1998

# A Fast Parallel Algorithm to Recognize P4-Sparse Graphs

Rong Lin

Stephan Olariu
*Old Dominion University*



## Original Publication Citation

Lin, R., & Olariu, S. (1998). A fast parallel algorithm to recognize P4-sparse graphs. *Discrete Applied Mathematics, 81*(1-3), 191-215. doi:10.1016/s0166-218x(97)00085-1

ELSEVIER

Discrete Applied Mathematics 81 (1998) 191–215

DISCRETE APPLIED MATHEMATICS

# A fast parallel algorithm to recognize P4-sparse graphs ☆

Rong Lin [a], Stephan Olariu [b,*]

[a] *Department of Computer Science, SUNY Geneseo, Geneseo, NY 14454, USA*
[b] *Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162, USA*

Received 8 April 1991; revised 5 May 1997

**Abstract**

A number of problems in mobile computing, group-based collaboration, automated theorem proving, networking, scheduling, and cluster analysis suggested the study of graphs featuring certain "local density" characteristics. Typically, the notion of local density is equated with the absence of chordless paths of length three or more. Recently, a new metric for local density has been proposed, allowing a number of such induced paths to occur. More precisely, a graph $G$ is called P4-sparse if no set of five vertices in $G$ induces more than one chordless path of length three. P4-sparse graphs generalize the well-known class of cographs corresponding to a more stringent local density metric. One remarkable feature of P4-sparse graphs is that they admit a tree representation unique up to isomorphism. In this work we present a parallel algorithm to recognize P4-sparse graphs and show how the data structures returned by the recognition algorithm can be used to construct the corresponding tree representation. With a graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ as input, our algorithms run in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors in the EREW-PRAM model.

*Keywords*: Cographs; Mobile computing; Group-based collaboration; Scheduling; Cluster analysis; NC algorithms; Wireless networks; Parallel algorithms; P4-sparse graphs; Shared memory model

## 1. Introduction and motivation

In recent years a number of problems originating in mobile computing, networking, scheduling, group-based collaboration, and cluster analysis, have suggested the study of graphs featuring a number of "local density" properties (see [5, 6, 9, 11, 20, 27, 31] for more details). Typically, researchers tend to equate the notion of local density with the absence of chordless paths of length three, hereinafter referred to as $P_4$s.

☆ Work supported by the NSF grants CCR-9522093 and MIP-9307664, and by ONR grants N00014-95-1-0779 and N00014-97-1-0526.

* Corresponding author. E-mail: olariu@cs.odu.edu.


*PII* S0166-218X(97)00085-1

In examination scheduling, for example, a *conflict graph* is readily constructed: the vertices represent different courses offered, while courses $x$ and $y$ are linked by an edge if, and only if, some student takes both of them. (In the weighted version, the weight of edge $xy$ stands for the number of students taking both $x$ and $y$.) Clearly, in any coloring of the conflict graph, vertices that are assigned the same color correspond to courses whose examinations can be held concurrently. It is usually anticipated that very few paths of length three will occur in the conflict graph. In the second application, to evaluate the clustering of, say, index terms, we construct a graph whose vertices are the index terms; an edge occurs between two index terms to denote self-referencing or semantic proximity. Again, very few $P_4$s are expected to occur.

These applications have motivated both the theoretical and algorithmic study of the classes of cographs [4–6, 15, 23, 24, 28–30] and P4-reducible graphs [14, 15] corresponding, respectively, to the local density metrics ($\mu 1$) and ($\mu 2$) described below:

($\mu 1$) the graph contains no induced $P_4$;

($\mu 2$) every vertex of the graph belongs to at most one induced $P_4$.

One of the most desirable properties of a graph $G$ is a unique tree representation; more precisely, this involves associating with $G$ a *unique* rooted tree $T(G)$ whose leaves are elements of $G$ (e.g. vertices, edges, maximal cliques, maximal stable sets, cutsets) and whose internal nodes correspond to certain graph operations. If $T(G)$ can be obtained *efficiently* (i.e. in polynomial time in the size of the graph $G$), and if the leaves of $T(G)$ can be tested for isomorphism in polynomial time, then the graph isomorphism problem (which is still open for arbitrary graphs) can be solved efficiently for $G$, since it reduces to tree isomorphism. Unique tree representations have been obtained for several classes of graphs including the cographs [23, 24], hook-up graphs [21], transitive series parallel digraphs [22], interval graphs [3], rooted directed path graphs [1], maximal outerplanar graphs, and P4-reducible graphs [14].

Recently, Hoang [10] and Jamison and Olariu [16] proposed a new local density metric in graphs:

($\mu 3$) no set of five vertices induces more than one $P_4$,

and argued that the class of graphs that naturally corresponds to this metric (the P4-sparse graphs) features a number of remarkable theoretical and algorithmic properties, including a unique tree representation up to isomorphism. In addition, in practical applications, metric ($\mu 3$) is less restrictive and, hence, more realistic than both ($\mu 1$) and ($\mu 2$). At the same time, it is immediate that the P4-sparse graphs generalize both the cographs and the P4-reducible graphs.

Quite recently, an incremental algorithm to recognize P4-sparse graphs and to construct the corresponding tree representation was proposed [17]. Although this algorithm runs in linear time (being, thus, optimal) its incremental nature does not lend itself naturally to parallel processing. The purpose of this paper is to propose a new characterization of P4-sparse graphs and to show that it yields a fast parallel recognition algorithm for this class of graphs. Furthermore, our recognition algorithm is subsequently used to obtain the unique tree associated with a P4-sparse graph.

The model of computation that we shall adopt is the parallel random access machine (PRAM, for short) in which all the processors have access to a common memory and run synchronously. It is further assumed that simultaneous reading from the same memory location, as well as simultaneous writing by several processors into the same memory location is prohibited: this submodel is referred to as EREW-PRAM (see [12] for an excellent survey of different models).

Our first major contribution is to provide a novel way of looking at P4-sparse graphs in terms of regular sets. The concept of a regular set is interesting in its own right and may find applications to elucidating the structure of other classes of graphs.

Our second major contribution is to show that the new characterization of P4-sparse graphs in terms of regular sets can be exploited to obtain a fast parallel recognition algorithm for this class of graphs. Specifically, with an arbitrary graph $G$ with $n$ vertices and $m$ edges as input, our recognition algorithm runs in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors in the EREW model. In case $G$ turns out to be a P4-sparse graph, our algorithm also constructs the corresponding tree representation.

Our parallel recognition algorithm builds on the parallel cograph recognition algorithm that the authors have recently devised [25]. We note that other parallel recognition algorithms have been devised. For example, Dahlhaus [7] has proposed a recognition algorithm for cographs running in $O(\log^2 n)$ time using $O(n+m)$ processors in the CREW-PRAM model of computation. Yet another such algorithm is contained in [8]. However, the algorithm in [8] runs in $O(\log^2 n)$ time using $O(n+m)$ processors in the CRCW model of computation. It would be interesting to see if the cograph recognition algorithms of [7, 8] can be extended to recognize P4-sparse graphs.

The remainder of this paper is organized as follows: Section 2 introduces the terminology and gives background information about cographs and P4-sparse graphs; Section 3 gives the new characterization for P4-sparse graphs which is at the heart of our parallel recognition for P4-sparse graphs; Section 4 presents the recognition algorithms; Section 5 deals with the task of constructing the tree representation of P4-sparse graphs; finally, Section 6 summarizes the results and proposes a number of open problems.

## 2. Background and terminology

All the graphs in this work are finite, with no loops or multiple edges. We use standard graph-theoretical terminology compatible with Bondy and Murty [2]. In addition, we use some new terms that we are about to define. For a vertex $x$ of a graph $G=(V,E)$, $N(x)$ will denote the set of all vertices of $G$ which are adjacent to $x$: we assume adjacency to be non-reflexive, and so $x \notin N(x)$; we let $N[x]$ stand for $N(x) \cup \{x\}$. As usual, we let $d_G(x)$ stand for $|N(x)|$. A vertex $z$ is said to *distinguish* between vertices $u$ and $v$ whenever $z$ is adjacent to precisely one of $u$, $v$. In the remaining part of this work we shall often associate, in some way, rooted trees with graphs. In this context, we shall refer to the vertices of trees as *nodes*. For a node $w$ in a tree $T$,

we let $p(w)$ stand for the parent of $w$ in $T$. The degree of a node $w$ in $T$ is denoted by $d(w)$.

To make this paper self-contained, we shall review some of the properties of cographs and P4-sparse graphs. To begin, Lerchs [23] showed how to associate with every cograph $G$ a unique tree $T(G)$ called the *cotree* of $G$, and defined as follows:

- every internal node, except possibly for the root, has at least two children;
- the internal nodes are labeled by either 0 (0-nodes) or 1 (1-nodes) in such a way that the root is always a 1-node, and such that 1-nodes and 0-nodes alternate along every path in $T(G)$ starting at the root;
- the leaves of $T(G)$ are precisely the vertices of $G$, such that vertices $x$ and $y$ are adjacent in $G$ if, and only if, the lowest common ancestor of $x$ and $y$ in $T(G)$ is a 1-node.

Lerchs [24] proved that the cographs are precisely the graphs obtained from single-vertex graphs by a finite sequence of ⓪ and ① operations defined as follows. Let $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ be arbitrary graphs with $V_1\cap V_2=\emptyset$. Now, set

- $G_1 ⓪ G_2=(V_1\cup V_2, E_1\cup E_2)$;
- $G_1 ① G_2=(V_1\cup V_2, E_1\cup E_2\cup\{xy\mid x\in V_1, y\in V_2\})$.

For the purpose of obtaining a constructive characterization of P4-sparse graphs, Jamison and Olariu [16] introduced a new graph operation defined as follows. Let the graphs $G_1=(V_1,\emptyset)$ and $G_2=(V_2,E_2)$ $(V_1\cap V_2=\emptyset)$ with $V_2=\{v\}\cup K\cup R$ be such that

- $|K|=|V_1|+1\geqslant 2$;
- $K$ is a clique;
- Every vertex in $R$ is adjacent to all vertices in $K$ and non-adjacent to $v$;
- There exists a vertex $v'$ in $K$ such that

$$N_{G_2}(v)=\{v'\} \text{ or } N_{G_2}(v)=K\backslash\{v'\}.$$

Choose a bijection

$$f: V_1\to K\backslash\{v'\}$$

and define

$$G_1 ② G_2=(V_1\cup V_2, E_2\cup E') \tag{1}$$

with

$$E'=\begin{cases}\{xf(x)\mid x\in V_1\} & \text{whenever } N_{G_2}(v)=\{v'\},\\ \{xz\mid x\in V_1,\ z\in K\backslash\{f(x)\}\} & \text{whenever } N_{G_2}(v)=K\backslash\{v'\}.\end{cases}$$

For an illustration of this operation we refer the reader to Fig. 2. Here, the heavy edges are those in $E'$.

The following result shows that the class of P4-sparse graphs is constructible from single-vertex graphs by a finite sequence of operations ⓪, ①, and ②. More precisely, we have

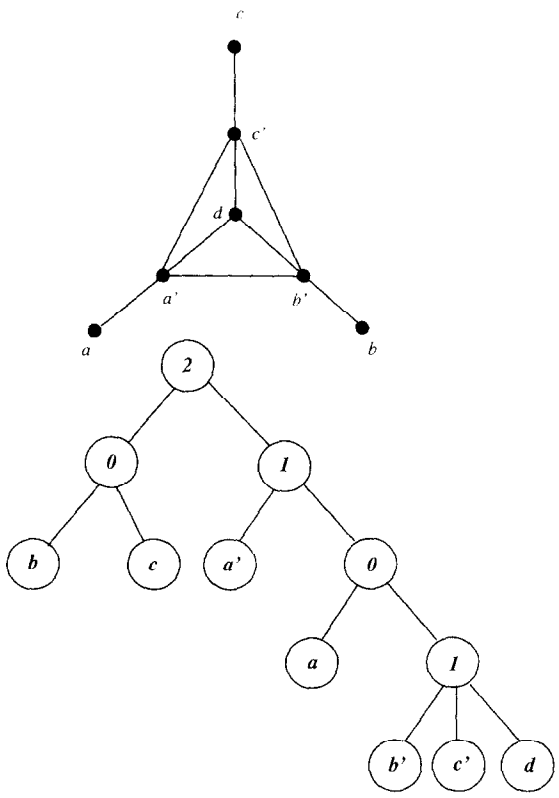


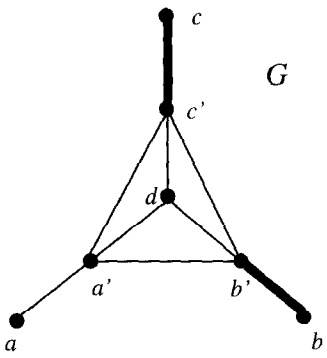

Fig. 1. A P4-sparse graph and the corresponding ps-tree.

**Proposition 2.1** (Jamison and Olariu [16, Theorem 2]). *$G$ is a P4-sparse graph if, and only if, $G$ is obtained from single-vertex graphs by a finite sequence of operations* ⓪, ①, *and* ②.

A nice consequence of Proposition 2.1 is that the P4-sparse graphs have a tree representation unique up to (labeled) tree isomorphism. Given a P4-sparse graph $G=(V,E)$, corresponding tree $T(G)$ will be termed the *ps-tree* of $G$. We refer the reader to Fig. 1 featuring a P4-sparse graph and the corresponding ps-tree.

## 3. A new characterization of P4-sparse graphs

Consider an arbitrary graph $G=(V,E)$. To simplify the notation, a $P_4$ with vertices $a,b,c,d$ and edges $ab,bc,cd$ will be denoted by $abcd$. In this context, the vertices $a$ and $d$ are referred to as *endpoints*, $b$ and $c$ are termed *midpoints*, while the edge $bc$ is termed a *midedge* of this $P_4$.




$G_1=(\{b,c\},\Phi)$

$G_2=(\{a\}\cup\{a',b',c'\}\cup\{d\},\{aa',a'b',a'c',b'c',a'd,b'd,c'd\})$

$v=a$

$v'=a'$

$K=\{a',b',c'\}$

$R=\{d\}$

Fig. 2. Illustrating operation ② on the graph in 2.

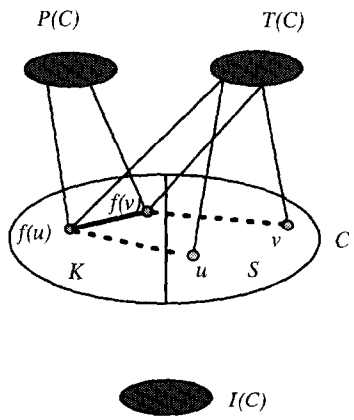



Fig. 3. Illustrating a regular set.

A set $C$ of vertices of $G$ is termed *regular* (for an illustration the reader is referred to Fig. 3) if it admits a partition into non-empty, disjoint sets $K$ and $S$ satisfying the following conditions:

(r1) $|K|=|S|\geqslant 2$, $S$ stable, $K$ a clique.

(r2) Every vertex in $V\backslash C$ belongs to one of the sets:

- $T(C)=\{x \mid x$ adjacent to every vertex in $C\}$;
- $I(C)=\{x \mid x$ adjacent to no vertex in $C\}$;
- $P(C)=\{x \mid x$ adjacent to every vertex in $K$ and to no vertex in $S\}$.

(r3) there exists a bijection $f : S \to K$ such that
- either $N(x) \cap K = \{f(x)\}$ for every $x$ in $S$, or else
- $N(x) \cap K = K \setminus \{f(x)\}$ for every $x$ in $S$.

For later reference we observe here that regular sets are invariant to edge complementation. In other words, a set $C$ is regular in a graph $G$ if and only if it is regular in the complement $\overline{G}$ of $G$. From now on, we shall often denote a regular set $C$ by the tuple $(K, S, f)$, with $K$, $S$, and $f$ as in (r1)–(r3). Additionally, if a regular set $C$ induces a $P_4$ in $G$, we shall refer to the $P_4$ itself as *regular*. As it turns out, both regular sets and regular $P_4$s are key ingredients in our new characterization of P4-sparse graphs as well as in our parallel recognition algorithm. To begin, however, we note that the following characterization of P4-sparse graphs is both well known (see [10, 16]) and, in addition, follows easily from the above definition by a routine argument.

**Proposition 3.1** (Jamison and Olariu [16, Theorem]). *A graph $G$ is P4 -sparse if, and only if, every $P_4$ in $G$ is regular.*

We shall now investigate a number of properties of regular sets in arbitrary graphs, not necessarily P4-sparse graphs. The first such property asserts that regular sets are hereditary in a sense that we are about to make precise.

**Lemma 3.2.** *Let $C = (K, S, f)$ be a regular set in an arbitrary graph $G$, and let $Z$ be a subset of $S$ with $|Z| \leqslant |S| - 2$. Then $C' = C \setminus \{x, f(x) \mid x \in Z\}$ is a regular set in $G$.*

**Proof.** Write $K' \leftarrow K \setminus \{f(x) \mid x \in Z\}$, $S' \leftarrow S \setminus Z$. To see that (r1) is satisfied we note that, since $f$ is a bijection, $|K'| = |S'| = |S \setminus Z| \geqslant 2$, with $K'$ a clique and $S'$ stable.

To see that (r2) is also satisfied, note that every vertex in $T(C)$ belongs to $T(C')$, every vertex in $P(C)$ belongs to $P(C')$, and every vertex in $I(C)$ belongs to $I(C')$. In addition, by virtue of (r3), with $x$ standing for an arbitrary vertex in $Z$, $x \in I(C')$ or $x \in P(C')$ depending on whether or not $N(x) = K \setminus \{f(x)\}$; similarly, $f(x) \in T(C')$ or $f(x) \in P(C')$ depending on whether or not $N(x) = K \setminus \{f(x)\}$.

Finally, to verify that (r3) holds, we only need observe that for every vertex $y$ in $S'$, $f(y)$ belongs to $K'$. This completes the proof of Lemma 3.2. □

**Lemma 3.3.** *Let $C = (K, S, f)$ be a regular set in an arbitrary graph $G$. For every pair of distinct vertices $u, v$ in $S$ (resp. $K$), the unique $P_4$ in $G$ containing both $u$ and $v$ is contained in $C$.*

**Proof.** Write $G = (V, E)$; we claim that

$$\text{if both } u \text{ and } v \text{ belong to } S, \text{ then } U = \{u, v, f(u), f(v)\} \text{ induces the unique } P_4 \text{ in } G \text{ containing both } u \text{ and } v. \tag{2}$$

To justify (2), observe that by (r3), $U$ induces the unique $P_4$ in $C$ containing both $u$ and $v$. If (2) is false, then some set $U' = \{u, v, w, z\}$ induces a $P_4$ in $G$ and contains

vertices from both $C$ and $V\backslash C$. We propose to show that this assumption leads to a contradiction. For this purpose, note that since $U'$ induces a $P_4$, at least one of the vertices $w,z$ distinguishes between $u$ and $v$. Symmetry allows us to assume, without loss of generality, that $wu \in E$ and $wv \notin E$. Note that by (r2), no vertex in $T(C) \cup P(C) \cup I(C)$ distinguishes between $u$ and $v$; it follows that $w \in K$. If $z$ is adjacent to both $u$ and $v$ then $z$ belongs to $T(C)$ or $z$ belongs to $K$. In either case $zw \in E$, contradicting that $U'$ induces a $P_4$. Therefore, $z$ cannot be adjacent to both $u$ and $v$. Since $U'$ induces a $P_4$, some vertex in $U'$ is adjacent to $v$; as we saw, none of $u$ and $w$ is. It follows that $z$ is adjacent to $v$ and, by the above argument, to $w$. But now, $U' \subseteq K \cup S$, a contradiction. Thus, (2) must hold.

Next, we note that if both $u,v$ belong to $K$, then the conclusion follows by (2) along with the noted invariance of regular sets with respect to complementation. This completes the proof of Lemma 3.3. □

Lemma 3.3 implies the following two results that we present next.

**Corollary 3.4.** *Let $C=(K,S,f)$ be a regular set in a graph $G$. If a $P_4$ in $G$ shares precisely two vertices with $C$, then one of them belongs to $S$ and the other one to $K$.*

**Corollary 3.5.** *Let $C=(K,S,f)$ be a regular set in a graph $G=(V,E)$ and let $\pi$ stand for a $P_4$ containing vertices from both $C$ and $V\backslash C$. Then $C$ and $\pi$ share at most two vertices.*

**Proof.** Suppose not; since $\pi$ contains vertices from $V\backslash C$, it must be the case that $C$ and $\pi$ share exactly three vertices. By virtue of (r1), (r2), and the fact that $P_4$s are self-complementary, we may assume that two of these vertices belong to $K$. However, by Lemma 3.3, $\pi$ must be included in $C$, a contradiction. The conclusion follows. □

A regular set $C$ is termed *maximal* if no regular set strictly contains $C$. The following result proposes a characterization of maximal regular sets which is both of an independent interest and an important ingredient in our algorithm.

**Theorem 3.6.** *A regular set $C$ is maximal if, and only if, every regular $P_4$ containing a vertex in $C$ is included in $C$.*

**Proof of Theorem 3.6.** To prove the "if" part, assume that some regular set $C$ is such that every regular $P_4$ containing a vertex in $C$ is included in $C$, yet $C$ is not maximal. In particular, we find vertices $v$, $w$ in $V\backslash C$ such that $C' \leftarrow C \cup \{v,w\}$ is a regular set.

Write $C'=(K',S',f')$ with $K' \leftarrow K \cup \{w\}$, $S' \leftarrow S \cup \{v\}$, and $f'=f \cup \{(v,w)\}$. Let $u$ be an arbitrary vertex in $S$. Since $v \notin C$, $u$ and $v$ are distinct vertices in $S'$. Lemma 3.2 guarantees that $\{u,v,f(u),f(v)\}$ induces a *regular* $P_4$ in $G$; but now we have reached a contradiction: this $P_4$ contains vertices from $C$ and $V\backslash C$.

To prove the "only if" part, let $C=(K,S,f)$ be a maximal regular set in $G$. If the statement is false, then we find a regular $P_4$ $\pi$ containing vertices from both $C$ and

$V\backslash C$. We note that Corollary 3.5 implies that $C$ and $\pi$ share at most two vertices. This observation motivates us to distinguish between the following two cases.

*Case* 1: $C$ and $\pi$ share exactly one vertex. Let $a$ be the unique vertex common to both $C$ and $\pi$. By the invariance of regularity with respect to complementation, we may assume without loss of generality that $a$ belongs to $S$. Let $b$ stand for a vertex in $S$ distinct from $a$. By (r1), $S$ is stable and so $ab \notin E$. Let $x$ stand for a vertex in $\pi$ adjacent to $a$; by the assumption of this case, $x \in V\backslash C$; furthermore, (r2) guarantees that $x \in T(C)$, and so $xb \in E$. Since $b$ is adjacent to $x$ but not to $a$, the regularity of $\pi$ guarantees that $a$ is an endpoint and that $x$ is a midpoint of $\pi$. Let $y$ stand for the midpoint of $\pi$ distinct from $x$. Again, the regularity of $\pi$ implies that $b$ is adjacent to $y$, and, by (r2), $y \in T(C)$. But now, $ya \in E$, contradicting that $\pi$ is a $P_4$. Therefore, Case 1 cannot occur.

*Case* 2: $C$ and $\pi$ share exactly two vertices. By Corollaries 3.4, 3.5 and by the invariance of regular sets under complementation, we can think of these two vertices as being $a$ and $b$ with $a \in S$, $b \in K$, and $ab \in E$. Let $c$ and $d$ be the remaining vertices of $\pi$. By (r2), neither $c$ nor $d$ can be adjacent to $a$ and, consequently, $a$ must be an endpoint of $\pi$. It follows that $c$ or $d$ must be the other endpoint. This allows us to assume, up to change of notation that $\pi$ has edges $ab, bc, cd$. Now, (r2) guarantees that $c \in P(C)$ and that $d \in I(C)$. We claim that

$$C' \leftarrow C \cup \{c, d\} \text{ is a regular set.} \tag{3}$$

To justify (3), note that since $c \in P(C)$ and $d \in I(C)$,

$$K' \leftarrow K \cup \{c\} \text{ is a clique}$$

and

$$S' \leftarrow S \cup \{d\} \text{ is a stable set.}$$

Let $z$ stand for an arbitrary vertex in $V\backslash C'$. If $z \in T(C)$ then $zb, za \in E$; the regularity of $\pi$ implies that $zc, zd \in E$ and so, $z \in T(C')$. Next, if $z \in P(C)$ then we have $zb \in E$ and $za \notin E$. The regularity of $\pi$ guarantees that $zc \in E$ and $zd \notin E$, confirming that $z \in P(C')$. Finally, if $z \in I(C)$, then $z$ is adjacent to neither $a$ nor $b$. Now the regularity of $\pi$ guarantees that $z$ is adjacent to none of $c$ and $y$; consequently, $z \in I(C')$, and the conclusion follows. Thus, (3) must hold, contradicting the maximality of $C$. With this, the proof of Theorem 3.6 is complete. □

The interaction of maximal regular sets and regular $P_4$s described by Theorem 3.6 can be extended to reveal the interaction pattern between two arbitrary maximal regular sets in an arbitrary graph $G$. As it turns out, distinct maximal regular sets cannot overlap. More precisely, we have the following result.

**Theorem 3.7.** *Two maximal regular sets in G coincide whenever they intersect.*

**Proof of Theorem 3.7.** Let $C = (K, S, f)$ and $C' = (K', S', f')$ be distinct maximal regular sets in $G$ such that $C \cap C' \neq \emptyset$. To show that $C$ and $C'$ must coincide, we shall proceed by induction on the size of $G$.

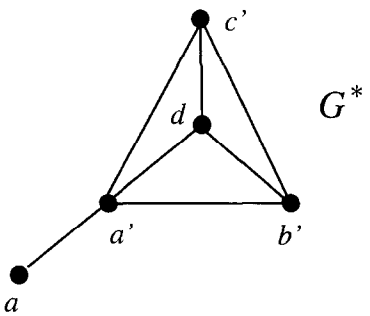



Fig. 4. Illustrating the graph $G^*$ corresponding to the graph $G$ in Fig. 2.

If the statement is false, then $(C\backslash C')\cup(C'\backslash C)\neq\emptyset$. Symmetry allows us to assume, without loss of generality that $C\backslash C'\neq\emptyset$. We claim that

$$\text{for every vertex } x \text{ in } C\backslash C',\ f(x) \text{ or } f^{-1}(x) \text{ belongs to } C\cap C' \text{ depending on whether } x\in S \text{ or } x\in K. \tag{4}$$

Let $x$ be a counterexample to (4) and let $C''$ stand for

$$C\backslash\{x,f(x)\}=(K\backslash\{f(x)\})\cup(S\backslash\{x\})$$

in case $x\in S$, and for

$$C\backslash\{x,f^{-1}(x)\}=(K\backslash\{x\})\cup(S\backslash\{f^{-1}(x)\})$$

in case $x\in K$.

By Lemma 3.2, $C''$ is a regular set in $G$; in fact, since $C$ is a maximal regular set in $G$, it follows instantly that with $G'$ standing for $G\backslash\{x,f(x)\}$ or for $G\backslash\{x,f^{-1}(x)\}$, depending on whether or not $N(x)\cap K=\{f(x)\}$, $C''$ is a maximal regular set in $G'$. We note that $C'$ and $C''$ intersect: this follows trivially by the assumption that $C\cap C'\neq\emptyset$ together with the fact that $x$ is a counterexample. By the induction hypothesis, $C'$ and $C''$ coincide in $G'$. However, this guarantees that, in $G$, $C'=C''\subset C$, contradicting the maximality of $C'$. Thus, (4) must hold.

Now (4) guarantees that for every $x$ in $C\backslash C'$, $f(x)$ or $f^{-1}(x)$ belongs to $C\cap C'$. Notice that for an arbitrary vertex $y$ in $C$ with $x\neq f(y)$ and $y\neq f(x)$, $\{x,y,f(x),f(y)\}$ induces a regular $P_4$ $\pi$ in $G$. Visibly, $\pi$ and $C'$ have at least one vertex in common (namely, $f(x)$ or $f^{-1}(x)$); now Theorem 3.6 guarantees that $x\in C'$, contradicting the assumption that $x\in C\backslash C'$.

With this, the proof of Theorem 3.7 is complete. □

Let $G$ be an arbitrary graph. The graph $G^*$ obtained from $G$ by removing in every maximal regular set $C=(K,S,f)$ all the vertices in $S$ except for an arbitrary one will be referred to as the *reduced* graph of $G$. For an illustration the reader is referred to Fig. 4. The uniqueness of $G^*$ implicit in the definition is justified by the following result.

**Lemma 3.8.** *For every graph $G$, the reduced graph $G^*$ is unique up to isomorphism.*

**Proof.** We shall proceed by induction on the number of maximal regular sets in $G$. Trivially, if $G$ contains no such set, then $G$ and $G^*$ coincide, and there is nothing to prove. Let, therefore, $C=(K,S,f)$ be a maximal regular set in $G$. By the induction hypothesis, the reduced graph $G'^*$ corresponding to $G'=G\backslash S$ is unique up to isomorphism. To complete the proof of Lemma 3.8, we only need observe that we obtain $G^*$ from $G'^*$ by adding an arbitrary vertex from $S$. Now the conclusion follows from (r2) and (r3) combined. □

Let $C=(K,S,f)$ be a maximal regular set in a graph $G$. Note that Lemma 3.8 implies that for the purpose of constructing the reduced graph $G^*$, the choice of the unique vertex in $S$ that belongs to $G^*$ is immaterial. We shall exploit this freedom later, without mentioning it again. We are now in a position to propose a new characterization of P4-sparse graphs in terms of their reduced graphs.

**Theorem 3.9.** *For an arbitrary graph $G$ the following statements are equivalent:*
(i) *$G$ is P4 -sparse;*
(ii) *the reduced graph $G^*$ is a cograph.*

**Proof of Theorem 3.9.** Let $G=(V,E)$ be an arbitrary graph. To settle the implication (i)→(ii), we note that if $G$ is a P4-sparse graph, then by Proposition 3.1 every $P_4$ in $G$ is regular; by Theorem 3.6 and Theorem 3.7, combined, every $P_4$ in $G$ belongs to a unique maximal regular set. Consequently, the reduced graph $G^*$ is a cograph, as claimed.

To prove the implication (ii)→(i), we shall rely on the following intermediate result.

**Lemma 3.10.** *Let $C=(K,S,f)$ be a maximal regular set in $G=(V,E)$. If some vertex in $S$ belongs to a $P_4$ containing vertices from both $C$ and $V\backslash C$, then every vertex in $S$ belongs to such a $P_4$; furthermore, these $P_4$s involve the same vertices in $V\backslash C$.*

**Proof.** Let $u$ be a vertex in $S$ belonging to a $P_4$ induced by the set $X=\{u,x,y,z\}$, containing vertices from both $C$ and $V\backslash S$. By Lemma 3.3, none of the vertices $x,y,z$ belongs to $S$; we claim that

$$\text{exactly one of the vertices } x,y,z \text{ belongs to } C. \tag{5}$$

[To justify (5), note that Corollary 3.5, implies that at most one of the vertices $x,y,z$ belongs to $C$. Further, we only need to show that at least one of the vertices $x,y,z$ belongs to $C$. Otherwise, since by (r3) the vertices $u$ and $v$ have the same adjacencies in $V\backslash C$, it follows instantly that $\{v,x,y,z\}$ induces a $P_4$ in G, and we are done. Thus, (5) must hold, as claimed.]

Let $v$ be an arbitrary vertex in $S\backslash\{u\}$. We propose to show that $v$ belongs to some $P_4$ with vertices from both $C$ and $V\backslash S$, featuring the same vertices from $V\backslash C$ as the

$P_4$ induced by $X$. To this end, recall that by Lemma 3.2, $\{u, v, f(u), f(v)\}$ induces a regular $P_4$ $\pi$ in $G$. To simplify the notation, we assume that $\pi$ features the edges $uw$, $wt$, $tv$ with $\{w, t\} = \{f(u), f(v)\}$.

By (5), we may assume, without loss of generality that

$z$ belongs to $C$ and $x, y$, belong to $V \backslash C$.

Recall that, as noted before, $z$ and $v$ are distinct.

- If $z = w$, then by (r2) $N(u) \cap (V \backslash C) \subset N(w) \cap (V \backslash C)$, and so, one of the vertices $x, y$ must belong to $P(C)$ and the other to $I(C)$. Symmetry allows us to assume that $x \in P(C)$ and $y \in I(C)$. But now, $\{v, t, x, y\}$ induces a $P_4$ in $G$ with edges $vt$, $tx$, $xy$;
- If $z = t$ then, obviously, $uz \notin E$; symmetry allows us to assume that $ux \in E$. Clearly, (r2) guarantees that $x \in T(C)$. This, in turn, implies that $y \in P(C)$. But now, $\{v, w, x, y\}$ induces a $P_4$ in $G$ with edges $vx$, $xw$, $wy$, and the proof of Lemma 3.10 is complete. □

We now return to the proof of Theorem 3.9. Suppose that the statement is false: $G^*$ is a cograph, yet G is not P4-sparse. We find a maximal regular set $C = (K, S, f)$, a vertex $w$ in $S$ but not in $G^*$, and a special $P_4$ $\pi_w$ containing $w$. By definition, we find a unique vertex $u$ in $S$ which belongs to $G^*$. By Lemma 3.10, $u$ is contained in a special $P_4$ $\pi$ involving the same vertices in $V \backslash C$ as $\pi_w$.

Since $u$ belongs to $G^*$ and yet, by assumption, $G^*$ is a cograph, at least one of the vertices of $\pi$ is removed in the process of going from $G$ to $G^*$: let $w'$ be an arbitrary such vertex. Trivially, there exists a maximal regular set $C' = K' \cup S'$ with $w' \in S'$ (note that Theorem 3.7 guarantees that $C$ and $C'$ are vertex-disjoint).

Let $u'$ stand for the unique vertex in $S'$ that also belongs to $G^*$. We propose to show that

there exists a $P_4$ in $G$ containing $u$ and $u'$ but not $w'$. (6)

[To justify (6), note that Lemma 3.10 guarantees that for a suitably chosen $P_4$ $\pi'$ containing $u', \pi$ and $\pi'$ share all the vertices in $V \backslash S'$. In particular, both $u$ and $u'$ belong to $\pi'$. Thus, (6) must hold.]

Since $w'$ was an arbitrary vertex in $\pi$, (6) guarantees that $G^*$ contains a $P_4$, a contradiction. This completes the proof of Theorem 3.9. □

Theorem 3.9 suggests a simple algorithm for recognizing P4-sparse graphs that we outline below. We assume that an arbitrary graph $G$ is input to the algorithm.

---

**Algorithm** Recognize($G$);
**Step 1.** Find all maximal regular sets in $G$;
**Step 2.** Compute $G^*$ by removing in every maximal regular set $C = (K, S, f)$, all vertices in $S$, except for an arbitrary one;
**Step 3.** if $G^*$ is not a cograph then return("no");
**Step 4.** return("yes").

---

## 4. The recognition algorithm

In the remainder of this paper we shall focus on demonstrating that this simple algorithm can be implemented efficiently in parallel.

To begin, we assume that each processor can perform standard arithmetic and boolean operations in one time unit, and can read from and write into shared memory. For convenience, we assume that the processors also have a small amount of local memory. As stated before, in the EREW model of computation, simultaneous read operations from the same memory cell as well as simultaneous write operations into the same memory cell are disallowed. At any moment in time, a processor is either idle (masked out) or executes the same instruction as the other active processors.

To make our exposition more transparent, we shall present, first, a number of basic assumptions related to the data structures used throughout the remainder of this work. An arbitrary graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ is assumed as input to our recognition algorithm. As usual, the graph $G$ is represented by its adjacency list; moreover, with every entry in this adjacency lists we associate a processor, for a total of $O(m)$ processors.

In addition, we shall enumerate the vertices and the edges of $G$, in an arbitrary way, as

$$v_1, v_2, \ldots, v_n \tag{7}$$

and

$$e_1, e_2, \ldots, e_m, \tag{8}$$

respectively.

We shall find it convenient to represent sets of vertices of $G$ by their characteristic vector; specifically, for a set $S \subset V$, this is an $n$-bit vector $(z_1, z_2, \ldots, z_n)$ such that, for all $i=1,2,\ldots,n$, $z_i=1$ if $v_i \in S$, and 0 otherwise. In this representation, the cardinality $|S|$ of a set $S$ can be computed in $O(\log n)$ EREW time using $O(n/\log n)$ processors in the obvious way. Similarly, given sets $S$, $S' \subset V$, the task of computing $S \backslash S'$ can be performed by the same technique in $O(\log n)$ time using $O(n/\log n)$ processors.

For a vertex $x$ of $G$ we compute the set $N[x]$ in the following two stages:

- in $O(\log n)$ time using $O(n/\log n)$ processors, initialize $N[x]$ to $\emptyset$, and then
- in $O(1)$ time the $d_G(x)$ processors associated with the adjacency list of $x$ set to 1 the corresponding bit of $N[x]$.

Consequently, to compute all the sets $N[x]$ we need $O(\log n)$ time and $O(n^2/\log n + m) \subset O((n^2 + mn)/\log n)$ processors.

For later reference, we shall associate with every edge $e_i=\{v,w\}$, $1 + \lceil n/\log n \rceil$ processors, referred to as $P(e_i,0), P(e_i,1), \ldots, P(e_i, \lceil n/\log n \rceil)$. Here, $P(e_i,0)$ is used for both computational and bookkeeping purposes, as we are about to describe, while $P(e_i,1), \ldots, P(e_i, \lceil n/\log n \rceil)$ are used for computational tasks only. It is easy to see that, altogether, $O(mn/\log n)$ processors are associated with the edges of $G$.

At the same time, for every edge $e_i = \{v, w\}$ of $G$, we compute the following sets:

- $N_{vw} = N[v] \backslash N[w]$, and
- $N_{wv} = N[w] \backslash N[v]$.

The motivation for computing these sets is provided by the following simple result.

**Lemma 4.1.** *An edge $e_i = \{v, w\}$ is the midedge of a regular $P_4$ in $G$ only if $|N_{vw}| = |N_{wv}| = 1$ and $uz \notin E$, with $u, z$ standing for the unique vertex in $N_{vw}$ and $N_{wv}$, respectively.*

**Proof.** To see that this is the case, let $uvwz$ stand for a regular $P_4$ having $vw$ as a midedge. Trivially, $u \in N_{vw}$, while $z \in N_{wv}$. Note that every vertex in $N_{vw} \backslash \{u\}$ distinguishes between $v$ and $w$ and, therefore, the assumption that $uvwz$ is a regular $P_4$ implies $N_{vw} = \{u\}$; similarly, $N_{wv} = \{z\}$. □

Next, we claim that for every edge $e_i = \{v, w\}$ of $G$, the sets $N_{vw} = N[v] \backslash N[w]$ and $N_{wv} = N[w] \backslash N[v]$ can be computed in $O(\log n)$ time using $O(n/\log n)$ processors. This is trivial once the $O(n/\log n)$ processors associated with $e_i$ know $N[v]$ and $N[w]$. Since no read conflicts are allowed, for every vertex $v$ of $G$, $N[v]$ will have to be broadcast to all the $d_G(v)$ edges incident with $v$. To restrict the running time to $O(\log n)$, we use $\lceil d_G(v)/\log n \rceil$ "superprocessors" for every vertex $v$ of $G$: each of these "superprocessors" can be thought of as a set of $n$ processors which will be used to transfer an $n$-bit vector in $O(1)$ time. With this trick, to broadcast $N[v]$ to the $d_G(v)$ edges incident with $v$ we do the following:

- in $O(\log \lceil d_G(v)/\log n \rceil) \subseteq O(\log n)$ time $N[v]$ will be broadcast to the $\lceil d_G(v)/\log n \rceil$ superprocessors (equivalently, to $\lceil d_G(v)/\log n \rceil$ edges incident with $v$);
- each of the $\lceil \frac{d_G(v)}{\log n} \rceil$ superprocessors will broadcast the value of $N[v]$, sequentially, to $\log n$ edges in $O(\log n)$ time.

Visibly, the broadcast operation takes $O(\log n)$ time altogether using

$$O\left(\frac{n}{\log n} \sum_{v \in V} d_G(v)\right) = O\left(\frac{mn}{\log n}\right)$$

of the processors available.

As it turns out, there is no need to compute the set of all the regular $P_4$s in $G$ explicitly. Instead, it will be stored by the subset of all the flagged processors $P(e_i, 0)$. The details are spelled out in the following procedure.

---

**procedure** Find_Regular_P4s($G$);

0. **begin**
1. **for** every edge $e_i = \{v, w\}$ of $G$ **pardo**
2. $N_{vw} \leftarrow N[v] \backslash N[w]$;
3. $N_{wv} \leftarrow N[w] \backslash N[v]$;
4. **if** $|N_{vw}| = |N_{wv}| = 1$ **then** {let $N_{vw} = \{u\}$, $N_{wv} = \{z\}$, $U = \{u, v, w, z\}$}

5. **if** $uz \notin E$ **then begin**
6. **for** all vertices $x$ in $V\backslash\{u,v,w,z\}$ **pardo**
7. **if** $x \notin T(U) \cup P(U) \cup I(U)$ **then**
8. some processor $P(e_i,t)$ $(t \neq 0)$ marks itself;
9. **if** no processor is marked **then** $P(e_i,0)$
10. remembers $u,v,w,z$;
11. flags itself
12. **end** {if}
13. **endfor**
14. **end**; {Find_Regular_P4s}

---

**Lemma 4.2.** *Procedure Find_Regular_*P4s *correctly computes the set of all the regular* $P_4$s *in* $G$ *in* $O(\log n)$ *EREW time using* $O((n^2+mn)/\log n)$ *processors.*

**Proof.** The correctness of the procedure follows directly from the definition of regular $P_4$s together with Lemma 4.1.

To argue for the complexity, we note that computing all the sets $N[x]$ for $x \in V$ requires $O(\log n)$ time using $O(n^2/\log n)$ processors. Further, recall that the broadcasting required for the purpose of the computation in lines 2–3 takes $O(\log n)$ time and $O(nm/\log n)$ processors. We note, further, that in line 7 we do not compute the sets $T(U)$, $P(U)$, and $I(U)$. Instead, for every $x$ in $V\backslash U$ we verify the following conditions:

- $x$ is adjacent to all of $u,v,w,z$, or else
- $x$ is adjacent to $v$ and $w$ and non-adjacent to $u$ and $z$, or else
- $x$ is non-adjacent to all of $u,v,w,z$.

For every edge $e_i = \{v,w\}$ the processors $P(e_i,1), P(e_i,2), \ldots, P(e_i, \lceil (n-4)/\log n \rceil)$ are assigned to check the conditions above: more precisely, every processor $P(e_i,j)(1 \leqslant j \leqslant \lceil (n-4)/\log n \rceil)$ verifies, roughly, $O(\log n)$ vertices in $V\backslash U$ sequentially. Since every vertex can be checked in constant time, line 7 takes $O(\log n)$ time and uses only processors that have been assigned already (i.e. no extra processors are needed). Finally, line 9 requires broadcasting (since no concurrent write is allowed). This can be performed in $O(\log\lceil (n-4)/\log n \rceil) = O(\log n)$ time using $\lceil (n-4)/\log n \rceil$ processors for every edge $e_i$. Therefore, the running time of Find_Regular_P4s is bounded by $O(\log n)$ using $O((n^2+mn)/\log n)$ processors in the EREW model, as claimed. □

From now on, every edge $e_j$ releases $\lceil n/\log n \rceil$ of its allocated processors which, thus, become available to perform other tasks, as we are about to explain. We shall assume, without loss of generality, that every edge retains processor $P(e_j,0)$ which will be referred to, simply, as $P(j)$.

For later reference we need to introduce some new terminology; in every *regular* $P_4$ $uvwz$ with endpoints $u = v_j$ and $z = v_k$ and $j < k$, we shall refer to $u$ as the *local winner* and to $z$ as the *local loser*. For the purpose of constructing the tree representation of $G$, should $G$ turn out to be a P4-sparse graph, we need to record relevant information about local losers and winners.

Every flagged processor P($i$) writes the identity of the local loser into $A[i]$, and the local winner into $B[i]$: here $A$, $B$ are one-dimensional arrays of $m$ elements each, initialized to 0. (We note that initializing $A$ and $B$ to 0 takes O(1) time using the processors P($i$) ($1 \leqslant i \leqslant m$).)

Furthermore, using an optimal sorting algorithm, we can sort all non-zero entries in $A$ and $B$ and eliminate all duplicates in $O(\log m) = O(\log n)$ EREW time using $O(m)$ processors. We assume that at the end of the sorting stage, $A[1], A[2], \ldots, A[k]$ contains the set of all the local losers with no duplicates; similarly, $B[1], B[2], \ldots, B[l]$ contains the set of all the local winners with no duplicates. For the purpose of constructing the reduced graph $G^*$ corresponding to $G$, we shall find it convenient to compress the information in $A$ into an $n$-bit vector $L$: bit $i$ of $L$ is set to one if, and only if, vertex $v_i$ is a local loser. Note that once $A$ is sorted with all duplicates removed, constructing the bit-vector $L$ takes $O(\log n)$ time and, at most, $O(n/\log n)$ processors.

An endpoint $u$ of a regular $P_4$ will be called a *global winner* if the bit corresponding to $u$ is set to 0 in $L$: this terminology is motivated by the observation that a local winner may turn out to be a local loser in a different regular $P_4$. For the purpose of recording the set of all the global winners, we introduce a bit-vector $W$ that we initialize in the following way: set the $i$-th bit of $W$ to 1 if $v_i$ is a local winner. Note that, once the information in the array $B$ is available with no duplicates, obtaining $W$ from $B$ can be done trivially in $O(\log n)$ time using $O(n/\log n)$ processors. Next, the assignment

$$W \leftarrow W \backslash L$$

yields the characteristic vector of the set of all global winners.

Once the characteristic vector $W$ of all the global winners is available, every flagged processor P($i$) finds out whether its local winner is also a global winner. If this is the case, then P($i$) will be referred to as *essential*. The details are spelled out in the following procedure.

---

**procedure** Find_Winners_and_Losers($G$);
0. **begin**
1. $A[1 : m] \leftarrow B[1 : m] \leftarrow 0$;
2. $L \leftarrow W \leftarrow 0$;
3. **for** every flagged processor P($i$) **pardo**
4. $A[i] \leftarrow$ local loser corresponding to $e_i$;
5. $B[i] \leftarrow$ local winner corresponding to $e_i$;
6. **endfor;**
7. let $A[1], A[2], \ldots, A[k]$ be the non-zero entries of $A$ in sorted order with all duplicates removed;
8. let $B[1], B[2], \ldots, B[l]$ be the non-zero entries of $B$ in sorted order with all duplicates removed;
9. **for** all $i \leftarrow 1$ **to** $k$ **pardo**

```
10.    set the A[i]-th bit of L to 1;
11.  for all i ← 1 to l pardo
12.    set the B[i]-th bit of W to 1;
13.  W ← W\L; {find global winners}
14.  broadcast W to all the processors P(i);
15.  for every flagged processor P(i) pardo
16.    if the local winner of e_i is in W then
17.      P(i) does the following:
18.        remembers that its local winner is a global winner;
19.        marks itself as "essential"
20.  return(L, W)
21. end; {Find_Winners_and_Losers}
```

---

**Lemma 4.3.** *Procedure Find_Winners_and_Losers correctly computes the set of all global winners and losers in* $O(\log n)$ *EREW time using* $O(mn/\log n)$ *processors.*

**Proof.** To address the correctness, note that by virtue of Theorem 3.7, two maximal regular sets are either disjoint or else coincide. By Theorem 3.6, every $P_4$ that shares vertices with some maximal regular set is contained in that regular set. Consequently, our strategy of finding losers guarantees that in every maximal regular set $C=(K,S,f)$ *exactly* one vertex in $S$ is a global winner, namely the one that comes first in the order $v_1, v_2, \ldots, v_n$ that we assumed.

To address the complexity note that, by the previous discussion, lines 1–6 take $O(1)$ time and $O(\max\{m,n\})$ processors. Similarly, lines 7 and 8 take $O(\log m)=O(\log n)$ EREW time using $O(m)$ processors by using any optimal sorting algorithm [12]. Lines 9–12 also take $O(1)$ time and require $O(n)$ processors; line 13 takes $O(\log n)$ time and $O(n/\log n)$ processors. To broadcast the bit vector $W$ to all the $m$ processors $P(i)$, we use the trick described at the end of Lemma 4.1.

Specifically, we use $\lceil m/\log n\rceil$ superprocessors to broadcast $W$; as already mentioned, each of these superprocessors can be thought of as a set of $n$ processors which will be used to transfer an $n$-bit vector in $O(1)$ time. With this trick, the task of broadcast $W$ to the $m$ processors $P(i)$ involves the following:

- in $O(\log\lceil m/\log n\rceil)\subseteq O(\log n)$ time $W$ will be broadcast to the $\lceil m/\log n\rceil$ superprocessors;
- each of the $\lceil m/\log n\rceil$ superprocessors will broadcast the value of $W$, sequentially, to $\log n$ other processors in $O(\log n)$ time.

The reader should have no difficulty confirming that the broadcast operation detailed above takes $O(\log n)$ time altogether using $O(mn/\log n)$ of the processors. The conclusion follows. □

In addition, our arguments about recognizing P4-sparse graphs rely, in part, on the following result concerning the recognition of cographs.

**Proposition 4.4** (Lin and Olariu [25, Theorem 4]). *For an arbitrary graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ as input, membership in the class of cographs can be detected in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors in the EREW model.*

We are now in a position to show how the different pieces fit together in our recognition algorithm for P4-sparse graphs.

---

**procedure** Recognize_P4sparse($G$);
{Input: an arbitrary graph $G=(V,E)$ with $|V|=n$ and $|E|=m$;
Output: "yes" or "no" depending on whether or not $G$ is P4-sparse;}
0. **begin**
1. Find_Regular_P4s($G$);
2. Find_Winners_and_Losers($G$);
3. using the information contained in $L$ construct the graph $G^*$;
4. **if** Cograph($G^*$) **then**
5. return("yes");
6. return("no")
7. **end**; {Recognize_P4sparse}

---

**Theorem 4.5.** *Procedure Recognize_P4sparse correctly determines whether an arbitrary graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ is a P4-sparse graph in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors in the EREW-PRAM model.*

**Proof of Theorem 4.5.** The correctness follows directly from Lemmas 4.1–4.3 and Theorem 3.9, combined.

To argue for the complexity, note that line 1 runs in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors. The test in line 2 takes, by virtue of Lemma 4.3, $O(\log n)$ time, using $O(mn/\log n) \subseteq O((n^2+mn)/\log n)$ processors.

Next, constructing $G^*$ in easy, once we know $L$; finally, for the purpose of performing the test in line 4 efficiently, we can use the cograph recognition algorithm in [25], running in $O(\log n)$ EREW time and using $O((n^2+mn)/\log n)$ processors.

Altogether, therefore, the entire procedure takes $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors, as claimed. With this the proof of Theorem 4.5 is complete. □

## 5. Constructing the tree representation for P4-sparse graphs

For convenience, we shall inherit the entire context and data structures of the previous sections. It is worth noting that an important byproduct of the cograph recognition

algorithm in [25] is that, upon successful recognition, the corresponding cotree is also constructed. This implies, in particular, that when our recognition algorithm for P4-sparse graphs terminates with a "yes" answer, the cotree $T(G)$ of the reduced (co)graph $G^*$ of $G$ is also constructed.

To make our exposition of the tree-constructing algorithm for P4-sparse graphs more transparent and easier to follow, we shall enumerate the maximal regular sets in $G$, arbitrarily, as $C_1=(K_1,S_1,f_1), C_2=(K_2,S_2,f_2),\ldots,C_p=(K_p,S_p,f_p)$ for some $p\geqslant 0$. We note that if $G$ and $G^*$ coincide then there are no (maximal) regular sets in $G$ and $p=0$. It is also useful to note that at the end of the (successful) recognition of a P4-sparse graph $G$, the relevant information about the graph is stored by the tuple $(T(G),SK(G))$: here, $T(G)$ is the cotree associated with $G^*$; $SK(G)$ is a structure that we are about to describe.

We can think of $SK(G)$ as a one-dimensional array, with $SK[i]$ $(1\leqslant i\leqslant p\neq 0)$ containing the following information:

- characteristic vectors of $K_i$ and $S_i$;
- the identity of the unique vertex $w_i$ in $S_i$ that belongs to $G^*$;
- the identity of $f_i(w_i)$;
- $r_i=|K_i|=|S_i|$.

For algorithmic purposes it is convenient (as is done in [25]) to represent $T(G)$ by parent pointers, that is, every node in $T(G)$ points to its unique parent, with the root of $T(G)$ pointing to itself. We also assume that, for every vertex $v$ in $G$, a pointer is maintained to the location of $v$ in $T(G)$ or $SK(G)$, as the case may be.

For simplicity, we shall assume that the global winners are $w_1,w_2,\ldots,w_p$ (trivially, the identity of these vertices is available instantly from $W$). Recall that every essential processor is aware of the identity of its local winner, say, $w_j$.

Now computing $S_i$ for all $i$ $(1\leqslant i\leqslant p)$ is easy: after having initialized $S_i$ by setting to 1 the bit corresponding to $w_i$, every essential processor whose local winner is $w_j$, sets the $j$th bit of $S_i$, with $v_j$ standing for its local loser.

To compute $K_i$ we proceed along similar lines: in $O(\log m)=O(\log n)$ time we identify, for every $i$ $(1\leqslant i\leqslant p)$, the subset $P(i1), P(i2),\ldots,P(it_i)$ of the essential processors whose local winner is $v_{W[i]}$. Note that this ordering can be readily computed in $O(\log n)$ time and $O(n)$ processors, by using an optimal sorting algorithm. After this, processor $P(i1)$ broadcasts to $P(i2),\ldots,P(it_i)$ the identity of the midpoints of the regular $P_4$ that is remembered in line 10 of procedure Find_Regular_$P4$s. Every processor $P(ij)$ $(2\leqslant j\leqslant t_i)$ marks its own midpoint coinciding with one received by broadcasting. Finally, every processor $P(i1), P(i2),\ldots,P(it_i)$ sets to 1 the bit of $K_i$ corresponding to the unmarked midpoint it stores. Note that this operation leads to no write conflicts.

Now computing $f_i(w_i)$ is easy: if $|N(w_i)\cap K_i|=1$ then $f_i(w_i)$ is precisely the unique vertex in $N(w_i)\cap K_i$; otherwise, $f_i(w_i)$ is the unique vertex in $K_i\backslash N(w_i)$.

Finally, the value of $r_i=|S_i|=|K_i|$ can be computed, for every $i$ $(1\leqslant i\leqslant p)$ in $O(\log n)$ time using $O(n/\log n)$ processors. We present the details in the following procedure.

**procedure** Construct_SK($G$);
0. **begin**
1. let $w_1, w_2, \ldots, w_p$ stand for the global winners;
2. **for** $i \leftarrow 1$ **to** $p$ **pardo**
3. **if** processor P($i$) is essential **then begin**
4. processor P($i$) sets to 1 the bit of $S_i$ corresponding to $w_i$;
5. let P($i1$), P($i2$),..., P($it_i$) ($1 \leqslant i \leqslant p$) be the essential processors whose local winner is $w_i$;
6. **for** $j \leftarrow 1$ **to** $t_i$ **pardo**
7. processor P($ij$) sets the $k$th bit of $S_i$ with $v_k$ standing for its local loser;
8. processor P($i1$) broadcasts to P($i2$),..., P($it_i$) the identity of the two midpoints it stores;
9. **for** $j \leftarrow 2$ **to** $t_i$ **pardo**
10. processor P($ij$) marks the midpoint it stores coinciding with one of the midpoints received;
11. **for** $j \leftarrow 1$ **to** $t_i$ **pardo**
12. processor P($ij$) sets to 1 the bit of $K_i$ corresponding to its unmarked midpoint;
13. $r_i \leftarrow |K_i| = |S_i|$;
14. **if** $|N(w_i) \cap K_i| = 1$ **then**
15. $f_i(w_i) \leftarrow$ the unique vertex in $N(w_i) \cap K_i$
16. **else**
17. $f_i(w_i) \leftarrow$ the unique vertex in $K_i \backslash N(w_i)$
18. **endif**;
19. return($SK(G)$)
20. **begin**; {Construct_SK}

To summarize our previous discussion, we state the following result.

**Lemma 5.1.** *Procedure Construct_SK correctly computes the information in every* $SK[i]$, $(1 \leqslant i \leqslant p)$, *in* $O(\log n)$ *time using* $O(n^2/\log n)$ *processors in the EREW-PRAM model.*

**Proof.** The correctness of procedure Construct_SK follows from Theorem 3.6 and Theorem 3.7. To argue about the complexity, we note, as before, that line 5 takes $O(\log n)$ time and $O(m/\log n)$ processors; to broadcast in line 8, we spend $O(\log n)$ time using $O(t_i/\log n)$ processors for every $i$ $(1 \leqslant i \leqslant p)$. Line 14 assumes that $N(w_i) \cap K_i$ has been computed; visibly, this takes $O(\log n)$ time and $O(n/\log n)$ processors for every $i$. Since $p$ is at most $n$ the conclusion follows. □

We now address the problem of efficiently constructing the ps-tree representation of $G$. Our arguments rely heavily on the following results.

**Proposition 5.2** (Jamison and Olariu [17, Theorem 4]). *For each $i$ $(1 \leqslant i \leqslant p)$, there exist a unique 0-node $\lambda(i)$ and a 1-node $\lambda'(i)$ in $T(G)$ such that, setting $z = w_i$, for any $v, w \in K_i$ with $zv \notin E$, $zw \in E$, the following are satisfied:*

$$\lambda(i) = p(z); \quad \lambda'(i) = p(w); \quad \lambda'(i) = p(\lambda(i)).$$

*Furthermore,*

$$\text{either } \lambda(i) = p(v) \text{ or else } \lambda''(i) = p(v) \text{ with } \lambda(i) = p(\lambda''(i)).$$

To construct the tree representation of a P4-sparse graph $G$, we need a way of incorporating the local users into the tree structure. For this purpose, a new type of node is needed; this is the 2-node which has precisely two children: a 0-node and a 1-node. Obviously, the 2-node corresponds to the ② defined in (1). The details of this tree construction can be found in [17].

We shall also rely on the following result which guarantees the correctness of the construction.

**Proposition 5.3** (Jamison and Olariu [17, Theorem 5]). *With a P4-sparse graph $G$ as input, the tree $T(G)$ can be computed in time linear in the size of $G$.*

As it turns out, procedure Build_ps_Tree presented in [16] can be easily parallelized, in such a way that the computation can be carried out in the EREW model. We present the parallel version of Build_ps_Tree next.

---

**procedure** Parallel_Build_ps_Tree($G$);
{Input: a $P4$-sparse graph represented as $(T(G), SK(G))$
Output: the corresponding ps-tree $T(G)$, rooted at $R$;}
0. **begin**
1. **for** every essential processor P($i$) **pardo**
2. create a 2-node $\beta$;
3. create a 1-node $\gamma$;
4. add $\gamma$ as a child of $\beta$;
5. add $\lambda$ as a child of $\gamma$;
6. **if** $r_i = 2$ **then begin**
7. add the unique vertex in $S_i \backslash \{w_i\}$ as a child of $\beta$;
8. add $f_i(w_i)$ as a child of $\gamma$
9. **end**
10. **else begin**
11. create a 0-node $\alpha$;
12. add $\alpha$ as a child of $\beta$;
13. add all vertices in $S_i \backslash \{w_i\}$ as children of $\alpha$;

```
14.       if w_i is adjacent to f_i(w_i) then
15.         add f_i(w_i) as a child of γ
16.       else
17.         add all vertices in K_i\{w_i} as children of γ
18.       endif;
19.     if d(λ') ≠ |N(w_i) ∩ K_i| + 1 then
20.       add β as a child of λ'
21.     else begin
22.       add β as a child of p(λ');
23.       delete λ'
24.     endif;
25.   endfor;
26.   if d(R) = 1 then R ← unique child of R;
27.   return(T(G))
28. end; {Parallel_Build_ps_Tree}
```

**Theorem 5.4.** *Procedure Parallel_Build_ps_Tree correctly constructs the ps-tree of a P4-sparse graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ in $O(\log n)$ EREW time using $O(n/\log n)$ processors.*

**Proof of Theorem 5.4.** The correctness follows immediately from Propositions 5.2, 5.3 and Lemma 5.1, combined. We propose to show that the computation inherent in procedure Parallel_Build_Tree can be performed in $O(\log n)$ EREW time by $O(n)$ processors. To justify this claim we note that

- lines 2–12 take constant time to execute and no read or write conflicts can occur;
- line 13 requires broadcasting the address of $\alpha$ to all the vertices in $S_i\backslash\{w_i\}$; trivially, this takes at most $O(\log n)$ time using $O(|S_i|/\log n)$ processors; altogether, therefore, line 13 runs in $O(\log n)$ time using $O(\sum_{i=1}^{p}|S_i|/\log n)=O(n/\log n)$ processors;
- similarly, line 17 requires broadcasting the address of $\gamma$ to all the vertices in $K_i\backslash\{w_i\}$; as before, this takes $O(\log n)$ time using $O(|K_i|/\log n)$ processors; altogether, therefore, line 13 runs in $O(\log n)$ time using $O(\sum_{i=1}^{p}|K_i|/\log n)=O(n/\log n)$ processors;
- to add $\beta$ as a child of $\lambda'$ (resp. $p(\lambda')$ in lines 20 (resp. 22) we only need pick up the address of $\lambda'$ from $\lambda$ (resp. the address of $p(\lambda')$ from $\lambda'$), and so no broadcasting is needed.

This completes the proof of Theorem 5.4. □

A graph $G$ is P4-reducible if every vertex of $G$ belongs to at most one $P_4$ (refer to Fig. 5 for an example). As pointed out in [15], every P4-reducible graph is P4-sparse, but not conversely. It is not hard to see that a P4-sparse graph is P4-reducible if and only if every maximal regular ser $C=(K,S,f)$ has the property that $|k|=|S|=2$. Consequently, Theorems 4.5 and 5.4 have the following consequence. A similar result has been obtained in [13].

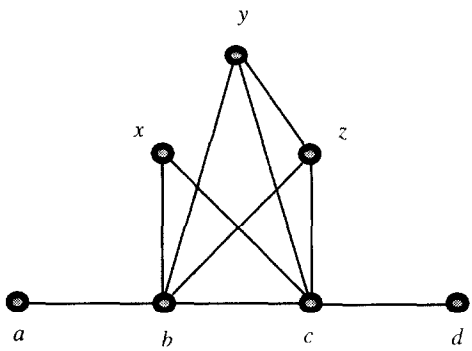



Fig. 5. A *P4*-reducible graph.

**Corollary 5.5.** *The task of recognizing whether a graph* $G=(V,E)$ *with* $|V|=n$ *and* $|E|=m$ *edges is P4-reducible can be performed in* $O(\log n)$ *time using* $O((n^2+mn)/\log n)$ *processors in the EREW-PRAM model. Moreover, should G be P4-reducible the corresponding tree representation can be built in the same complexity.*

## 6. Conclusion and open problems

The class of P4-sparse graphs corresponds naturally to a new local density metric: specifically, we allow chordless paths of length three to occur, provided no set of five vertices induces more than one such path. The class of P4-sparse graphs strictly contains the class of cographs and P4-reducible graphs that correspond to more stringent local density metrics.

Our first major contribution was to provide a novel way of looking at P4-sparse graphs in terms of regular sets. The concept of a regular set is interesting in its own right and may find applications to elucidating the structure of other classes of graphs.

Our second main contribution was to have presented a parallel algorithm to recognize P4-sparse graphs and to construct their unique tree representation. With an arbitrary graph $G=(V,E)$ with $|V|=n$ and $|E|=m$ as input, our algorithm runs in $O(\log n)$ time using $O((n^2+mn)/\log n)$ processors in the EREW-PRAM model of computation. Our algorithm is not cost-optimal. Nonetheless, the method used in this paper may help to develop a cost-optimal parallel recognition algorithm for this class of graphs.

As our structural theorem shows, the major bottleneck in the task of recognizing P4-sparse graphs is cograph recognition. To date, in spite of persistent efforts by several researchers, no cost-optimal recognition algorithm for cographs is known.

Yet another bottleneck in the recognition algorithm that we presented is the computation of $N_{vw}=N(v)-N(w)$ and $N_{wv}=N(w)-N(v)$. The only case of interest is when the sizes of the two neighborhoods differ by one, otherwise this is not a candidate edge for the mid-edge of a $P_4$. Therefore, only $\min\{d(v),d(w)\}$ processors are

needed for the computation. Moreover, it is relatively straightforward to show that $\sum_{vw \in E} \min\{d(v), d(w)\} \in O(m^{3/2})$. In the case of sparse graphs this leads to a better bound. Using this observation it may be possible to reduce the processor bound. This is an interesting area for further investigations.

## Acknowledgements

The authors wish to thank Professor Hammer for his professional way of handling this submission. We would also like to thank four anonymous referees for very detailed comments that resulted to a better presentation of the material. In particular, we are indebted to one of the referees for spotting and pointing out a mistake in our original manuscript, and to another referee for suggesting handling the problem of recognizing P4-reducible graphs as a simple corollary of our algorithm. The second author acknowledges inspiring discussions with Luitpold Babel.

## References

[1] L. Babel, I. Ponomarenko, G. Tinhofer, Directed path graph isomorphism, in: Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG'94, Lecture Notes in Computer Science, vol. 903, Springer, Berlin, 1995, pp. 395–406, to appear.
[2] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications, North-Holland, Amsterdam, 1976.
[3] K.S. Booth, G.S. Lueker, A linear time algorithm for deciding interval graph isomorphism, J. ACM 26 (1979) 183–195.
[4] D.G. Corneil, D.G. Kirkpatrick, Families of recursively defined perfect graphs, Congressus Numerantium 39 (1983) 237–246.
[5] D.G. Corneil, H. Lerchs, L. Stewart Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.
[6] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (1985) 926–934.
[7] E. Dahlhaus, Efficient parallel recognition algorithms for cographs and distance hereditary graphs, Discrete Appl. Math. 57 (1995) 29–45.
[8] X. He, Parallel algorithms for cograph recognition with applications, J. Algorithms 15 (1993) 284–313.
[9] L. Hellerstein, M. Karpinski, Learning read-once formulas using membership queries, Proceedings of the 2nd Annual Workshop on Computational Learning Theory, 1989, pp. 146–161.
[10] C. Hoang, Doctoral Dissertation, McGill University, Montreal, 1985.
[11] W. Hochstättler, G. Tinhofer, Hamiltonicity in graphs with few $P_4$'s, Computing 54 (1995) 213–225.
[12] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1991.
[13] B. Jamison, R. Lin, S. Olariu, A fast parallel recognition algorithm for P4-reducible graphs, Proceedings of the 5th Jerusalem Conference on Information Technology, Jerusalem, Israel, 1990, pp. 402–413.
[14] B. Jamison, S. Olariu, P4-reducible graphs, a class of uniquely tree representable graphs, Stud. Appl. Math. 81 (1989) 79–87.
[15] B. Jamison, S. Olariu, A tree representation for P4-sparse graphs, Discrete Appl. Math. 35 (1992) 115–129.
[16] B. Jamison, S. Olariu, Recognizing P4-sparse graphs in linear time, SIAM J. Comput. 21 (1992) 381–406.
[17] B. Jamison, S. Olariu, A linear-time recognition algorithm for P4-reducible graphs, Theoret. Comput. Sci. 145 (1995) 329–344.
[18] B. Jamison, S. Olariu, P-components and the homogeneous decomposition of graphs, SIAM J. Discrete Math. 8 (1995) 448–463.

[19] B. Jamison, S. Olariu, Optimal algorithms for P4-sparse graphs, Discrete Appl. Math. 61 (1995) 155–175.
[20] H.A. Jung, On a class of posets and the corresponding comparability graphs, J. Combin. Theory Ser. B 24 (1978) 125–133.
[21] M.M. Klawe, D.G. Corneil, A. Proskurowski, Isomorphism testing in hook-up graphs, SIAM J. Algebraic Discrete Methods 3 (1982) 260–274.
[22] E.L. Lawler, Graphical algorithms and their complexity, Math. Center Tracts 81 (1976) 3–32.
[23] H. Lerchs, On cliques and kernels, a manuscript, Dept. of Computer Science, University of Toronto, March 1971.
[24] H. Lerchs, On the clique-kernel structure of graphs, Dept. of Computer Science, University of Toronto, October 1972.
[25] R. Lin, S. Olariu, An NC recognition algorithm for cographs, J. Parallel Distributed Comput. 13 (1991) 76–90.
[26] R. Lin, S. Olariu, An optimal parallel matching algorithm for cographs, J. Parallel Distributed Comput. 22 (1994) 26–37.
[27] P.K. McKinley, J.W.S. Liu, Multicast tree construction in bus-based networks, Comm. ACM 33 (1990) 29–42.
[28] D. Seinsche, On a property of the class of n-colorable graphs, J. Combin. Theory Ser. B 16 (1974) 191–193.
[29] L. Stewart, Cographs, a class of tree representable graphs, M.Sc. Thesis, TR 126/78, Dept. of Computer Science, University of Toronto, 1978.
[30] D.P. Sumner, Dacey graphs, J. Australian Math. Soc. 18 (1974) 492–502.
[31] A. Zomaya (Ed.), Handbook of Parallel and Distributed Computing, McGraw-Hill, New York, 1995.