1995

A Linear-Time Recognition Algorithm for P4-Reducible Graphs

B. Jamison
Old Dominion University

S. Olariu
Old Dominion University

Theoretical Computer Science 145 (1995) 329–344

Theoretical Computer Science

Note

A linear-time recognition algorithm for P_4-reducible graphs[☆]

B. Jamison, S. Olariu*

Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162, USA

Received June 1992; revised December 1994
Communicated by M. Nivat

Abstract

The P_4-reducible graphs are a natural generalization of the well-known class of cographs, with applications to scheduling, computational semantics, and clustering. More precisely, the P_4-reducible graphs are exactly the graphs none of whose vertices belong to more than one chordless path with three edges. A remarkable property of P_4-reducible graphs is their unique tree representation up to isomorphism. In this paper we present a linear-time algorithm to recognize P_4-reducible graphs and to construct their corresponding tree representation.

1. Introduction

The class of cographs arises naturally in many different areas of applied mathematics and computer science [2–4, 7–12]. Jamison and Olariu [5] introduced the notion of a P_4-reducible graph: this is a graph none of whose vertices belongs to more than one P_4. Clearly, P_4-reducible graphs strictly contain the class of cographs. As it turns out, a remarkable property of the P_4-reducible graphs is their unique tree representation up to (labelled) tree isomorphism. The purpose of this paper is to present a linear-time incremental algorithm to recognize P_4-reducible graphs. As a by-product of our algorithm we obtain, for a P_4-reducible graph G, in linear time, the largest induced cograph of G. Our recognition algorithm can be perceived as computing an incremental modular decomposition of the graph at hand [13]. Our recognition algorithm is subsequently used for the purpose of obtaining the unique tree associated with a P_4-reducible graph.

The paper is organized as follows: Section 2 provides background information on cographs and P_4-reducible graphs; Section 3 presents our linear-time recognition for

[☆] This work was supported by the National Science Foundation under grant CCR-9407180.
*Corresponding author.

P_4-reducible graphs; finally, Section 4 shows how to use the canonical cotree of a P_4-reducible graph G in order to obtain in linear time the corresponding pr-tree.

2. Background and terminology

All the graphs in this work are finite, with no loops nor multiple edges. In addition to standard graph-theoretical terminology compatible with Berge [1], we use some new terms that we are about to define. For a vertex x of a graph G, $N_G(x)$ will denote the set of all the vertices of G which are adjacent to x: since we assume adjacency to be nonreflexive, $x \notin N_G(x)$. We let $d_G(x)$ stand for $|N_G(x)|$.

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* while b and c are termed *midpoints* of the P_4. Consider a P_4 in G induced by $A = [a, b, c, d]$. A vertex x outside A is said to have a *partner* in A if x together with three vertices in A induces a P_4 in G. Given an induces subgraph H of G and a vertex x outside H, we say that x is *natural* with respect to H if x has a partner in no P_4 in H. 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.

Lerchs [8] 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-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 [9] 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,

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

Next, Jamison and Olariu proved the following fundamental results [5, Theorems 1 and 2] which is at the heart of a constructive characterization of P_4-reducible graphs.

Proposition 1. *A graph G is P_4-reducible if, and only if, for every induced subgraph H of G exactly one of the following conditions are satisfied:*

(i) *H is disconnected;*

(ii) *$\bar{H}$ is disconnected;*

(iii) *there exists a unique P_4 abcd in H such that every vertex of H outside $\{a, b, c, d\}$ is adjacent to both b and c and nonadjacent to both a and d.*

For the purpose of constructing the P_4-reducible graphs, Jamison and Olariu [5] defined yet another graph operation as follows. Let the graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ $(V_1 \cap V_2 = \emptyset)$ be such that $V_1 = \{a, d\}$, $E_1 = \emptyset$, and some adjacent vertices b, c in V_2 are adjacent to all the remaining vertices in V_2. Now

$$G_1 \textcircled{2} G_2 = (V_1 \cup V_2 \{ab, cd\} \cup E_2). \qquad (*)$$

Proposition 2. *G is a P_4-reducible graph if, and only if, G is obtained from single-vertex graphs by a finite sequence of operations $\textcircled{0}$, $\textcircled{1}$, $\textcircled{2}$.*

The following natural observation follows directly from Proposition 2.

Observation 0. *Let G be a P_4-reducible graph. If G $(\bar{G})$ is disconnected with components $G_1, G_2, \ldots, G_p$ $(p \geqslant 2)$, then we can write $G = G_1 \textcircled{0}(\textcircled{1}) \cdots \textcircled{0}(\textcircled{1}) G_p$.*

Propositions 1, 2, and Observation 0 combined suggest a natural way of associating with every P_4-reducible graph G a tree $T(G)$ (called the *pr-tree* of G), as described by the following recursive procedure.

```
Procedure Build_tree($G$);
{Input: a $P_4$-reducible graph $G = (V, E)$;
 Output: the pr-tree $T(G)$ corresponding to $G$.}
begin
  if $|V| = 1$ then
    return the tree $T(G)$ consisting of the unique vertex of $G$;
  if $G$ ($\bar{G}$) is disconnected then begin
    let $G_1, G_2, \ldots, G_p$ $(p \geqslant 2)$ be the components of $G$ ($\bar{G}$);
    let $T_1, T_2, \ldots, T_p$ be the corresponding pr-trees rooted at $r_1, r_2, \ldots, r_p$;
    return the tree $T(G)$ obtained by adding $r_1, r_2, \ldots, r_p$ as children of a node
    labelled 0 (1);
    end
  else begin {now both $G$ and $\bar{G}$ are connected}
    write $G = G_1 \textcircled{2} G_2$ as in (*);
    let $T_1, T_2$ be the corresponding pr-trees rooted at $r_1$ and $r_2$;
    return the tree $T(G)$ obtained by adding $r_1, r_2$ as children of a node labelled 2
    end
end; {Build_tree}
```

As it turns out (see [5]) the pr-tree of a P_4-reducible graph G is unique up to isomorphism. Let $G = (V, E)$ be a P_4-reducible graph. The *canonical cograph*

$C(G)$ associated with G is the induced subgraph of G obtained by the following procedure.

```
Procedure Greedy($G$):
{Input: a $P_4$-reducible graph $G$;
 Output: the canonical cograph $C(G)$}
begin
  $H \leftarrow G$;
  while there exist $P_4$'s in $H$ do begin
    pick a $P_4$ $uvxy$ in $H$:
    pick $z$ at random in $\{u, y\}$;
    $H \leftarrow H - \{z\}$
  end;
  return($H$)
end;
```

Clearly, procedure Greedy removes *precisely* one endpoint of every P_4 in G. The fact that the graph $C(G)$ returned by Greedy is a cograph follows from the definition of P_4-reducible graphs; the uniqueness implied by the *definition* of the canonical cograph is justified by the following result.

Proposition 3 (Jamison and Olariu [5, Theorem 3]). *The canonical cograph of a P_4-reducible graph is unique up to isomorphism.*

3. The recognition algorithm

To outline our recognition algorithm for P_4-reducible graphs, consider an arbitrary graph G. We assume that we have already processed a nonempty induced P_4-reducible subgraph H of G. (Note that such a subgraph H can always be found: in fact, the subgraph induced by a subset of at most four vertices in G is a P_4-reducible graph.)

The relevant information about H is stored in the tuple $(T(H), L(H))$: $T(H)$ is the cotree associated with the canonical cograph $C(H)$ of H (we shall refer to $T(H)$ as the *canonical cotree* of H); $L(H)$ contains precisely one endpoint of every P_4 in H. In addition, for the purpose of checking that no vertex belongs to more than one P_4, those vertices that are known to belong to some P_4 in H are "flagged".

To process a new vertex x we need to verify the following conditions:

- x is neutral with respect to H;
- x belongs to at most one P_4 in $H + x$; furthermore, this P_4 involves no "flagged" vertex;

Trivially, if either of these conditions fails, then $H + x$ cannot be a P_4-reducible graph and the algorithm terminates. If, on the other hand, both conditions are satisfied, then $H + x$ is a P_4-reducible subgraph of G and we proceed to update the tuple $(T(H), L(H))$. This involves the following operations:

- if x belongs to no P_4 in $H + x$, then x is added as a leaf in $T(H)$, and $L(H)$ is unchanged;
- if x is an endpoint of a P_4 in $H + x$, then $T(H)$ is unchanged, and x is added to $L(H)$;
- if x is a midpoint of a P_4 in $H + x$, then with y standing for an endpoint of this P_4, we do the following: y is removed from $T(H)$ and added to $L(H)$; x is added as a leaf in $T(H - y)$.

Our recognition algorithm for P_4-reducible graphs relies, in part, on a marking scheme similar to that developed by Corneil et al. [4]. We borrow their notation relevant to the marking scheme.

For a vertex u in the canonical cotree $T(H)$, rooted at R, we let $d(u)$ stand for the number of children of u; $md(u)$ represents the current number of marked, and subsequently unmarked children of u. (Initially, $md(u)$ is 0 for all the nodes u in $T(H)$; when u is unmarked, $md(u)$ is reset to 0.) A marked 1-node of $T(H)$ is said to be *properly marked* whenever $md(u) = d(u) - 1$; otherwise it will be termed *improperly marked*.

The next procedure using the adjacency information of a new vertex x performs the following:

- marks, and subsequently unmarks, as appropriate, certain nodes of $T(H)$;
- builds up a linked list $\Pi(x)$ of P_4's in H containing vertices adjacent to x;
- adds marked but not subsequently unmarked nodes of $T(H)$ to one or the other of the linked lists M_0 (containing marked 0-nodes). M_1 (containing improperly marked 1-nodes) or M_2 (containing properly marked 1-nodes).

```
Procedure Mark($x$);
 0. begin
 1.  $M_0 \leftarrow M_1 \leftarrow M_2 \leftarrow \emptyset$; $c_0 \leftarrow c_1 \leftarrow c_2 \leftarrow 0$; $\Pi(x) \leftarrow \emptyset$;
 2.  for each $v$ in $N_G(x)$ do
 3.    if ($v$ is a leaf in $T(H)$ or ($v \in L(H)$)) then begin
 4.      $\Pi(x) \leftarrow \Pi(x) \cup \{P_4$ in $H$ containing $v\}$:
 5.      mark $v$ unless $v \in L(H)$
 6.    end;
 7.  for each marked node $u$ in $T(H)$ do
 8.    if $d(u) = md(u)$ then begin
 9.      unmark $u$:
10.      $md(u) \leftarrow 0$;
11.      if $u \neq R$ then begin
12.        $w \leftarrow p(u)$;
13.        mark $w$;
14.        $md(w) \leftarrow md(w) + 1$;
15.        add $u$ to the list of marked and subsequently unmarked children of $w$
```

```
16.       end
17.     end
18.   else {now $d(u) \neq md(u)$, and so $u$ is marked but not unmarked}
19.     case label ($u$) of
20.       0: begin ($u$ is a marked 0-node}
21.            $c_0 \leftarrow c_0 + 1$;
22.            $M_0 \leftarrow M_0 \cup \{u\}$
23.            end;
24.       1: begin
25.            if $md(u) \neq d(u) - 1$ then begin $u$ is improperly marked
26.              $c_1 \leftarrow c_1 + 1$;
27.              $M_1 \leftarrow M_1 \cup \{u\}$
28.              end
29.            else begin {now $u$ is a properly marked 1-node}
30.              $c_2 \leftarrow c_2 + 1$;
31.              $M_2 \leftarrow M_2 \cup \{u\}$
32.              end
33.          end
34.     endcase;
35.   if ($c_0 + c_1 + c_2 > 0$) and $d(R) = 1$ then mark $R$
36. end; {Mark}
```

In the remainder of this paper a node w of $T(H)$ will be referred to as *marked* only if w remains marked at the end of procedure Mark (i.e. w is marked but not subsequently unmarked). For a node w in $T(H)$, $T(w)$ will denote the subtree of $T(H)$ rooted at w. For later reference, we make note of the following simple observations.

Observation 1. *Let w be a marked node in $T(H)$. There must exist a child w' of w such that all the leaves in $T(w')$ are adjacent to x.*

Observation 2. *Let w be a never marked or a marked, but not unmarked, node of $T(H)$. There must exist a descendant w'' of w in $T(w)$ such that all the leaves in $T(w'')$ are nonadjacent to x.*

Let w be an arbitrary node of $T(H)$ and let $I(w)$ stand for the set of children of w which have a marked (and not subsequently unmarked) descendant in $T(H)$. Let $T'(w)$ stand for the subtree of $T(w)$ defined by

$$T'(w) = T(w) - \bigcup_{u \in I(w)} T(u).$$

Partition of the leaves of $T'(w)$ into nonempty, disjoint sets $A(w)$ and $B(w)$, in such a way that x is adjacent to all the leaves in $A(w)$ and nonadjacent to all the leaves in $B(w)$.

Observation 3. *w is the lowest common ancestor of any leaves a in $A(w)$ and b in $B(w)$.*

If $T(H)$ contains marked nodes, then the marked node with the lowest level in $T(H)$, denoted $\alpha(x)$ (or simply α, if no confusion is possible) plays a distinguished role in our algorithm. (If several marked nodes are at the same level, pick one at random.) Let

$$(P) \quad R = w_1, w_2, \dots, w_p = \alpha(x) \quad (p \geqslant 1) \tag{1}$$

stand for the unique path in $T(H)$ joining R and α. The path (P) is referred to as *complete* if no marked vertex in $T(H)$ lies outside (P).

For nodes w_j with $1 \leqslant j \leqslant p-1$ of a complete path (P), the subtree $T(w_j) - T(w_{j+1})$ contains no marked node: as before, we let

- $A(w_j)$ stand for the set of leaves in$T(w_j) - T(w_{j+1})$ which are adjacent to x;
- $B(w_j)$ stand for the set of leaves in $T(w_j) - T(w_{j+1})$ which are not adjacent to x.

For w_p ($= \alpha(x)$), denote by

- $A(w_p)$ the set of all the leaves in $T(w_p)$ which are adjacent to x;
- $B(w_p)$ the set of all the remaining leaves in $T(w_p)$.

Observation 4. *No w_k $(1 \leqslant k \leqslant p)$ on the path (P) is marked and subsequently unmarked.*

Observation 5. *Let w be an arbitrary unmarked node, or an improperly marked* 1*-node in P. There exists a nonempty set S of leaves of $T(w)$, such that x is nonadjacent to all the leaves in S.*

Observation 6. *If $d(R) = 1$ and R is marked, then R is properly marked.*

Call a node w_j $(1 \leqslant j \leqslant p-1)$ of (P) regular if w_j is either a properly marked 1-node or else an unmarked 0-node. Otherwise, w_j is termed special. The path (P) is said to be *admissible* if the following conditions are satisfied.

(a1) (P) is complete:

(a2) there is at most one subscript k $(1 \leqslant k \leqslant p-1)$ such that the node w_k is special. Furthermore, if a special node exists, then the following conditions must be true

(a2.1) $k = p-2$ or $k = p-1$;

(a2.2) if $k = p-1$ then $|A(w_p)| = |B(w_p)| = 1$ with both vertices in $A(w_p)$ and $B(w_p)$ unflagged; furthermore.

- $|B(w_k)| = 1$ and the vertex in $B(w_k)$ is unflagged whenever w_p is a 0-node; and
- $|A(w_k)| = 1$ and the vertex in $A(w_k)$ is unflagged whenever w_p is a 1-node.

(a2.3) if $k = p-2$ then

- $|B(w_p)| = |A(w_{p-1})| = |A(w_k)| = 1$ with none of the vertices in $B(w_p)$, $A(w_{p-1})$ $A(w_k)$ flagged and $B(w_{p-1}) = \emptyset$ whenever w_p is a 0-node;
- $|A(w_p)| = |B(w_{p-1})| = |B(w_k) = 1$ with none of the vertices in $A(w_p)$, $B(w_{p-1})$, $B(w_k)$ flagged and $A(w_{p-1}) = \emptyset$ whenever w_p is a 1-node;

Note that, if $T(H)$ contains no marked nodes, then the path (P) is, trivially, empty and hence vacuously admissible. Now in our notation, Theorem 1 in Corneil et al. [4] can be formulated as follows.

Proposition 4 (Corneil et al. [4]). *If H is a cograph, then $H + x$ is a cograph if, and only if, the path in $T(H)$ joining the root and $\alpha(x)$ is admissible and contains no special nodes.*

We are now ready to state a result which provides the theoretical basis for our recognition algorithm for P_4-reducible graphs. We assume the existence of an underlying graph $G = (V, E)$ which is in the process of being investigated by the recognition algorithm. For the proof the reader is referred to [6].

Theorem 1. *If H is a P_4-reducible graph, then $H + x$ is a P_4-reducible graph if, and only if, x is neutral with respect to H and the path joining the root of $T(H)$ and $\alpha(x)$ is admissible.*

Corollary 1. *If $|M_0 \cup M_1| > 2$, then $H + x$ is not a P_4-reducible graph.*

Proof. If $c_0 + c_1 > 2$ then the path (P) joining α and R cannot be admissible. The conclusion follows by Theorem 1. □

As previously mentioned, our recognition algorithm for P_4-reducible graphs is incremental. Given a graph $G = (V, E)$, whose vertices are enumerated as $v_1, v_2, \ldots, v_n$ we proceed in the following two stages.

```
Algorithm Recognize($G$);
Stage 1. [Initialization]
    set all the vertices in $G$ "unflagged";
    $H \leftarrow \{v_1, v_2\}$;
    construct the cotree $T(H)$ rooted at $R$;
    $L(H) \leftarrow \emptyset$;
Stage 2. [Incrementally process the remaining vertices in $G - H$, as follows]
    Step 2.0. pick $x$ in $G - H$; Mark($x$);
    Step 2.1. if $x$ is not neutral with respect to $H$ then return('no");
    Step 2.2. if $x$ belongs to more than one $P_4$ or if $x$ belongs to a $P_4$ involving
              a "flagged" vertex in $H + x$ then return("no");
    Step 2.3. $H \leftarrow H + x$; update ($T(H)$, $L(H)$).
```

We assume that upon executing the statement **return("no")** the entire algorithm terminates: $H + x$ is not a P_4-reducible graph (this will be justified later). Since the details of Step 2.0 have been discussed in Section 3, we shall turn our attention to the remaining steps in Stage 2. For this purpose, we note that Step 2.1 can be implemented by the following procedure.

```
Procedure Test_Neutral($x$);
{$\Pi(x)$ is a list of $P_4$'s created in procedure Mark}
  1. begin
  2. while $\Pi(x) \neq \emptyset$ do begin
  3.    pick a $P_4$ in $\Pi(x)$ with endpoints $x_0$ and $x_1$ and midpoints $x_1$ and $x_2$;
  4.    if $x$ has a partner in $\{x_0, x_1, x_2, x_3\}$ then return ("no");
  5.    $\Pi(x) \leftarrow \Pi(x) - \{x_0, x_1, x_3\}$
  6.  end
  7. end;
```

Two nodes of $T(H)$ play a distinguished role in Steps 2.2–2.3; first, $\alpha(x)$ stands, as before, for a marked node in $T(H)$ with the lowest level (ties being broken arbitrarily); next, $\gamma(x)$ is a candidate for a special node on the path joining α and R. (We shall write, simply, α and γ instead of $\alpha(x)$ and $\gamma(x)$ since no confusion is possible.)

Step 2.2. is further refined into two substeps as follows.

Step 2.2. [if x belongs to more than one P_4 or if x belongs to a P_4 involving a "flagged' vertex in $H + x$ then return ("no");]

Step 2.2.1. Find α:

Step 2.2.2. If the path in $T(H)$ joining R and α is not admissible then return("no");

Step 2.2.1 is implemented by the procedure Find whose details are given below.

```
Procedure Find;
{returns a node that plays the role of $\alpha$.}
  1. begin Find $\leftarrow$ undefined;
  2. if $c_0 \leftarrow c_1 + c_2 = 0$ then Find $\leftarrow \Lambda$;
  3. case $c_0 + c_1$ of
  4.  0: if $p(p(z))$ is an unmarked node of $T(H)$ for some $z$ in $M_2$ then
  5.         Find $\leftarrow z$
  6.       else begin
  7.          let $z$ be a node in $M_2$ such that $z \neq p(p(z'))$ for all $z' \in M_2$;
  8.          Find $\leftarrow z$
  9.       end;
 10.  1: begin
 11.         let $z$ be the unique node in $M_0 \cup M_1$;
 12.         if $z = p(z')$ or $z = p(p(z'))$ for some $z' \in M_2$ then
 13.            Find $\leftarrow z'$
 14.         else
 15.            Find $\leftarrow z$
 16.     end;
```

```
17. 2: if for distinct $z$, $z'$ in $M_0 \cup M_1$, $z' = p(z)$ or $z' = p(p(z))$ then
18.         Find $\leftarrow z$
19. endcase
20. end; {Find}
```

The following result (for the proof refer to [5b]) shows that $H + x$ is a P_4-reducible graph only if the node returned by the procedure Find can play the role of α. More precisely:

Fact 5. *Let z be the node returned by the function Find. $H + x$ is P_4-reducible, only if the following statements are satisfied:*

(5.1) *$z = \Lambda$ whenever $T(H)$ contains no marked nodes;*

(5.2) *z and α coincide whenever $T(H)$ contains marked nodes.*

We assume that whenever the unmark w statement is executed during Steps 2.2 and 2.3 with $w \in M_1$, the following statements are implicitly performed

$$M_i \leftarrow M_i - \{w\}; \quad c_i \leftarrow c_i - 1; \quad md(w) \leftarrow 0;$$

Step 2.2.2 is implemented by the procedure Test_Admissible whose details are spelled out next. As justified by Fact 5, we may use α for the node returned by procedure Find.

```
Procedure Test_Admissible; {tests the path in $T(H)$ joining $\alpha$ and $R$ for admissibility.}
 1. begin
 2.   if $\alpha$ = undefined then return("no");
 3.     $\gamma \leftarrow \alpha$; if $\alpha = \Lambda$ then exit;
 4.   if ($p(\alpha) \in M_0$) or (label($p(\alpha)$) = 1 and $p(\alpha) \notin M_2$) then $\gamma \leftarrow p(\alpha)$
 5.   else if ($p(p(\alpha)) \in M_0$) or (label($p(p(\alpha))$) = 1 and $p(p(\alpha)) \notin M_2$) then $\gamma \leftarrow p(p(\alpha))$;
      {to begin, check the path between $\gamma$ and $R$}
 6.   $z \leftarrow \gamma$;
 7.   if label($z$) = 0 then begin
 8.     $z \leftarrow p(z)$
 9.   else $z \leftarrow p(p(z))$;
10.   while $z \in T(H)$ do begin
11.     if $z \notin M_2$ then return("no") else unmark $z$;
12.     $z \leftarrow p(p(z))$
13.   end;
      {check whether an appropriate number of nodes remain marked}
14.   if ($\gamma = p(p(\alpha))$) and (label($\alpha$) = 0) then {we know that $p(\alpha) \in M_2$}
15.     unmark $p(\alpha)$;
16.   if ($c_0 + c_1 + c_2 > 2$) or (($c_0 + c_1 + c_2 > 1$) and ($\gamma$ not marked)) then
        return("no");
      (finally, check conditions 2.2 and 2.3)
```

```
17. case $\gamma$ of
18    $p(\alpha)$: begin if $|A(\alpha)| \neq 1$ or $|B(\alpha)| \neq 1$ or one of the vertices in $A(\alpha)$, $B(\alpha)$ is
      "flagged" then return("no")
                  else flag the vertices in $A(\alpha)$, $B(\alpha)$;
19.              if label$(\alpha) = 0$ and ($|B(\gamma)| \neq 1$ or $B(\gamma)$ contains a "flagged" vertex)
                 then return("no")
                 else flag the vertex in $B(\gamma)$;
20.              if label$(\alpha) = 1$ and ($|A(\gamma)| \neq 1$ or $A(\gamma)$ contains a "flagged" vertex)
                 then return("no")
                 else flag be vertex in $A(\gamma)$
21.            end;
22.  $p(p(\alpha))$: begin if label$(\alpha) = 0$ and ($|B(\alpha)| \neq 1$ or $|A(p(\alpha))| \neq 1$ or $|A(\gamma)| \neq 1$ or
            $B(\alpha)$, $A(p))$, $A(\gamma)$ contain "flagged" vertices
                  or $B(p(\alpha)) \neq \emptyset$) then return("no")
              else flag the vertices in $B(\alpha)$, $A(p(\alpha))$, $A(\gamma)$;
23.           if label$(\alpha) = 1$ and ($|A(\alpha)| \neq 1$ or $|B(\gamma)| \neq 1$ or $|B(p(\alpha))| \neq 1$ or $A(\alpha)$,
              $B(\gamma)$, $B(p(\alpha))$ contain "flagged" vertices
                  or $A(p(\alpha)) \neq \emptyset$) then return("no")
               else flag the vertices in $A(\alpha)$, $B(\gamma)$, $B(p(\alpha))$
24.          end
25.   endcase;
26.  if $\alpha \neq \gamma$ then flag $x$
27. end; {Test_Admissible}
```

Fact 6. *The path (P) in T(H) from α to R is admissible if, and only if the statement return "no" is not executed in Test_Admissible.*

The proof of Fact 6 can be found in [6]. We note that by virtue of Facts 5 and 6, Theorem 1 can be reformulated as follows.

Theorem 2. *If H is a P_4-reducible graph, then $H + x$ is a P_4-reducible graph if, and only if, the statement return("no) is not executed in Steps* 2.1 *and* 2.2.

To make our arguments more transparent, we further refine Step 2.3 as follows

```
Step 2.3 [$H \leftarrow H + x$; update $(T(H), L(H))$]
  if $\alpha = \gamma$ then
    Update_1
  else
    Update_2;
```

Here, Update_1 is reminiscent of the way Corneil et al. [4] update the cotree once they know that x is contained in no P_4 in $H + x$. The procedure Update_2 deals with the more general case where the path in $T(H)$, though admissible, is known to contain a special node, namely γ. $T(H)$ is altered to represent the canonical cotree of $H + x$. The details of these two procedures are spelled out next.

```
**Procedure** Update_1;
{$x$ is contained in no $P_4$ in $H + x$;
 we do: $H \leftarrow H + x$; $T(H + x) \leftarrow T(H) + x$; $L(H + x) \leftarrow L(H)$}
 1. begin
 2.   if $\alpha = \Lambda$ then
 3.     if all nodes in $T(H)$ were marked and subsequently unmarked then
 4.       add $x$ as a child of $R$
 5.     else {no node in $T(H)$ was marked}
 6.       if $d(R) = 1$ then
 7.         make $x$ a child of the (only) child of $R$
 8.       else begin
 9.         make the old root and $x$ children of a new 0-node $\theta$;
10.         make $\theta$ the only son of the new root
11.         end
12.   else {now $\alpha$ is the only marked node in $T(H)$}
13.     if label($\alpha$) = 0(1) then
14.       if $md(\alpha) = 1 (d(\alpha) - md(\alpha) = 1)$ then begin
15.         $\lambda \leftarrow$ unique marked and unmarked (never marked) child of $\alpha$ in $T(\alpha)$;
16.         if $\lambda$ is a leaf in $T(H)$ then begin
17.           make $\lambda$, $x$ children of a new node $\theta$;
18.           make $\theta$ a child of $\alpha$
19.           end
20.         else
21.           make $x$ a child of $\lambda$
22.         end
23.       else begin {now $md(\alpha) \neq 1$ $(d(\alpha) = md(\alpha) \neq 1)$}
24.         add every marked child of $\alpha$ to a new node $\theta$ with label($\theta$) = label($\alpha$);
25.         if label($\alpha$) = 0 then begin
26.           make $x$, $\theta$ children of a new node $\theta'$;
27.           make $\theta'$ child of $\alpha$
28.           end
29.         else begin
30.           make $\theta$ a child of $p(\alpha)$;
31.           make $x$, $\alpha$ children of a new node $\theta'$;
32.           make $\theta'$ a child of $\theta$
33.           end
34.          end
35. end; {Update_1}
```

To specify the details of the procedure Update_2, we shall find it convenient to introduce the following notation:

- write $A(\alpha) = \{a\}$, whenever $|A(\alpha)| = 1$;
- write $B(\alpha) = \{b\}$, whenever $|B(\alpha)| = 1$;
- write $A(\gamma) = \{c\}$, whenever $|A(\gamma)| = 1$;
- write $B(\gamma) = \{d\}$, whenever $|B(\gamma)| = 1$;

  if $\gamma \neq p(\alpha)$ then
- write $A(p(\alpha)) = \{t\}$ whenever $|A(p(\alpha))| = 1$;
- write $B(p(\alpha)) = \{t'\}$ whenever $|B(p(\alpha))| = 1$;

For the purpose of justifying our way of updating the tuple $(T(H), L(H))$ in Step 2.3 we need the following intermediate result (see [6]).

Fact 7. *x is the endpoint of a unique P_4 in $H + x$ if, and only if, γ is a 1-node.*

Procedure Update_2;
{x is contained in precisely one P_4 in $H + x$;
the procedure performs $H \leftarrow H + x$ and updates $T(H)$ and $L(H)$ accordingly}

```
begin
 if label($\gamma$) = 1 then begin
    $T(H + x) \leftarrow T(H)$;
    $L(H + x) \leftarrow L(H) \cup \{x\}$;
    end
 else begin {now $\gamma$ is a 0-node}
    $L(H + x) \leftarrow L(H) \cup \{b\}$;
    remove $b$ from $T(H)$;
    case $\alpha$ of
       1: if $B(\gamma) = \emptyset$ then begin
             add $a$ as a child of $\gamma$;
             add $x$ as a child of $p(\gamma)$;
             remove $\alpha$ from $T(H)$
             end
          else begin
             make $a$, $c$, children of a new 0-node $\theta$;
             make $\theta$, $x$ children of $\alpha$
             end;
       0: begin
             if $B(\gamma) = \emptyset$ then
                add $x$ as a child o $p(\gamma)$
             else begin
                make $p(\alpha)$ and $c$ children of a new 0-node $\theta$;
                make $\theta$ and $x$ children of a new 1-node $\theta'$;
                make $\theta'$ a child of $\gamma$;
             end;
```

```
27.            if $md(\alpha) = 1$ then begin
28.              let $\alpha'$ be the marked and subsequently unmarked child of $\alpha$;
29.              if $\alpha'$ is a leaf in $T(H)$ then
30.                make $\alpha'$ child of $p(\alpha)$
31.              else begin
32.                make every child of $\alpha'$ a child of $p(\alpha)$;
33.                remove $\alpha'$ from $T(H)$
34.                end;
35.              remove $\alpha$ from $T(H)$
36.            end;
37.            unmark $\alpha$, unless already removed;
38.            if $\gamma$ is marked then unmark $\gamma$
39.          end
40.    endcase
41.  end; {Update}
```

Fact 8. *The cotree $T(H + x)$ returned by Step 2.3 is the canonical cotree of $H + x$.*

Our next result shows that the iteration consisting of processing $x \in G - H$ takes time proportional to the degree of x. The reader can find the proof in [6].

Theorem 3. *Given a P_4-reducible graph H specified by $(T(H), L(H))$ and a given vertex $x \notin H$, the algorithm Recognize performs in time $O(d_G(x))$ one of the following:*
(i) *either determines that $H + x$ is not a P_4-reducible graph, or else*
(ii) *incorporates x into H, updating $T(H)$ and $L(H)$ accordingly.*

5. A tree representation for P_4-reducible graphs

Let G be a P_4-reducible graph represented by the tuple $(T(G), L(G))$. We now address the problem of efficiently constructing the pr-tree representation of G. For this purpose we shall use the fact that $T(G)$ is the canonical cotree of G (i.e. the cotree corresponding to the canonical cograph $C(G)$ of G), and that every vertex in $L(G)$ is endpoint of precisely one P_4 in G. Our arguments make use of the following result whose proof can be found in [6].

Theorem 4. *For every $u \in L(G)$ such that $uvwz$ is a P_4 in G with v, w, z in $T(G)$, there exist a unique 0-node $\lambda(u)$ and a 1-node $\lambda'(u)$ in $T(G)$ satisfying*

$$\lambda(u) = p(z); \quad \lambda'(u) = p(w); \quad \lambda'(u) = p(\lambda(u)), \tag{11}$$

Furthermore,

$$\text{either } \lambda(u) = p(v) \text{ or else } \lambda''(u) = p(v) \text{ with } \lambda(u) = p(\lambda''(u)). \tag{12}$$

Since for every vertex u in $L(G)$ there is a *unique* $\lambda(u)$ with the properties mentioned in Theorem 4, we shall write simply λ, λ', λ'' dropping the reference to u.

To construct the tree representation of a P_4-*reducible* graph G, we need a way of incorporating the vertices of $L(G)$ 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 operation as in (∗). The details of theis tree construction are spelled out in the following procedure.

```
Procedure Build-tree1($G$);
{Input: a $P_4$-reducible graph represented as $(T(G), L(G))$
 Output: a tree $T1(G)$, rooted at $R$;}
 1. begin
 2.    $T1(G) \leftarrow T(G)$;
 3. while $L(G) \neq \emptyset$ do begin
 4.    pick an arbitrary vertex $u$ in $L(G)$;
 5.    find $v, w, z$ in $T1(G)$ such that $uvzw$ is a $P_4$ in $G$;
 6.    $\lambda \leftarrow p(z)$; $\lambda' \leftarrow p(w)$; $\lambda'' \leftarrow p(v)$;
 7.    create a 2-node $\beta$;
 8.    add $u$ as a child of $\lambda$;
 9.   if $\lambda = \lambda''$ then begin
10.       if $d(\lambda') \neq 2$ then begin
11.          add $\beta$ as a child of $\lambda'$;
12.          add $\lambda$ and a new 1-node $\tau$ as children of $\beta$;
13.          add $v, w$ as children of $\tau$
14.          end
15.       else begin
16.          add $\lambda, \lambda'$ as children of $\beta$;
17.          add $\beta$ as a child of $p(\lambda')$;
18.          add $v$ as a child of $\lambda'$
19.          end
20.    else begin
21.       add $\lambda, \lambda''$ as children of $\beta$;
22.       add $w$ as a child of $\lambda''$;
23.      if $d(\lambda') \neq 2$ then
24.          add $\beta$ as a child of $\lambda'$
25.       else begin
26.          add $\beta$ as a child of $p(\lambda')$;
27.          remove $\lambda'$ from $T1(G)$
28.          end
29.    end;
30.    $L(G) \leftarrow L(G) - \{u\}$
31. end;
```

```
32. if $d(R) = 1$ then $R \leftarrow$ unique child of $R$
33. end;
```

The following result argues about the correctness and the running time of procedure Build-tree1. More precisely, we have the following theorem whose proof can be found in [6].

Theorem 5. *The tree $T1(G)$ returned by the procedure Build-tree1 is precisely the pr-tree corresponding to G. Furthermore, $T1(G)$ is constructed in linear time.*

References

[1] C. Berge, *Graphs and Hypergraphs* (North-Holland, Amsterdam, 1973).

[2] D.G. Corneil and D.G. Kirkpatrick, Families of recursively defined perfect graphs, *Congr. Numer.* **39** (1983) 237–246.

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

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

[5] B. Jamison and S. Olariu, *P_4-reducible*-graphs, a class of uniquely tree representable graphs, *Stud. Appl. Math.* **81** (1989) 79–87.

[6] B. Jamison and S. Olariu, A linear-time algorithm to recognize P4-reducible graphs, in: *Proc. 9th Conf. On Foundations of Software Technology and Theoretical Computer Science*, Bangalore, India, 1989, Lecture Notes in Computer Science, (Springer, Berlin, 1989) 1–19.

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

[8] H. Lerchs, On cliques and kernels, Dept. of Computer Science, University of Toronto, March 1971.

[9] H. Lerchs, On the clique-kernel structure of graphs, Dept. of Computer Science, University of Toronto, October 1972.

[10] D. Seinsche, On a property of the class of n-colorable graphs, *J. Combin. Theory Ser. B* **16** (1974) 191–193.

[11] L. Stewart, Cographs, a class of tree representable graphs, M.Sc. Thesis, Dept. of Computer Science, University of Toronto, 1978, TR 126/78.

[12] D.P. Sumner, Dacey Graphs, *J. Australian Math. Soc.* **18** (1974) 492–502.

[13] J.H. Muller and J. Spinrad, Incremental modular decomposition, *J. ACM* **36** (1989) 1–19.