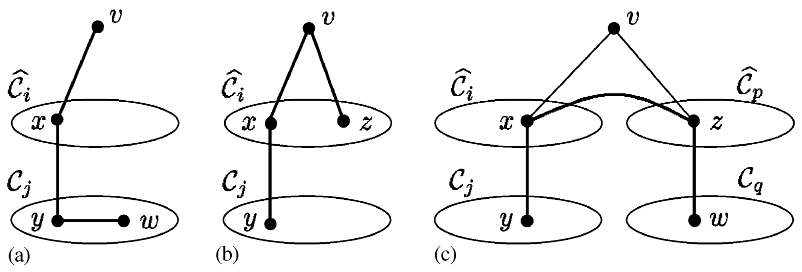

Fig. 1.

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 $vxyw$ with $x \in N(v)$ and $y, w \in V(G) - N[v]$ (see Fig. 1(a)), or of the form $zvxy$ with $x, z \in N(v)$ and $y \in V(G) - N[v]$ (see Fig. 1(b)). In the former case, y, w belong to the same connected component of $G[V(G) - N[v]]$ and x sees exactly one of them, while, in the latter, x, z belong to the same co-component of $G[N(v)]$ and y 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, the path $avxy$ is a P_4, which as in case (a) contradicts the fact that condition (i) holds.
- the P_4 is $wzxy$ where $z \in N(v)$ and $w \in V(G) - N[v]$: Since condition (i) holds, it must be the case that the vertices x, z belong to different co-components of $G[N(v)]$ and the vertices y, w belong to different components of $G[V(G) - N[v]]$ (see Fig. 1(c)). Let $x \in \hat{\mathscr{C}}_i$, $z \in \hat{\mathscr{C}}_p$, where $i \neq p$, and suppose without loss of generality that $|\hat{I}_i| < |\hat{I}_p|$. Then, condition (ii) implies that $\hat{I}_i \subseteq \hat{I}_p$. Moreover, if $y \in \mathscr{C}_j$, from condition (i) we have that $j \in \hat{I}_i$. Since $\hat{I}_i \subseteq \hat{I}_p$, we get that $j \in \hat{I}_p$, which contradicts the fact that the vertices y and z are not adjacent (see P_4 $wzxy$).

In all cases, we reached a contradiction; therefore, the graph G cannot contain a P_4 with vertices in both $N[v]$ and $V(G) - N[v]$, that is, the component-partition of G with respect to v is good. $\square$

From the proof of Lemma 2.1, we see that if G contains any P_4 with vertices in both $N[v]$ and $V(G) - N[v]$ which is of a general form other than those shown in Fig. 1, then G contains a P_4 of the form of Fig. 1(a) or Fig. 1(b). Thus, condition (ii) of Lemma 2.1 guarantees that no P_4 of the form of Fig. 1(c) exists, while condition (i) guarantees that no other P_4 exists with vertices in both $N[v]$ and $V(G) - N[v]$. In fact, condition (ii) can be phrased in another equivalent way, as given in the following corollary.

Corollary 2.1. *Let G be a graph, v a vertex of G, and $(v; \hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2, \ldots, \hat{\mathcal{C}}_\ell; \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$ the component-partition of G with respect to v. Then, the component-partition of G with respect to v is good if and only if the following two conditions hold*:

(i) *Every co-component $\hat{\mathcal{C}}_i$ either sees or misses every component $\mathcal{C}_j$;*

(ii) *Suppose that the ordering of the co-components $\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2, \ldots, \hat{\mathcal{C}}_\ell$ corresponds to their ordering by non-decreasing $|\hat{I}_i|$, where $\hat{I}_i = \{j \mid \hat{\mathcal{C}}_i \text{ sees } \mathcal{C}_j\}$. If we associate each component $\mathcal{C}_i$, $1 \leqslant i \leqslant k$, with the set $I_i = \{j \mid \mathcal{C}_i \text{ sees } \hat{\mathcal{C}}_j\}$, 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. It suffices to show that condition (ii) of Lemma 2.1 and condition (ii) of Corollary 2.1 are equivalent.

($\Rightarrow$) Suppose that condition (ii) of Lemma 2.1 holds; we will show that condition (ii) of Corollary 2.1 holds. For any component $\mathcal{C}_i$ such that $I_i \neq \emptyset$, it suffices to show that if $h \in I_i$ then $\forall j > h, \mathcal{C}_i$ sees $\hat{\mathcal{C}}_j$. Consider any such j; since $h < j$, it holds that $|\hat{I}_h| \leqslant |\hat{I}_j|$, which according to condition (ii) of Lemma 2.1 yields that $\hat{I}_h \subseteq \hat{I}_j$. Since $h \in I_i$, we have that $\mathcal{C}_i$ sees $\hat{\mathcal{C}}_h$, or equivalently that $\hat{\mathcal{C}}_h$ sees $\mathcal{C}_i$; that is, $i \in \hat{I}_h$. Since $\hat{I}_h \subseteq \hat{I}_j$, then $i \in \hat{I}_j$, i.e., $\mathcal{C}_i$ sees $\hat{\mathcal{C}}_j$.

($\Leftarrow$) Suppose that condition (ii) of Corollary 2.1 holds; we will show that condition (ii) of Lemma 2.1 holds. Let us consider two co-components $\hat{\mathcal{C}}_p$ and $\hat{\mathcal{C}}_q$, and suppose without loss of generality that $|\hat{I}_p| \leqslant |\hat{I}_q|$. We need to show that $\hat{I}_p \subseteq \hat{I}_q$. Let $t \in \hat{I}_p$; this is equivalent to the fact that the component $\mathcal{C}_t$ sees $\hat{\mathcal{C}}_p$. But then, $p \in I_t$ and in fact $q \in I_t$, since the inequality $|\hat{I}_p| \leqslant |\hat{I}_q|$ implies that $p < q$ in the ordering of the co-components of $G[N(v)]$ by non-decreasing $|\hat{I}_i|$. Therefore, $t \in \hat{I}_q$. Since this holds for any $t \in \hat{I}_p$, we have that $\hat{I}_p \subseteq \hat{I}_q$, as desired. $\square$

Consider the partition of the set of co-components $\{\hat{\mathcal{C}}_1, \hat{\mathcal{C}}_2, \ldots, \hat{\mathcal{C}}_\ell\}$ of the subgraph $G[N(v)]$ into a collection of sets where any two co-components $\hat{\mathcal{C}}_i, \hat{\mathcal{C}}_j$ belong to the same set if and only if $\hat{I}_i = \hat{I}_j$, i.e., $\hat{\mathcal{C}}_i$ and $\hat{\mathcal{C}}_j$ see the same components of the subgraph $G[V(G) - N[v]]$. Let us denote these partition sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$, where, for every i, j such that $1 \leqslant i < j \leqslant \ell'$, and every $\hat{\mathcal{C}}_r \in \hat{S}_i$ and $\hat{\mathcal{C}}_s \in \hat{S}_j$, it holds that $\hat{I}_r \subset \hat{I}_s$; the value ℓ' is equal to the number of distinct values of the $\hat{I}_i$s, and thus each set $\hat{S}_j$ is non-empty. It is not difficult to see that the partition sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ have the following properties:

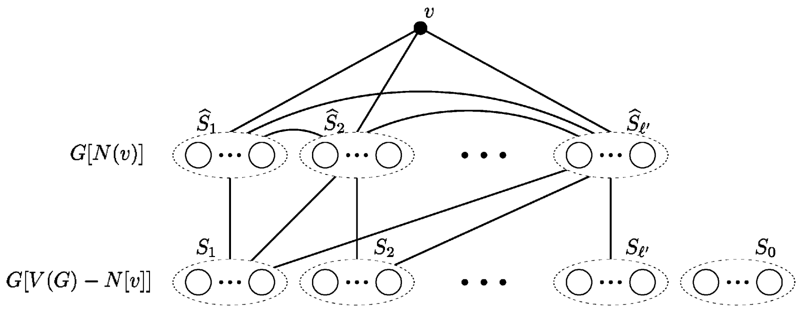

Fig. 2.

Observation 2.1. *Let G be a graph, v a vertex of G, and $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ be the partition of the set of co-components $\{\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell\}$ of the subgraph $G[N(v)]$ as described above. Moreover, suppose that condition* (i) *of Lemma* 2.1 *holds. The definition of the partition sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ easily implies the following*:

- *If a connected component $\mathscr{C}$ of the subgraph $G[V(G) - N[v]]$ sees a co-component $\hat{\mathscr{C}}_i \in \hat{S}_j$, then $\mathscr{C}$ sees all the co-components in $\hat{S}_j$.*
- *Let us consider the ordering of the co-components $\{\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell\}$ consisting of an arbitrary ordering of the elements of the set $\hat{S}_1$ followed by an arbitrary ordering of the elements of $\hat{S}_2$ and so on up to the set $\hat{S}_{\ell'}$. In this ordering, the co-components $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, are ordered by non-decreasing value of $|\hat{I}_i|$.*

In light of the above observations and due to condition (ii) of Corollary 2.1, in a good component-partition of a graph G with respect to v, we can partition the set of connected components $\{\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k\}$ of the subgraph $G[V(G) - N[v]]$ into sets $S_0, S_1, \ldots, S_{\ell'}$ as follows:

$$S_1 = \{\mathscr{C}_j \mid \forall \hat{\mathscr{C}} \in \hat{S}_1,\ \mathscr{C}_j \text{ sees } \hat{\mathscr{C}}\},$$
$$S_i = \{\mathscr{C}_j \mid \forall \hat{\mathscr{C}} \in \hat{S}_i \text{ and } \hat{\mathscr{C}}' \in \hat{S}_{i-1}, \mathscr{C}_j \text{ sees } \hat{\mathscr{C}} \text{ but does not see } \hat{\mathscr{C}}'\} \quad (2 \leqslant i \leqslant \ell'),$$
$$S_0 = \{\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k\} - \bigcup_{i=1,\ldots,\ell'} S_i.$$

The definition of the sets $\hat{S}_j$, $j = 1, 2, \ldots, \ell'$, 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; in fact, S_0 contains the connected components of G except for the component to which v belongs. Fig. 2 illustrates the partitions of the set of co-components and of the set of components described above and their adjacencies in a good component-partition of the graph G with respect to vertex v; 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 the partitions into sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ and $S_0, S_1, \ldots, S_{\ell'}$, the cotree of a cograph G has a very special structure, which is described in the following lemma (clearly, the component-partition of a cograph with respect to any of its vertices is good so that the

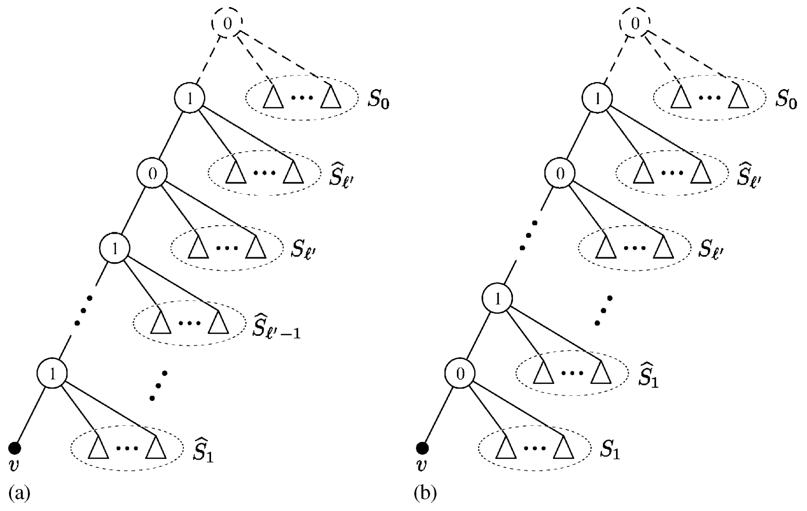

Fig. 3.

conditions (i) and (ii) of Lemma 2.1 and Corollary 2.1 hold and the sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ and $S_0, S_1, \ldots, S_{\ell'}$ are well defined).

Lemma 2.2. *Let G be a cograph, v a vertex of G, and $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_{\ell'}$ 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 Fig. 3*(a);
(ii) *if $S_1 \neq \emptyset$, the cotree of G has the general form depicted in Fig. 3*(b).

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

The circular nodes labelled with a 0 or a 1 in Fig. 3 are 0- 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 2.2 gives us a way of constructing the cotree of an input cograph G: we compute the partitions $\hat{S}_1, \ldots, \hat{S}_{\ell'}$ and $S_0, S_1, \ldots, S_{\ell'}$; we recursively construct the cotrees of the elements of each of the above partition sets; we link these cotrees as indicated in Fig. 3. 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.

The good selection of the vertex v based on which we compute the co-components of the subgraph $G[N(v)]$ and the components of the subgraph $G[V(G) - N[v]]$ is crucial both for the cograph recognition and the cotree construction. We will follow the selection principle used by Dahlhaus [10], although we will be more concrete in our choices. If the number of vertices of the graph G is n, we define the sets L, M, and H of the low-, middle-, and high-degree vertices of G, respectively, as follows:

$$\begin{aligned} L &= \left\{x \in V(G) \mid \text{degree of } x \text{ in } G < \tfrac{1}{4}n\right\}, \\ M &= \left\{x \in V(G) \mid \tfrac{1}{4}n \leqslant \text{degree of } x \text{ in } G \leqslant \tfrac{3}{4}n\right\}, \\ H &= \left\{x \in V(G) \mid \text{degree of } x \text{ in } G > \tfrac{3}{4}n\right\}. \end{aligned}$$

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 $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, of the subgraph $G[N(v)]$ and of each connected component $\mathscr{C}_j$, $1 \leqslant j \leqslant k$, of the subgraph $G[V(G) - N[v]]$ does not exceed $\frac{3}{4}n$.*

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

Lemma 2.3. *Let G be a graph on n vertices, the set L as defined above, F a connected subgraph of G such that every vertex of F belongs to L. Then, if the number q of vertices of F is at least $\frac{1}{2}n$, the subgraph F is not a cograph and in particular its component-partition with respect to any of its vertices is not good.*

Proof. Let v be an arbitrary vertex of F, and suppose for contradiction that the component-partition of F with respect to v is good, that is, F contains no P_4 with vertices in both $N[v]$ and $V(F) - N[v]$. Then, from Corollary 2.1, conditions (i) and (ii) hold. Assuming that the ordering of the co-components $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ of $F[N(v)]$ corresponds to their ordering by non-decreasing $|\hat{I}_i|$ (see Corollary 2.1), let us consider any vertex x in $\hat{\mathscr{C}}_\ell$. Then, x sees v and all the vertices in $V(F) - N[v]$; since $|V(F) - N[v]| = q - (1 + deg(v))$ where $deg(v)$ is the degree of v in F, the degree $deg(x)$ of x in F is $deg(x) \geqslant 1 + q - (1 + deg(v)) = q - deg(v)$. If we solve for q, we get: $q \leqslant deg(x) + deg(v)$. Since all the vertices of F belong to L, their degrees are less than $\frac{1}{4}n$, and thus we have that $q < \frac{2}{4}n$; a contradiction. $\square$

Lemma 2.3 can be used to prove Lemma 6 of [10] in a different way. More importantly, however, for a subgraph F as described in Lemma 2.3 which has at least $\frac{n}{2}$ vertices, it gives us the location of a P_4; this proves very useful in our certificate producing step. Moreover, thanks to Lemma 2.3, we establish in Lemma 2.4 an extension of a result given in [10]; Lemma 2.4 has a simpler proof than the proof in [10] and also gives us a way of locating a P_4 whenever the graph G is not a cograph.

Lemma 2.4. *Let G be a graph on n vertices such that the set M is empty. Let v be the vertex in the set L which has the maximum number of neighbors in the set H, and let $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ be the connected components of $G[V(G) - N[v]]$. If there exists a component $\mathscr{C}_i$ such that*

$|\mathscr{C}_i| > \frac{3}{4}n$ and the cardinality of a co-component $\hat{\mathscr{A}}_{i,j}$ of the graph $G[\mathscr{C}_i]$ is at least $\frac{1}{2}n$, then G is not a cograph and the component-partition of $G[\mathscr{C}_i]$ with respect to any of its vertices or the component-partition of $G[\hat{\mathscr{A}}_{i,j}]$ with respect to any of its vertices is not good.

Proof. Observe that every vertex $x \in H - \mathscr{C}_i$ is adjacent to at least one vertex of $\mathscr{C}_i$; if not, then the degree of x would be at most equal to $n - |\mathscr{C}_i| < \frac{1}{4}n$, which contradicts the definition of the set H. But then, from Lemma 2.1 condition (i), such a vertex x sees the entire $\mathscr{C}_i$; this follows from the fact that x belongs to a co-component of $G[N(v)]$, since x is adjacent to a vertex in $\mathscr{C}_i$ and it does not belong to $\mathscr{C}_i$. If $\mathscr{C}_i$ contains no high-degree vertex, it would be a connected subgraph of G whose vertices all belong to L and then, according to Lemma 2.3, $G[\mathscr{C}_i]$ is not a cograph and more specifically the component-partition of $G[\mathscr{C}_i]$ with respect to any of its vertices is not good.

Suppose now that $\mathscr{C}_i$ contains at least one high-degree vertex. We show that $\mathscr{C}_i$ contains no low-degree vertices. Suppose that there existed such a vertex z. Since $\mathscr{C}_i$ is connected and contains a high-degree vertex, there would exist a path from z to that high-degree vertex in $G[\mathscr{C}_i]$; since $M = \emptyset$, such a path would contain an edge connecting a low-degree vertex, say, w, to a high-degree vertex. Then, w is adjacent to at least one high-degree vertex in $\mathscr{C}_i$ and to all the high-degree vertices in $H - \mathscr{C}_i$ because every vertex in $H - \mathscr{C}_i$ sees the entire $\mathscr{C}_i$. Since$H \cap N(v) \subseteq H - \mathscr{C}_i$, this contradicts the choice of v as the low-degree vertex that has the maximum number of neighbors in H. Therefore, $\mathscr{C}_i$ contains only high-degree vertices. Then, in the complement of G, the vertices of $\mathscr{C}_i$ belong to the low-degree vertex set L' of $\bar{G}$ and the co-components of $G[\mathscr{C}_i]$ would be subsets of the connected components of $\bar{G}[L']$; Lemma 2.3 implies that if the cardinality of such a co-component $\hat{\mathscr{A}}_{i,j}$ is at least $\frac{1}{2}n$, the subgraph $G[\hat{\mathscr{A}}_{i,j}]$ is not a cograph and its component-partition with respect to any of its vertices is not good. □

Finally, 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 2.3. *Let G be a graph on n vertices and m edges, v a vertex of G, and $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ the co-connected components of $G[N(v)]$. Then, $\ell < \sqrt{2m}$.*

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

$$1/2 \sum_i \left(|\hat{\mathscr{C}}_i| \cdot \sum_{j \neq i} |\hat{\mathscr{C}}_j| \right) \geqslant 1/2 \sum_i (|\hat{\mathscr{C}}_i| \cdot (\ell - 1)) \geqslant \ell(\ell-1)/2$$

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 $m \geqslant \ell + \ell(\ell-1)/2 > \ell^2/2$, from which the observation follows. □

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; in Section 3.1, we also show how to augment the algorithm to return a P_4 whenever it detects such a subgraph in the input graph.

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, 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 if and only if $f(f(x)) = f(f(y))$. 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*; *in particular, if $f(f(x)) < f(f(y))$ then G contains a P_4 abxy whereas if $f(f(x)) > f(f(y))$ then G contains a P_4 abyx.*

(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 $\mathscr{C}_i$ of G, $f(f(x))=\min\{u \mid u \in \mathscr{C}_i\}$; 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. Throughout the section, we assume that the vertex set $V(G)$ of the input graph G equals the set $\{1, 2, \dots, n\}$, where n is the number of vertices of G.

```
Algorithm Components-or-P4
Input:   an undirected graph $G$.
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 mark the edge $uv$     {$G$ contains a $P_4$ with wing $uv$}
    if there exists a marked edge of $G$
    then print that $G$ contains a $P_4$ as an induced subgraph; return.
4.  For each vertex $v \in V(G)$ do in parallel
        comp$[v] \leftarrow B[v]$;
    return the array comp[].
```

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 [3,13,23]. We assume that the input graph G is given in adjacency-list representation, i.e., for each vertex v, we have a linked list *List*(v) of the neighbors of v in G.

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. $A[v] \leftarrow \min\{A_v[i] \mid 1 \leqslant i \leqslant deg(v)\}$.
```

Clearly, by taking advantage of the "twin" entries $W[v,u]$ and $W[u,v]$ in Step 2.3, we ensure that $A[v]$ is correctly updated. In Step 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. Steps 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 mark all the edges 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 $v \in V(G)$, two auxiliary arrays $B_v[]$ and $C_v[]$, each 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 $B_v[r_v(u)] \leftarrow 0$;
              $C_v[r_v(u)] \leftarrow u$;
  3.3. let $B_v[\hat{\imath}]$ be equal to $\min\{B_v[i] \mid 1 \leqslant i \leqslant deg(v)\}$;
       if $B_v[\hat{\imath}] = 0$
       then mark the edge $vw$, where $w = C_v[\hat{\imath}]$.
```

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 $\mathrm{O}(\log n)$ time with $\mathrm{O}(n/\log n)$ 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* $\mathrm{O}(\log n)$ *time using* $\mathrm{O}((n+m)/\log n)$ *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 $\mathrm{O}(d \log n) = \mathrm{O}(\log n)$ time using $\mathrm{O}((n+m)/\log n)$ processors on the EREW PRAM.

Corollary 3.1. *Let G be a graph on n vertices and m edges, which has constant diameter. Then, the connected components of G can be computed in* $\mathrm{O}(\log n)$ *time using* $\mathrm{O}((n+m)/\log n)$ *processors on the EREW PRAM model.*

Remark 3.1. *Computing the representatives of the connected components*. Let G be a graph on n vertices and let $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ 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 $v \in \mathscr{C}_i$, `comp`$[v]$ is equal to the representative of the connected component $\mathscr{C}_i$; in fact, the representatives $v_1, v_2, \ldots, v_k$ of the connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ are such that $v_i = \min\{v \in \mathscr{C}_i\}, 1 \leqslant i \leqslant k$. The representatives can be isolated in $\mathrm{O}(\log n)$ time using $\mathrm{O}(n/\log n)$ 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 $1 \leqslant i \leqslant k$.

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 $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ 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 $\mathscr{C}_i$, $1 \leqslant i \leqslant k$, 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 $v_i$, $1 \leqslant i \leqslant k$, 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 $1 \leqslant i \leqslant k$, the resulting packed array associated with vertex v_i contains each of the vertices in $\mathscr{C}_i - \{v_i\}$ exactly once; adding an entry for v_i yields the entire set of vertices of the connected component $\mathscr{C}_i$. 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 $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ of the graph G, we can also compute the adjacency-list representation of each induced subgraph

$G[\mathscr{C}_1], G[\mathscr{C}_2], \ldots, G[\mathscr{C}_k]$ within the same time and processor bounds using the same model of computation.

3.1. Finding a P_4

The algorithm Components-or-P4 can be easily augmented so that it finds and prints a P_4 of the input graph G whenever it decides that G contains a P_4. To do that, we replace Step 3 of the algorithm by

```
3.    For each edge $uv \in E(G)$ do in parallel
          if $B[u] > B[v]$
          then mark the edge $uv$ with the vertex-pair $(u, v)$;
          else if $B[u] < B[v]$
              then mark the edge $uv$ with the vertex-pair $(v, u)$;
      if there exists an edge which is marked with a pair and let $(x, y)$ be this pair
      then call Subroutine Find-$P4(G, (x, y))$; return;
```

where Subroutine Find-P4$(G, (x, y))$ finds and prints a P_4 *xypq* of G; its description is given below. The correctness of the augmented Step 3 follows from Lemma 3.1, statement (ii), and from the correctness of the subroutine Find-P4. From a complexity point of view, the augmented Step 3 is nearly identical to the original Step 3; since a call of the subroutine Find-P4 on a graph on n vertices and m edges takes $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM, the augmented algorithm Components-or-P4 has the same time and processor complexity.

The subroutine Find-P4 works very similarly to the algorithm Components-or-P4; it involves the following steps.

Subroutine Find-$P4$
Input: a graph G and a pair of vertices (x, y) such that G contains a P_4 of the form *xypq*.
Output: a P_4 of G with wing xy.

```
1.   Compute the subgraph $H$ of $G$ by deleting the edges $xz$ for all $z \in N(x) - \{y\}$.
2.   For each vertex $v \in V(G)$ do in parallel
         $A[v] \leftarrow 1$;
     Assign the value 0 to $A[x]$, that is, $A[x] \leftarrow 0$.
3.   For each vertex $v \in V(G)$ do in parallel
         $A[v] \leftarrow \min\{A[u] \mid u \in N[v]\}$;
         $B[v] \leftarrow \min\{A[u] \mid u \in N[v]\}$.
4.   For each edge $uv \in E(G)$ do in parallel
         if $B[u] > B[v]$
             then mark the edge $uv$ with the vertex-pair $(u, v)$    $\{P_4\ xyvu\}$
         else if $B[u] < B[v]$
                 then mark the edge $uv$ with the vertex-pair $(v, u)$    $\{P_4\ xyuv\}$
     if there exists an edge which is marked with a pair and let $(a, b)$ be this pair
     then print the $P_4$ xyba.
```

Given a graph G and a pair of vertices (x, y) such that G contains a P_4 $xypq$, the subroutine Find-P4 removes all the edges incident on x in G except for the edge xy (Step 1), and works on the resulting subgraph H. Because of this, if a vertex w of G, other than x and y, ends up with $B[w] = 0$ at the end of Step 3, then w is adjacent to y and non-adjacent to x in G; moreover, if a vertex w' ends up with $B[w'] \neq 0$, then $B[w'] = 1$ and w' is adjacent neither to x nor to y. Thus, since G contains a P_4 of the form $xypq$, a P_4 is guaranteed to be found. Then, the correctness of the subroutine Find-P4 follows from Lemma 3.1, statement (ii).

It is important to note that it is necessary for the subroutine Find-P4 to work on the subgraph H which results from the input graph G after the removal of the edges incident on x except for xy: if the sought P_4 participates in a chordless cycle on 5 vertices or is the top of a “house” (a simple cycle on 5 vertices with exactly one chord), then applying Steps 2–4 of subroutine Find-P4 on the entire graph G would not produce any P_4.

Steps 2–4 are very similar to Steps 1–3 of the augmented algorithm Components-or-P4 and can all be executed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM, where n and m are the numbers of vertices and edges of the input graph G. Step 1 can be executed by computing the subgraph of G induced by the vertices in $V(G) - \{x\}$ and then by adding x and making it adjacent only to y; the former can be easily done in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM by removing from the adjacency-list representation of G the adjacency list of x and any records storing x; the latter can be done in constant sequential time. Therefore, we obtain the following result.

Theorem 3.2. *Subroutine Find-$P4$ runs in* $O(\log n)$ *time using* $O((n+m)/\log n)$ *processors on the EREW PRAM model*.

4. Checking whether a component-partition is good

In this section we present a parallel algorithm which takes as input a graph G and a vertex $v \in V(G)$ and checks whether the component-partition of G with respect to v is good (see Definition 2.2); if so, the algorithm returns an appropriate message, otherwise it returns a P_4 using Subroutine Find-P4. The input graph G is assumed to be 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. We give next the detailed description of the algorithm.

```
Algorithm Good-Partition-or-P4
Input:  a graph $G$ and a vertex $v \in V(G)$.
Output: a message that the component-partition of $G$ with respect to $v$ is good, or an
        induced $P_4$ with vertices in both $N[v]$ and $V(G) - N[v]$.

1.  Compute the following induced subgraphs $G_1$ and $G_2$ of the graph $G$:
        $G_1 = G[N(v)]$;
        $G_2 = G[V(G) - N[v]]$.
2.  Compute the co-components $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots \hat{\mathscr{C}}_\ell$ of the graph $G_1$.
```

```
3.  Use Algorithm Components-or-$P4$ on the graph $G_2$ in order either to compute its
    connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ or to detect and return a $P_4$ using Subrou-
    tine Find-$P_4$;
    if a $P_4$ is returned then stop and return this $P_4$.
4.  For each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, of $G_1$ do in parallel
        check if there exist two non-adjacent vertices $x, y \in \hat{\mathscr{C}}_i$ such that $\exists z \in V(G) -$
        $N[v]$ which is adjacent to $y$ and is not adjacent to $x$;
        if there exists such a vertex $x$
        then mark vertex $x$     $\{G\ contains\ the\ P_4\ xvyz\}$
    if there exists a marked vertex $x$
    then call Subroutine Find-$P4$ on the graph $G$ and the vertex-pair $(x, v)$;
          stop and return the $P_4$ that Subroutine Find-P4 returned.
5.  For each connected component $\mathscr{C}_i$, $1 \leqslant i \leqslant k$, of $G_2$ do in parallel
        check if there exist two adjacent vertices $x, y \in \mathscr{C}_i$ such that $\exists z \in N(v)$
        which is adjacent to $y$ and is not adjacent to $x$;
        if there exist such vertices $x, y$
        then mark the vertex-pair $(x, y)$    $\{G\ contains\ the\ P_4\ xyzv\}$
    if there exists a marked vertex-pair $(x, y)$
    then call Subroutine $Find\text{-}P4$ on the graph $G$ and the vertex-pair $(x, y)$;
        stop and return the $P_4$ that Subroutine Find-P4 returned.
6.  Sort the co-components $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ of the graph $G_1$ in non-decreasing
    number of the connected components of the graph $G_2$ that each co-component
    sees;
    let $\hat{S} = (\hat{\mathscr{C}}_{\pi(1)}, \hat{\mathscr{C}}_{\pi(2)}, \ldots, \hat{\mathscr{C}}_{\pi(\ell)})$ be the sorted list.
7.  If there exist two consecutive co-components $\hat{\mathscr{C}}_{\pi(i)}$ and $\hat{\mathscr{C}}_{\pi(i+1)}$ in $\hat{S}$, where
    $1 \leqslant i \leqslant \ell$, such that $\hat{\mathscr{C}}_{\pi(i)}$ sees a connected component $\mathscr{C}$ of the graph $G_2$ which
    $\hat{\mathscr{C}}_{\pi(i+1)}$ misses
    then $\{G\ contains\ an\ induced\ P_4\ as\ shown\ in\ Fig.\ 1(c)\}$
          select a vertex $x$ from $\mathscr{C}$ and a vertex $y$ from $\hat{\mathscr{C}}_{\pi(i)}$;
          call Subroutine Find-$P4$ on the graph $G$ and the vertex-pair $(x, y)$;
          stop and return the $P_4$ that Subroutine Find-P4 returned.
8.  Return the message that the component-partition of $G$ with respect to $v$ is good.
```

In Steps 1–3, the algorithm constructs the component-partition. Then, it checks whether condition (i) and condition (ii) of Corollary 2.1 hold in Steps 4–5 and Steps 6–7, respectively.

Correctness: For the correctness of Step 4, we note that if a co-component $\hat{\mathscr{C}}_i$ of $G_1 = G[N(v)]$ contains two vertices a, b which do not have the same neighbors in $V(G) - N[v]$, then it contains two non-adjacent such vertices; it suffices to consider the pairs of consecutive vertices along a path in $\bar{G}[\hat{\mathscr{C}}_i]$ from a to b. Similarly, for the correctness of Step 5, if a component $\mathscr{C}_i$ of $G_2 = G[V(G) - N[v]]$ contains two vertices which do not have the same neighbors in $N(v)$, then it contains two adjacent such vertices. Then, the correctness of Algorithm Good-Partition-or-P4 follows from Corollary 2.1.

Time and processor complexity: The analysis of Algorithm Good-Partition-or-P4 is done on the PRAM model of computation; for details on the PRAM techniques mentioned below, see [3,13,23]. Let n and m be the number of vertices and edges of the input graph G; recall that the graph G is assumed to be given in adjacency-list representation where additionally for each edge uv of G, the two records in the adjacency lists of u and v are linked together.

Step 1: 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 in G. 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
  1.1.  for each vertex $y$ in the adjacency list $List(x)$ of $x$ do in parallel
            $A_x[r_x(y)] \leftarrow y$;
  1.2.  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
  1.3.  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 $A_w[r_w(u)]$;
  1.4.  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 (on n_1 vertices and m_1 edges). Using standard and simple parallel techniques, such as interval broadcasting and array packing, it is easy to see that the linked list representation of G_1 can be computed in $O(\log n_1)$ time with $O((n_1+m_1)/\log n_1)$ processors or in $O(\log n)$ time using $O((n_1+m_1)/\log n) = O((n+m)/\log n)$ processors, on the EREW PRAM model. The computation of the linked list representation of the induced subgraph G_2 is done in a fashion similar to that previously described and in the same time and processor complexity.

Step 2: The computation of the co-components of the graph G_1 can be optimally done in $O(\log n_1)$ time using $O((n_1+m_1)/\log n_1)$ processors, or in $O(\log n)$ time using $O((n_1+m_1)/\log n) = O((n+m)/\log n)$ processors, on the EREW PRAM model [7].

Step 3: Here, we use Algorithm Components-or-P4 that we have presented in Section 3, and either compute the connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ of the graph G_2 or detect that the graph G_2 contains a P_4 as an induced subgraph. Thus, if the number of vertices of G_2 is n_2 and its number of edges is m_2, the step is executed in $O(\log n_2)$ time using $O((n_2+m_2)/\log n_2)$ processors or in $O/(\log n)$ time using $O((n_2+m_2)/\log n) = O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 4: In this step, we check whether for each pair $\hat{\mathscr{C}}_i, \mathscr{C}_j$, the co-component $\hat{\mathscr{C}}_i$ either sees or misses the connected component $\mathscr{C}_j$, where $1 \leqslant i \leqslant \ell$ and $1 \leqslant j \leqslant k$. To do that, we first construct a subgraph G^* of the graph G as follows:

$$V(G^*) = V(G) - \{v\};$$

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

we will use the graph G^* in the execution of Step 5 as well. An adjacency-list representation of G^* can easily be constructed from the graph G in $O(logn)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model (see Step 1). By taking advantage of the graph G^*, for each co-component $\hat{\mathscr{C}}_i$, we will check whether there exist two vertices $x, y \in \hat{\mathscr{C}}_i$ which are non-adjacent in G such that $\exists z \in V(G) - N[v]$ which is adjacent to y and is not adjacent to x in G^*. To do that for a co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, we work in two phases: first, we check whether there exist two vertices $x, y \in \hat{\mathscr{C}}_i$ which are not adjacent in G and have different number of neighbors in G^*; next, if all the vertices of the co-component $\hat{\mathscr{C}}_i$ have the same number of neighbors in G^*, then we check whether there exist two vertices $x, y \in \hat{\mathscr{C}}_i$ which are not adjacent in G and have no identical neighborhoods in G^*. It is important to note that if there exists any such pair of vertices x, y, then G contains a P_4 of the form $xvyz$ if $deg^*(x) \leqslant deg^*(y)$, or of the form $yvxz$ otherwise, where $z \in V(G) - N[v]$.

Each of the phases involves three substeps which are executed separately on each of the co-components of G_1 and three substeps which are executed on all the co-components together; note that any two vertices from different co-components are adjacent in G. In detail, Step 4 is as follows:

```
• For each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, do in parallel
    4.1. compute a linked list $LC_i$ containing the vertices in $\hat{\mathscr{C}}_i$;
    4.2. for each vertex $x \in LC_i$, compute the degree $deg^*(x)$ of $x$ in $G^*$;
    4.3. find a vertex $u$ with minimum degree in $G^*$ and, then, partition the vertices
         of the co-component $\hat{\mathscr{C}}_i$ into two vertex sets $S_{i,1}$ and $S_{i,2}$ as follows:
         $S_{i,1} = \{x \in \hat{\mathscr{C}}_i \mid deg^*(x) = deg^*(u)\}$,   and
         $S_{i,2} = \hat{\mathscr{C}}_i - S_{i,1}$.
• Check if there exist two vertices $x, y$ such that $xy \notin E(G)$ and $x \in S_{i,1}$ and
  $y \in S_{i,2}$ (then, $x, y$ belong to the same co-component of $G_1$, and in $G^*$ they
  have different number of neighbors belonging to $V(G) - N[v]$);
    4.4. compute the vertex sets $S_1 = \bigcup_{i=1}^{\ell} S_{i,1}$ and $S_2 = \bigcup_{i=1}^{\ell} S_{i,2}$;
         compute the graph $\tilde{G} = G[V(G) - S_2]$ and the degree $deg_{\tilde{G}}(x)$ of each vertex
         $x$ in $\tilde{G}$;
    4.5. for each vertex $x \in S_1$, do in parallel
             if $deg_G(x) < deg_{\tilde{G}}(x) + |S_2|$
             then {$x$ is not adjacent in $G$ to a vertex in $S_1$}
                 mark the vertex $x$;
```

```
4.6.  if there exists a marked vertex
      then select any such vertex $x$;
         call Subroutine Find-$P4$ on the graph $G$ and the vertex-pair $(x, v)$;
         stop and return the $P_4$ that Subroutine Find-P4 returned.
```

In case Step 4.6 finds no marked vertices, then we proceed to the second phase where we check whether all the vertices of each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, have identical neighborhoods in G^*. 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 $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ of the graph G_1. For each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, we use an auxiliary array $D_i[]$ of size $\hat{n}_i - 1$, and arrays $P_i[]$, $P_{i,j}[]$ $(1 \leqslant j \leqslant \hat{n}_i - 1)$, and $M_{i,x}[]$ $(x \in \hat{\mathscr{C}}_i - \{\hat{v}_i\})$, each of size equal to the degree $deg^*(\hat{v}_i)$ of the representative $\hat{v}_i$ in G^*. We proceed as follows:

```
•  For each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, do in parallel
   4.7.  copy the neighbors of $\hat{v}_i$ in $G^*$ in the array $P_i[1..deg^*(\hat{v}_i)]$;
   4.8.  make $\hat{n}_i - 1$ copies $P_{i,1}[], \ldots, P_{i,\hat{n}_i-1}[]$ of the array $P_i[]$;
   4.9.  $S_{i,1} \leftarrow \{\hat{v}_i\}$;   $S_{1,2} \leftarrow \emptyset$;
         for each vertex $x \in LC_i - \{\hat{v}_i\}$, do in parallel
            ∘  copy the neighbors of $x$ in $G^*$ in the array $M_{i,x}[]$;
            ∘  if $M_{i,x}[] = P_{i,r_i(x)}[]$, where $r_i(x)$ is the rank of $x$ in $LC_i - \{\hat{v}_i\}$,
               then insert vertex $x$ in the set $S_{i,1}$;
               else insert vertex $x$ in the set $S_{i,2}$.
•  Check if there exist two vertices $x, y$ such that $xy \notin E(G)$ and $x \in S_{i,1}$ and
   $y \in S_{i,2}$ (then, $x, y$ belong to the same co-component of $G_1$, and in $G^*$ they
   have different neighborhoods), by executing Steps 4.4–4.6 for the sets $S_{i,1}$ and
   $S_{i,2}$ computed in Step 4.9;
```

For the correctness of the computation, observe that if the condition "$deg_G(x) < deg_{\tilde{G}}(x) + |S_2|$" in Step 4.4 is true, then x is not adjacent to a vertex in S_2. Then, for the sets $S_{i,1}$ and $S_{i,2}$ computed in Step 4.3, this is equivalent to the existence of a vertex y such that $xy \notin E(G)$ and $deg^*(y) > deg^*(x)$ or equivalently $|N(y) - N[v]| > |N(x) - N[v]|$; for the sets $S_{i,1}$ and $S_{i,2}$ computed in Step 4.9, this is equivalent to the existence of a vertex y such that $xy \notin E(G)$, $|N(y) - N[v]| = |N(x) - N[v]|$, and $N(y) - N[v] \neq N(x) - N[v]$. In either case, there exists a vertex $z \in V(G) - N[v]$ such that $yz \in E(G)$ and $xz \notin E(G)$; this implies that the graph G contains the P_4 $xvyz$ as reported by the algorithm thanks to Subroutine Find-P4.

We next compute the time and processor complexity of Step 4 of Algorithm Good-Partition-or-P4 by analyzing Steps 4.1–4.8.

Having computed the vertices of each co-component $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, we can easily construct the linked list LC_i (Step 4.1) in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

The computation of the degree $deg^*(x)$ of a vertex x of the graph G^* can be done by applying list ranking on the adjacency list of x in G^* and by taking the maximum rank; this can be done in $O(\log n)$ time using $O(deg^*(x)/\log n)$ processors on the EREW PRAM. Since the graph G^* has $n-1$ vertices and $O(m)$ edges, the computation for all the vertices takes $O(\log n)$ time and $O((n+m)/\log n)$ processors on the same model of computation. Additionally, finding a vertex of $\hat{\mathscr{C}}_i$ of minimum degree in G^* can be easily done optimally. For the construction of the sets $S_{i,1}$ and $S_{i,2}$, we use two auxiliary arrays of size $\hat{n}_i$ each, in which we first set the entries of the vertices of each set equal to the respective vertex and then use array packing to collect these vertices together. Thus, all the operations in Steps 4.2 and 4.3 can be executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Forming the sets S_1 and S_2 is done in a fashion similar to forming the sets $S_{i,1}$ and $S_{i,2}$, hence, in $O(\log n)$ time and $O(n/\log n)$ processors on the EREW PRAM model. The computation of the adjacency-list representation of the induced subgraph $\tilde{G}$ can be computed using standard and simple parallel techniques, such as list ranking, interval broadcasting, and array packing [3,13,23]; see Step 1. If $|V(\tilde{G})| = \tilde{n}$ and $|E(\tilde{G})| = \tilde{m}$, this computation can be done in $O(\log \tilde{n})$ time with $O((\tilde{n}+\tilde{m})/\log \tilde{n})$ processors or in $O(\log n)$ time with $O((\tilde{n}+\tilde{m})/\log n) = O((n+m)/\log n)$ processors on the EREW PRAM model. Moreover, the degrees of all the vertices in $\tilde{G}$ can also be computed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

In order to avoid concurrent read operations while checking the if-condition in Step 4.5, we make $\hat{n}_i$ copies of the value $|S_2|$ in an auxiliary array $Q_i[1..\hat{n}_i]$; this computation can be easily done in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Since the number of marked vertices x is less that n, the selection of a marked vertex in Step 4.6 can be done in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. Additionally, from Theorem 3.2, Subroutine Find-P4 has the same time and processor complexity on the EREW PRAM model.

If the algorithm does not return in Step 4.6, then, for every vertex $x \in \hat{\mathscr{C}}_i$, $deg^*(x) = deg^*(\hat{v}_i)$. Since $deg^*(x)$ is less than the degree of x in G, it follows that $\sum_{i=1}^{\ell} (\hat{n}_i \cdot deg^*(\hat{v}_i)) = O(m)$, and thus Steps 4.7 and 4.8 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, Step 4.9 is 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. If not, then in the same time and processor complexity we find an induced P_4 by means of Subroutine Find-P4.

Step 5: Processing the vertices of the connected components $\mathscr{C}_i$, $1 \leqslant i \leqslant k$, is done in a similar fashion using the graph G^*: we look for an edge xy of G, where $x, y \in \mathscr{C}_i$ such that either $deg^*(x) < deg^*(y)$ or $deg^*(x) = deg^*(y)$ and the vertices x, y have no identical neighborhoods in G^*. Such an edge xy is a wing of a P_4 $xyav$ in G, where $a \in N(v)$. Based on the time and processor complexity of Steps 4.1–4.9, we can show that the execution of Step 5 takes $O(\log n)$ time and requires $O((n+m)/\log n)$ processors on the EREW PRAM model as well.

Step 6: Here, we sort the co-components $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ in non-decreasing number of the connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ 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 $\hat{\mathscr{C}}_i$ sees; then, a_i is equal to the degree of the representative $\hat{v}_i$ of $\hat{\mathscr{C}}_i$ in the subgraph of the graph G^* induced by the representatives of the co-components $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, and the components $\mathscr{C}_j$, $1 \leqslant j \leqslant k$. 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 2.3), 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 $\sqrt{m} \geqslant \log n$ then $\sqrt{m} = O(m/\log n)$.

Step 7: For simplicity, we assume that $(\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell)$ 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 6 in the description of the algorithm). 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_j, $1 \leqslant j \leqslant k$, is the number of co-components of the graph G_1 that the connected component $\mathscr{C}_j$ sees. Then, we implement Step 7 as follows:

```
• For each connected component $\mathscr{C}_j$, $1 \leqslant j \leqslant k$, do in parallel
  7.1. find the co-component $\hat{\mathscr{C}}_i$ with the minimum index that the representative $v_j$
       of $\mathscr{C}_j$ sees;
  7.2. if $b_j \neq \ell - i + 1$
       then select a vertex $x$ from $\mathscr{C}_j$ and a vertex $y$ from $\hat{\mathscr{C}}_i$;
            call Subroutine Find-$P4$ on the graph $G$ and the vertex-pair $(x, y)$;
            stop and return the $P4$ that Subroutine Find-P4 returned.
```

The correctness of the computation follows from Corollary 2.1, condition (ii): note that if $b_j \neq \ell - i + 1$, then there exists a co-component $\hat{\mathscr{C}}_p$ where $p > i$ such that $\mathscr{C}_j$ does not see $\hat{\mathscr{C}}_p$. Since $p > i$, we have that $|\hat{I}_p| \geqslant |\hat{I}_i|$, and since $j \in \hat{I}_i - \hat{I}_p$, there exists $q \in \hat{I}_p - \hat{I}_i$; thus, G contains a P_4 of the form shown in Fig. 1(c), which proves the correctness of the computation.

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 $\mathscr{C}_j$ sees, can also be computed within the same time and processor bounds. Thus, Step 7 is executed in $O(\log n)$ time with $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 8: In this step, the algorithm returns in $O(1)$ sequential time the message that the component-partition of the input graph G with respect to $v \in V(G)$ is good, i.e., G contains no P_4 with vertices in both $N[v]$ and $V(G) - N[v]$.

Taking into consideration the time and processor complexity of each step of the algorithm Good-Partition-or-P4, we obtain the following result.

Theorem 4.1. *Algorithm Good-Partition-or-$P4$ runs in* $O(\log n)$ *time using* $O((n+m)/\log n)$ *processors on the EREW PRAM model*.

5. The recognition algorithm

In this section, we present a parallel algorithm for recognizing whether a given graph G is a cograph, and if it is not, a P_4 of G is returned. As in Section 4, the input graph G is assumed to be given in adjacency-list representation where additionally for each edge uv of G, the two records in the adjacency lists of u and v are linked together.

```
Algorithm Recognize-Cograph
Input:   an undirected graph $G$ on $n$ vertices and $m$ edges.
Output:  either a message that $G$ is a cograph or a $P_4$ of $G$.

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 $\hat{\mathscr{A}}_1, \hat{\mathscr{A}}_2, \ldots, \hat{\mathscr{A}}_p$ of $G$;
   (b) if any of the co-components $\hat{\mathscr{A}}_i$, $1 \leqslant i \leqslant p$, has cardinality at least equal to $\frac{1}{2}n$
       then {$G$ is not a cograph}
           call Algorithm Good-Partition-or-$P4$ on the graph $G[\hat{c}\mathscr{A}_i]$ and an arbitrary
           vertex $x$ in $\hat{\mathscr{A}}_i$; (the algorithm returns with a $P_4$ of $G$)
       else compute the subgraphs $G[\hat{\mathscr{A}}_1], G[\hat{\mathscr{A}}_2], \ldots, G[\hat{\mathscr{A}}_t]$ of $G$ and
           call recursively Algorithm Recognize-Cograph on each of these subgraphs;
   (c) go to Step 7.
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. Call Algorithm Good-Partition-or-$P4$ on the graph $G$ and the vertex $v$;
   if the algorithm returns a $P_4$ then go to Step 7.
5. Compute the induced subgraphs $G[\hat{\mathscr{C}}_i]$ and $G[\mathscr{C}_j]$, where $\hat{\mathscr{C}}_i$, $1 \leqslant i \leqslant \ell$, are the
   co-components of the subgraph $G[N(v)]$, and $\mathscr{C}_j$, $1 \leqslant j \leqslant k$, are the connected
   components of $G[V(G) - N[v]]$.
6. Call recursively Algorithm Recognize-Cograph on each $G[\hat{\mathscr{C}}_i]$ $(1 \leqslant i \leqslant \ell)$, and
   on each $G[\mathscr{C}_j]$ $(1 \leqslant j \leqslant k)$ such that $|\mathscr{C}_j| \leqslant \frac{3}{4}n$;
   if there exists a $\mathscr{C}_i$ such that $|\mathscr{C}_i| > \frac{3}{4}n$ then
   (a) compute the co-components $\hat{\mathscr{A}}_{i,1}, \hat{\mathscr{A}}_{i,2}, \ldots, \hat{\mathscr{A}}_{i,h}$ of $G[\mathscr{C}_i]$;
   (b) if any of the co-components $\hat{\mathscr{A}}_{i,j}$, $1 \leqslant j \leqslant h$, has cardinality at least equal to $\frac{1}{2}n$
       then {$G$ is not a cograph}
```

```
            call Algorithm Good-Partition-or-$P4$ on the graph $G[\mathscr{C}_i]$ and an
            arbitrary $y$ in $\mathscr{C}_i$;
            call Algorithm Good-Partition-or-$P4$ on the graph $G[\hat{\mathscr{A}}_{i,j}]$ and an
            arbitrary vertex $z$ in $\hat{\mathscr{A}}_{i,j}$; (at least one of these calls returns with a
            $P4$ of $G$)
         else compute the subgraphs $G[\hat{\mathscr{A}}_{i,1}]$, $G[\hat{\mathscr{A}}_{i,2}]$, ..., $G[\hat{\mathscr{A}}_{i,h}]$ of $G$ and
            call recursively Algorithm Recognize-Cograph on each of these subgraphs.
7.    If any call to Algorithm Recognize-Cograph or Good-Partition-or-$P4$ returns
      with a $P_4$
      then return such a $P_4$;
      else return that $G$ is a cograph.
```

It is important to note that the following lemma holds.

Lemma 5.1. *If during the execution of Algorithm Recognize-Cograph 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 6 of Algorithm Recognize-Cograph; in either case, the graphs on which the calls are executed are subgraphs of the input graph G. When Recognize-Cograph is called on a subgraph $G[\hat{\mathscr{A}}_i]$ of G in Step 2, then the number of vertices of the subgraph is less than $\frac{1}{2}n$. Moreover, when Recognize-Cograph is called on a subgraph $G[\hat{\mathscr{C}}_i]$ or a subgraph $G[\mathscr{C}_j]$ such that $|\mathscr{C}_j| \leqslant \frac{3}{4}n$ in Step 6, then the number of vertices of these subgraphs does not exceed $\frac{3}{4}n$; note that v belongs to the set M or the set L, and $|\hat{\mathscr{C}}_i| \leqslant |N(v)| \leqslant \frac{3}{4}n$. Finally, if in Step 6 there exists a component $\mathscr{C}_i$ such that $|\mathscr{C}_i| > \frac{3}{4}n$, 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 $G[\mathscr{C}_i]$ does not exceed $\frac{1}{2}n$. □

Correctness: The correctness of Step 4 of Algorithm Recognize-Cograph readily follows from the correctness of Algorithm Good-Partition-or-P4 (see Section 4). The correctness of Step 2 follows from Lemma 2.3 and the fact that a graph is a cograph iff each of its co-components is a cograph (note that a graph contains no P_4 with vertices in more than one co-component). The latter observation also helps establish the correctness of Step 6 along with Corollary 2.1 and Observation 2.2 and Lemma 2.4; note that if there exists a component $\mathscr{C}_i$ such that $|\mathscr{C}_i| > \frac{3}{4}n$, then Observation 2.2 implies that $v \notin M$ which is true only if $M = \emptyset$ and thus the conditions of Lemma 2.4 hold. Finally, it is important to note that if the component-partition of a graph G with respect to a vertex v is good, then G contains a P_4 iff a co-component of $G[N(v)]$ or a component of $G[V(G) - N[v]]$ contains a P_4.

Time and processor complexity: We use a step-by-step analysis for computing the time and processor complexities of each step of Algorithm Recognize-Cograph on the PRAM model of computation (see [3,13,23]).

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 $O((n+m)/\log n)$ 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 [7], and so can the subgraphs of G induced by each of these co-components. Additionally, since $G[\hat{\mathscr{A}}_i]$ has at least $n/2$ and no more than n vertices and $O(m)$ edges, the execution of Algorithm Good-Partition-or-P4 takes $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model. Thus, if we ignore the time for any recursive calls to Algorithm Recognize-Cograph, Step 2 takes $O(\log n)$ time and $O((n+m)/\log n)$ processors on the EREW PRAM model.

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: The step takes $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model (Theorem 4.1).

Step 5: The induced subgraphs $G[\hat{\mathscr{C}}_i]$, $1 \leqslant i \leqslant \ell$, and $G[\mathscr{C}_j]$, $1 \leqslant j \leqslant k$, can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.

Step 6: The processing of a component $\mathscr{C}_i$ such that $|\mathscr{C}_i| > \frac{3}{4}n$ is identical to the processing in Step 2. Thus, if we ignore the time for any recursive calls to Algorithm Recognize-Cograph, Step 6 takes $O(\log n)$ time and $O((n+m)/\log n)$ processors on the EREW PRAM.

Step 7: Since the calls to Algorithms Recognize-Cograph and Good-Partition-or-P4 are executed on subgraphs of the graph G which are vertex disjoint, we can use an array of size n (initialized to 0) where the different calls store their results, a P_4 of G if a P_4 was found, or 0 otherwise. Then, packing this array so that the 0-entries are suppressed suffices for checking whether a P_4 has been returned and if yes, for obtaining such a P_4. Thus, Step 7 can be completed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

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

$$T(n, m) = \mathrm{O}(\log n) + \max_i\{T(n_i, m_i)\},$$
$$P(n, m) = \max\left\{\mathrm{O}((n+m)/\log n), \sum_i P(n_i, m_i)\right\},$$

where n_i and m_i are the numbers of vertices and edges of the subgraphs on which Algorithm *Recognize-Cograph* is recursively called. Since $\sum_i n_i \leqslant n$, $\sum_i m_i \leqslant m$, and for each i, $n_i \leqslant 3n/4$ (see Lemma 5.1), the equalities for $T(n, m)$ and $P(n, m)$ admit the solution: $T(n, m) = \mathrm{O}(\log^2 n)$, $P(n, m) = \mathrm{O}((n+m)/\log n)$. Thus, we obtain the following results.

Theorem 5.1. *Algorithm Recognize-Cograph runs in* $\mathrm{O}(\log^2 n)$ *time using* $\mathrm{O}((n+m)/\log n)$ *processors on the EREW PRAM model*.

Corollary 5.1. *Cographs can be recognized in* $\mathrm{O}(\log^2 n)$ *time with* $\mathrm{O}((n+m)/\log n)$ *processors on the EREW PRAM model of computation*.

6. Constructing the cotree or finding a P_4

Given a graph, we give below a parallel algorithm which constructs its cotree if the input graph is a cograph, or otherwise prints an induced P_4. The algorithm first calls Algorithm Recognize-Cograph on the input graph to determine whether it is a cograph and to provide a P_4 if it is not. If the graph is a cograph, then the algorithm constructs its cotree by taking advantage of Lemma 2.2 which gives the structure of the cotree of a cograph in terms of the graph's component-partition with respect to any of its vertices. In particular, 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 the subgraph $G[N(v)]$ and the connected components of the subgraph $G[V(G) - N[v]]$, and then uses Lemma 2.2 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 where additionally for each edge uv of G, the two records in the adjacency lists of u and v are linked together.

```
Algorithm Cotree-or-$P4$
Input:  an undirected graph.
Output: the root-node of the cotree of the input graph if it is a cograph, or an induced
        $P_4$ otherwise.
```

```
1.    Execute Algorithm *Recognize-Cograph* on the input graph.
2.    If Algorithm *Recognize-Cograph* returns a $P_4$
      then return this $P_4$;
      else {the input graph is a cograph}
           execute Subroutine *Construct-Cotree* on the input graph and return the root of
           the cotree that it has constructed,
```

where the description of Subroutine Construct-Cotree is given below.

```
Subroutine 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   $\{each\ v \in V(G)\ has\ degree\ deg(v) > \frac{3}{4}n\}$.
   (a) compute the co-components $\hat{\mathscr{A}}_1, \hat{\mathscr{A}}_2, \ldots, \hat{\mathscr{A}}_p$ of the graph $G$;
   (b) construct a 1-node $r$;
   (c) for $i = 1, 2, \ldots, p$ do in parallel
           compute the induced subgraph $G[\hat{\mathscr{A}}_i]$;
           apply recursively Subroutine Construct-Cotree on $G[\hat{\mathscr{A}}_i]$; 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 $\hat{\mathscr{C}}_1, \hat{\mathscr{C}}_2, \ldots, \hat{\mathscr{C}}_\ell$ of the graph $G_1 = G[N(v)]$;
   for $i = 1, 2, \ldots, \ell$ do in parallel
           compute the induced subgraph $G[\hat{\mathscr{C}}_i]$;
           apply recursively Subroutine Construct-Cotree on $G[\hat{\mathscr{C}}_i]$; let $\hat{r}_i$ be the
           root-node of the returned tree.
5. Compute the connected components $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ of the graph $G_2 =$
   $G[V(G) - N[v]]$;
   for $i = 1, 2, \ldots, k$ do in parallel
       compute the induced subgraph $G[\mathscr{C}_i]$;
       if $|\mathscr{C}_i| \leqslant \frac{3}{4}n$
       then apply recursively Subroutine Construct-Cotree on $G[\mathscr{C}_i]$; let $r_i$ be
            the root-node of the returned tree;
       else construct a 1-node $r_i$;
             compute the co-components $\hat{\mathscr{A}}_{i,j}$, $1 \leqslant j \leqslant h$, of the graph $G[\mathscr{C}_i]$ and the
             induced subgraphs $G[\hat{\mathscr{A}}_{i,j}]$;
            for $j = 1, 2, \ldots, h$ do in parallel
                apply recursively Subroutine *Construct-Cotree* on $G[\hat{\mathscr{A}}_{i,j}]$; 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 $\tilde{G}$ of $G$ spanned by the edges incident upon a co-component
   representative $\hat{v}_i$ $(1 \leqslant i \leqslant \ell)$ and a component representative $v_j$ $(1 \leqslant j \leqslant k)$;
   compute the degrees of the $\hat{v}_i$s, $1 \leqslant i \leqslant \ell$, in $\tilde{G}$, sort them in non-decreasing
   order, and locate
   the distinct values; let $\delta[i]$, $1 \leqslant i \leqslant \ell'$, be the resulting ordered sequence.
7. Compute the entries of an array $pos[i]$, $1 \leqslant i \leqslant \ell$, such that $pos[i] = j$ if and only
   if the degree of $\hat{v}_i$ in $\tilde{G}$ is equal to $\delta[j]$.
8. Construct a tree-path of alternating 1- and 0-nodes as follows:
   (a) construct $\ell'$ 1-nodes $\hat{t}_i$, $1 \leqslant i \leqslant \ell'$, and $\ell'$ 0-nodes $t_j$, $1 \leqslant j \leqslant \ell'$;
       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 $\tilde{G}\}$;
   (c) if there exist component representatives $v_i$ in $\tilde{G}$ of degree equal to 0
       then construct a 0-node $r$;
            for each component representative $v_i$ of degree equal to 0 in $\tilde{G}$ do
                $parent(r_i) \leftarrow r$;
            $parent(\hat{t}_{\ell'}) \leftarrow r$;
       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 5, 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 2.2: 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 $\hat{S}_i$, 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 Fig. 3); 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: Since the execution of Algorithm Recognize-Cograph on a graph on n vertices and m edges takes $\mathrm{O}(\log^2 n)$ time using $\mathrm{O}((n+m)/\log n)$ processors on the EREW PRAM (Theorem 5.1), it suffices to compute the time and processor complexities of each step of Subroutine Construct-Cotree.

Steps 1–5: All the operations performed in these steps are also performed in Steps 1–3, 5, and 6 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 Subroutine Construct-Cotree takes $\mathrm{O}(\log n)$ time and requires $\mathrm{O}((n+m)/\log n)$ processors on the EREW PRAM model.

Step 6: The subgraph $\tilde{G}$ coincides with the subgraph of the graph G^* (see the analysis of Step 4 of Algorithm Good-Partition-or-P4) 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 1 of Algorithm Good-Partition-or-P4. Thus, $\tilde{G}$'s construction takes $\mathrm{O}(\log n)$ time and requires $\mathrm{O}((n+m)/\log n)$ 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 $\tilde{G}$ 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 $\tilde{G}$, $1 \leqslant i \leqslant \ell$. Since the number of co-components is $\mathrm{O}(\sqrt{m})$ according to Observation 2.3, the array $d[]$ can be sorted in $\mathrm{O}(\log n)$ time with $\mathrm{O}((n+m)/\log n)$ 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 $\delta[i]$, $1 \leqslant i \leqslant \ell'$, can be computed in $\mathrm{O}(\log n)$ time with $\mathrm{O}((n+m)/\log n)$ 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 $d[\pi(i)] \leqslant d[\pi(j)]$ for every $1 \leqslant i < j \leqslant \ell$. 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 $\mathrm{O}(\log n)$ time using $\mathrm{O}(n/\log n)$ processors on the EREW PRAM model.

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

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

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

```
• For each co-component representative $\hat{v}_i$, $1 \leqslant i \leqslant \ell$, do in parallel
  9.1. copy the value $pos[i]$ to each entry of $A_{\hat{v}_i}[]$.
• For each component representative $v_i$, $1 \leqslant i \leqslant k$, do in parallel
  9.2. for each vertex $u$ adjacent to $v_i$ in $\tilde{G}$ 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
         adjacency list of $x$ in $\tilde{G}$;
  9.3. $p_i \leftarrow$ the minimum element of the array $A_{v_i}[]$.
```

It is easy to see that the above Steps 9.1–9.3 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 Subroutine 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 6.1. *Algorithm Cotree-or-$P4$ runs in $O(\log^2 n)$ time using $O((n + m)/\log n)$ processors on the EREW PRAM model.*

Corollary 6.1. *Let G be a graph on n vertices and m edges. Then, constructing the cotree of G if G is a cograph, or finding an induced P_4 otherwise, can be done in $O(\log^2 n)$ time with $O((n + m)/\log n)$ processors on the EREW PRAM model.*

7. Concluding remarks

In this paper, we have presented parallel algorithms for recognizing cographs and for constructing the cotree of a graph if it is a cograph; if the input graph is not a cograph, the algorithms return an induced P_4. 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 [10,12]. 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 [14,15]. Recently, Lin and Olariu presented parallel recognition and tree construction algorithms for P_4-sparse graphs [20]; for an input graph on n vertices and m edges, both the recognition and the tree construction algorithms run in $\mathrm{O}(\log n)$ time using $\mathrm{O}((n^2+nm)/\log n)$ 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.

Acknowledgements

The authors would like to thank the anonymous referees for their suggestions which improved the presentation of the paper and motivated them to extend the algorithms so that they produce a P_4 as a certificate whenever the input graph is not a cograph.

References

[1] G.S. Adhar, S. Peng, Parallel algorithms for cographs and parity graphs with applications, J. Algorithms 11 (1990) 252–284.
[2] G.S. Adhar, S. Peng, Parallel algorithms for path covering, Hamiltonian path, Hamiltonian cycle in cographs, Proceedings of the International Conference on Parallel Processing, vol. III, 1990, pp. 364–365.
[3] S.G. Akl, Parallel Computation: Models and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[4] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (1993) 181–188.
[5] A. Brandstädt, V.B. Le, J.P. Spinrad, Graph Classes: A Survey, SIAM Monographs on Discrete Mathematics and Applications, 1999.
[6] A. Bretscher, D. Corneil, M. Habib, C. Paul, A simple linear time LexBFS cograph recognition algorithm, Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG ’03), 2003, pp. 119–130.
[7] K.W. Chong, S.D. Nikolopoulos, L. Palios, An optimal parallel co-connectivity algorithm, Theory Comput. Systems 37 (2004) 527–546.
[8] D.G. Corneil, H. Lerchs, L. Stewart-Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.
[9] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (1985) 926–934.
[10] E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance hereditary graphs, Discrete Appl. Math. 57 (1995) 29–44.
[11] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, Inc., New York, 1980.
[12] X. He, Parallel algorithm for cograph recognition with applications, J. Algorithms 15 (1993) 284–313.
[13] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1992.
[14] B. Jamison, S. Olariu, Recognizing P_4-sparse graphs in linear time, SIAM J. Comput. 21 (1992) 381–406.
[15] B. Jamison, S. Olariu, A linear-time recognition algorithm for P_4-reducible graphs, Theoret. Comput. Sci. 145 (1995) 329–344.
[16] H.A. Jung, On a class of posets and the corresponding comparability graphs, J. Combin. Theory Ser. B 24 (1978) 125–133.
[17] D.G. Kirkpatrick, T. Przytycka, Parallel recognition of complement reducible graphs and cotree construction, Discrete Appl. Math. 29 (1990) 79–96.

[18] H. Lerchs, On cliques and kernels, Technical Report, Department of Computer Science, University of Toronto, March 1971.
[19] R. Lin, S. Olariu, An NC recognition algorithm for cographs, J. Parallel Distributed Comput. 13 (1991) 76–90.
[20] R. Lin, S. Olariu, A fast parallel algorithm to recognize $P4$-sparse graphs, Discrete Appl. Math. 81 (1998) 191–215.
[21] R. Lin, S. Olariu, G. Pruesse, An optimal path cover algorithm for cographs, Comput. Math. Appl. 30 (1995) 75–83.
[22] K. Nakano, S. Olariu, A.Y. Zomaya, A time-optimal solution for the path cover problem on cographs, Theoret. Comput. Sci. 290 (2003) 1541–1556.
[23] J. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann, Los Altos, CA, 1993.
[24] D.P. Sumner, Dacey graphs, J. Austral. Math. Soc. 18 (1974) 492–502.