Cograph Recognition Algorithm Revisited and Online Induced P_4 Search

Christian Capelle and Alain Cournier and Michel Habib

LIRMM
UMR 9928 UNIVERSITE MONTPELLIER II/CNRS
161, Rue Ada
34392 Montpellier cedex 5 France
email: {capelle, cournier, habib}@lirmm.fr

Abstract. In 1985, Corneil, Perl and Stewart [CPS85] gave a linear incremental algorithm to recognize cographs (graphs with no induced P_4). When this algorithm stops, either the initial graph is a cograph and the cotree of the whole graph has been built, or the initial graph is not a cograph and this algorithm ends up with a vertex v and a cotree *cot* such that v cannot be inserted in *cot*; so the input graph must contain a P_4. In many applications such as graph decomposition [Cou93, CH93a, CH93b, CH94, EGMS94, Spi92, MS94], transitive orientation [Spi83, ST94], not only the existence but a P_4 is also explicitly needed. In this paper, we present a new characterization of cograph in terms of its modular structure. This characterization yields a structural labeling of the cotree for incremental cograph recognition, and we show how to go from this labeling to the Corneil *et al.* one's. Furthermore, we show how to adapt this algorithm in order to produce a P_4 in case of failure when adding a new vertex v, in $O(|N(v)|)$ time complexity.

1 Introduction

The well known class of cograph, is the smallest one containing the one vertex graph and closed under union and complementation. In other words, any cograph G can be decomposed using only parallel decomposition (*i.e.* decomposition into connected components of G) and series decomposition (*i.e.* decomposition into connected components of $\overline{G}$), see [Cha47, Hab81, CE80, HM79, Kel85]). A decomposition rooted tree, called a *"cotree"*, using internal nodes labeled Series or Parallel, is associated to such a decomposition scheme (in such a tree, Series and Parallel labels form a coloring of internal nodes). This tree is a special case of the decomposition trees obtained in substitution (or modular) decomposition [Cou93, CH94]. Leaves of that cotree correspond to vertices of the graph. The leaves of the subtree induced by any node of a cotree is a *module* (i.e. a set of vertices M such that $\forall x, y \in M, N(x) - M = N(y) - M$, where $N(x)$ is the neighbourhood of x) of the cograph. Moreover these modules are *strong* (i.e. M is a strong module iff M is a module and $\forall M'$ module of G, $M \subseteq M'$, or $M' \subseteq M$, or $M \cap M' = \emptyset$). The smallest graph undecomposable according to the previous decomposition scheme is the path with four vertices, called P_4. In

the following, a $P_4 = [a, b, c, d]$ is the graph on vertex set $\{a, b, c, d\}$ with edge set $\{(a, b), (b, c), (c, d)\}$. Let us recall the fundamental property of cographs:

Property 1. *[Ler71, Ler72, Sei74] G is a cograph if and only if it does not contain P_4 as a subgraph .*

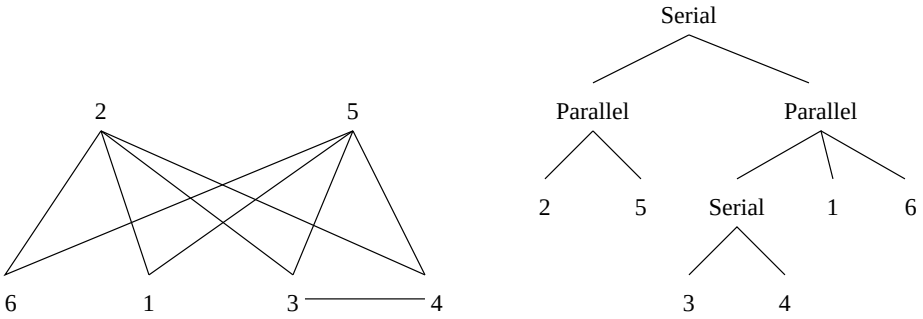

Fig. 1. A cograph and its cotree

As it can be noticed on the example of Fig. 1, the cotree contains all the informations of the graph: there exists an edge between two vertices a and b if and only if the nearest common ancestor of a and b in the cotree is a Series node. The ancestors of a node y of a rooted tree T are the nodes x of T such that there exists a bottom-up path from y to x. The nearest common ancestor of a set Y of nodes of T, is the common ancestor x of nodes of Y such that all other common ancestors of Y are ancestors of x.

Corneil, Perl and Stewart give the first incremental algorithm, called *CPS-algorithm* in the following, linear in time and space to build the cotree of any cograph [CPS85]. This algorithm constructs the cotree of a subgraph G_Y of G starting from a single vertex. Y increases of one vertex at each step of the algorithm. If the graph $G = (X, E)$ is not a cograph the algorithm stops as soon as it meets a vertex v of $X \setminus Y$ such that $G_{Y \cup \{v\}}$ is not a cograph. So, using Property 1, $G_{Y \cup \{v\}}$ contains a P_4.

But several applications need explicitly such a P_4 as a basis for further computations. Using a naive algorithm, a P_4 can be found in $O(|X|^3)$, but this complexity is too high. In this paper we complete *CPS-algorithm* in a way to find a P_4 in $O(|N_G(v)|)$ time complexity when the *CPS-algorithm* fails introducing v in the cotree. Since several authors used such an algorithm (which is far from obvious) as a basic tool in their work [Cou93, CH94, Spi92, MS94, ST94] without ever define it, we think it is worth to present it here in full details.

In the meantime, *CPS-algorithm* is reformulated using a structural characterization of cotrees presented in Sect. 2, based on their modular properties. The new version of *CPS-algorithm* is presented in Sect. 3, and a short proof of *CPS-algorithm* is given. An algorithm that searches efficiently a P_4 in case of failure when introducing a new vertex in the cotree is presented in Sect. 4.

2 The Modular Structure of Cotrees

Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$ initially empty, and $cot(G_Y)$ the cotree of G_Y. The elements of Y are the leaves of $cot(G_Y)$. We call L_y the set of leaves of the subtree rooted by any node y of $cot(G_Y)$. As previously noticed, for every node n of $cot(G_Y)$, L_n is a *strong module* of G_Y.

If G_Y is a cograph with cotree $cot(G_Y)$, $\overline{G_Y}$ is a cograph with cotree $cot(\overline{G_Y})$. $cot(\overline{G_Y})$ can be obtained from $cot(G_Y)$ just interchanging the labels (*Parallel* nodes become *Series* and dually).

Furthermore $P_4 = [a, b, c, d]$ is an induced subgraph of a graph G_Y iff $\overline{P_4} = [b, d, a, c]$ is an induced subgraph of $\overline{G_Y}$.

Characterization 2. *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of a cograph G_Y and v a vertex of $X \setminus Y$.*
$G_{Y\cup\{v\}}$ is a cograph if and only if for all nodes n of $cot(G_Y)$, L_n or $L_n \cup \{v\}$ is a module of $G_{Y\cup\{v\}}$.

Proof. $\Longleftarrow$ (*if*)
If G_Y is a cograph and $G_{Y\cup\{v\}}$ is not a cograph, necessarily there exists a $P_4 = [v, l_1, l_2, l_3]$ (case 1) or a $P_4 = [l_2, v, l_3, l_1]$ (case 2) in $G_{Y\cup\{v\}}$.

(1) $P_4 = [v, l_1, l_2, l_3]$. We call n the nearest common ancestor of l_1 and l_3 in $cot(G_Y)$. If we assume $l_2 \in L_n$, this implies the Parallel node n is the nearest common ancestor of l_1 and l_2 or l_3 and l_2 which is impossible. So $l_2 \notin L_n$. L_n is not a module of $G_{Y\cup\{v\}}$ since $(l_1, v) \in E$ and $(l_3, v) \notin E$. Moreover $L_n \cup \{v\}$ is not a module of $G_{Y\cup\{v\}}$ since elements of L_n are adjacent to l_2 and v is not adjacent to l_2.

(2) $P_4 = [l_2, v, l_3, l_1]$ is an induced subgraph of $G_{Y\cup\{c\}}$, so $\overline{P_4} = [v, l_1, l_2, l_3]$ is an induced subgraph of $\overline{G_{Y\cup\{c\}}}$. According to (1) there exists an internal node n in $cot(\overline{G_{Y\cup\{v\}}})$ such that L_n and $L_n \cup \{v\}$ are not modules of $\overline{G_{Y\cup\{v\}}}$. So, L_n and $L_n \cup \{v\}$ are not modules of $G_{Y\cup\{v\}}$.

$\Longrightarrow$ (*only if*)
Let m be an internal node of $cot(G_Y)$ such that L_m and $L_m \cup \{v\}$ are not modules of $G_{Y\cup\{v\}}$. Since L_m is not a module of $G_{Y\cup\{v\}}$ and L_m is a module of G_Y, so there exist l_1 and l_2 in L_m such that $(v, l_1) \in E$ and $(v, l_2) \notin E$. $L_m \cup \{v\}$ is not a module of $G_{Y\cup\{v\}}$ and L_m is a module of G_Y, so there exists an internal node n, the nearest in $cot(G_Y)$, such that n is the nearest common ancestor of leaves l_1, l_2 and l_3; where l_3 is a leaf of $cot(G_Y)$ such that $(l_3, v) \in E$ and n is a Parallel node (l_3 not adjacent to any element of L_m) (case 1), or $(l_3, v) \notin E$ and n is a Series node (l_3 is adjacent to all elements of L_m) (case 2). We call k the nearest common ancestor of l_1 and l_2 in $cot(G_Y)$.

For (case 1), see Fig. 2, n is a Parallel node, if k is a Series node, $(l_1, l_2) \in E$, then $[l_3, v, l_1, l_2]$ is a P_4. If k is a Parallel node, then the father k' of k in $cot(G_Y)$ is a Series node. All leaves of $L_{k'} \setminus L_k$ are adjacent to v, otherwise k' would have been chosen as n. We choose one of the elements of $L_{k'} \setminus L_k$ as l_1. So, the situation is the same than previously (see Fig. 2), $[l_3, v, l_1, l_2]$ is a P_4.

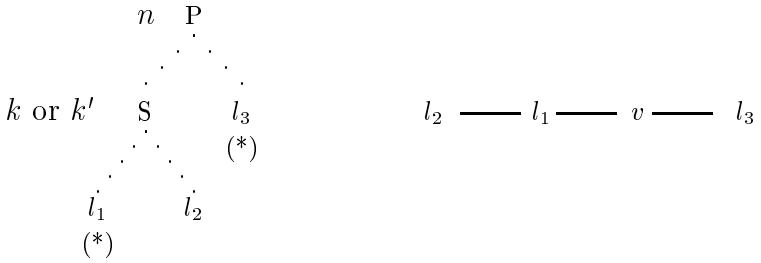

Fig. 2. case (1) – in all figures, leaves adjacent to v are denoted by (*)

(case 2) is dual from (case 1). n is a Series node and k or k', the father of k, is a Parallel node. In $cot(\overline{G_Y})$, n is a Parallel node, k or k' is a Series node, $(l_3, v) \in \overline{E}$, $(l_1, v) \notin \overline{E}$ and $(l_2, v) \in \overline{E}$. So this situation is equivalent to the previous (see Fig. 2) except roles of l_1 and l_2 are interchanged. So, $[l_3, v, l_2, l_1]$ is an induced subgraph of $\overline{G_{Y\cup\{v\}}}$, then $[v, l_1, l_3, l_2]$ is an induced subgraph of $G_{Y\cup\{v\}}$. □

$G_{Y\cup\{v\}}$ is a cograph if and only if $cot(G_Y)$ can be modified in order to introduce v as a new leaf, preserving the strong module structure of the subtrees. This yields immediately the following labeling of nodes n of the cotree $cot(G_Y)$ according to the strong module property of L_n and $L_n \cup \{v\}$ in $G_{Y\cup\{v\}}$.

Definition 3. Modular Labeling
Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$. Let us define the following labeling of $cot(G_Y)$. For each node n of $cot(G_Y)$:

1. n is labeled $InOut$ if L_n and $L_n \cup \{v\}$ are modules of $G_{Y\cup\{v\}}$ (in $cot(G_{Y\cup\{v\}})$, v can be present as a leaf In the subtree rooted by n, or it can can be left Out),
2. n is labeled Out if L_n is a module of $G_{Y\cup\{v\}}$ and $L_n \cup \{v\}$ is not a module of $G_{Y\cup\{v\}}$ (in $cot(G_{Y\cup\{v\}})$, v is necessarily Out of the subtree rooted by n),
3. n is labeled In if L_n is not a module and $L_n \cup \{v\}$ is a module of $G_{Y\cup\{v\}}$ (in $cot(G_{Y\cup\{v\}})$, v must be present as a leaf In the subtree rooted by n),
4. n is labeled $Failure$ if L_n and $L_n \cup \{v\}$ are not modules of $G_{Y\cup\{v\}}$.

We can remark that the root of $cot(G_Y)$ is always labeled In by Modular Labeling, even if $G_{Y\cup\{v\}}$ is not a cograph. As stated by Characterization 2, $cot(G_Y)$ admits a node labeled $Failure$ if and only if $G_{Y\cup\{v\}}$ is not a cograph. The following properties show that labels of nodes are not independent, and respect some rules. Moreover, these properties confirm the intuitive interpretation of the labeling given in Def. 3.

Property 4. *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$. The Modular Labeling of $cot(G_Y)$ satisfies the following properties:*

(a) Let n be a node labeled In, then all its ancestors in $cot(G_Y)$ are labeled In,
(b) Let n be a node labeled $Failure$, then all its ancestors in $cot(G_Y)$ are necessarily labeled In or $Failure$,
(c) Let n_1 be a node labeled In and n_2 a node labeled In or $Failure$, then the nearest common ancestor of n_1 and n_2 in $cot(G_Y)$ is equal to n_1 or n_2.

Proof. First, the two following basic properties have to be pointed out:

- n is labeled In or $Failure$ iff L_n contains at least a node adjacent to v and a node non adjacent to v in $G_{Y\cup\{v\}}$,
- n is labeled Out or $Failure$ iff there exists an ancestor n' of n such that n' is a Series node and $L_{n'} \setminus L_n$ contains at least a node non adjacent to v in $G_{Y\cup\{v\}}$, or n' is a Parallel node and $L_{n'} \setminus L_n$ contains at least a node adjacent to v in $G_{Y\cup\{v\}}$.

(a) : Let us assume n_1 a node labeled In and n_2 an ancestor of n_1 with a label different from In. Since $L_{n_1} \subseteq L_{n_2}$ then n_2 is labeled $Failure$. So there exists an ancestor n_3 of n_2 such that n_3 is a Series node and $L_{n_3} \setminus L_{n_2}$ contains at least a node non adjacent to v in $G_{Y\cup\{v\}}$, or n_3 is a Parallel node and $L_{n_3} \setminus L_{n_2}$ contains at least a node adjacent to v in $G_{Y\cup\{v\}}$. n_3 is also an ancestor of n_1 satisfying the same property for n_1 than for n_2, so n_1 must be labeled $Failure$, a contradiction.
(b) : L_n contains a node adjacent to v and a node non adjacent to v. Any ancestor n' of n is such that $L_{n'}$ contains also these two nodes. So, n' is labeled In or $Failure$.
(c) : Let us assume that n, the nearest common ancestor of n_1 and n_2, is different from n_1 and n_2. Then there exist $l_1, l_2 \in L_{n_1}$ such that $l_1 \in adj(v)$ and $l_2 \notin adj(v)$, and $l_3, l_4 \in L_{n_2}$ such that $l_3 \in adj(v)$ and $l_4 \notin adj(v)$. $L_{n_1} \cup \{v\}$ is a module of $G_{Y\cup\{v\}}$, so $adj(v) \setminus L_{n_1} = adj(L_{n_1}) \setminus \{v\}$. If n is a Parallel node $l_3 \in adj(v) - adj(L_{n_1})$, a contradiction. Similarly, if n is a Series node $l_4 \in adj(L_{n_1}) - adj(v)$, a contradiction. □

Proposition 5. *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$. The Modular Labeling of $cot(G_Y)$ satisfies the following property:*
$cot(G_Y)$ contains no node labeled $Failure$, if and only if the set of nodes in $cot(G_Y)$ labeled In is a path $\mu = n_0, n_1, \ldots n_l = root$, possibly empty, such that the children of n_0 are leaves or nodes labeled $InOut$ or Out.

Proof. $\Longrightarrow$ According to Properties 4.a and 4.c, if there are in $cot(G_y)$ some nodes labeled In, they necessarily are on the same path μ from a node n_0 to the *root*. The children of n_0 are leaves of the cotree or are labeled $InOut$ or Out since there is no node labeled $Failure$.
$\Longleftarrow$ Let us assume $cot(G_Y)$ contains a node n labeled $Failure$. If the path μ of nodes labeled In is empty, according to Property 4.b, all the ancestors of n must be labeled $Failure$, including the root of $cot(G_Y)$. But the root cannot be labeled $Failure$ because $L_{root} \cup \{v\}$ is a module of $G_{Y\cup\{v\}}$. So, there is a contradiction.

If $\mu = n_0, n_1, \ldots n_l = root$ is not empty: There is no node labeled $Failure$ in the subtrees rooted by children of n_0 because according to Property 4.b, ancestors of a node labeled $Failure$ must be labeled In or $Failure$. If we assume there is a node n labeled $Failure$ in the subtree rooted by n_i, $1 \leq i \leq l$, such that n does not belong to the subtree rooted by n_{i-1}, then the nearest common ancestor of n_{i-1} and n is different from n_{i-1} and n, which contradicts Property 4.c. So, there is no node n labeled $Failure$ in $cot(G_Y)$. □

If we consider the set of modules ordered by inclusion, Proposition 5 and Characterization 2 can be rephrased in the following way:

Consequence: *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$.*
$G_{Y \cup \{v\}}$ is a cograph if and only if nodes labeled In form an interval of the inclusion order of strong modules which includes the root and such that the elements covered by this interval are labeled $InOut$ or Out.

This property yields an $O(n^2)$ algorithm for cograph recognition, but in the following sections it will be seen that the Modular Labeling can be simplified in a way to obtain a linear algorithm.

Algorithm 1: Incremental-Cograph-Recognition
begin
  $Y \longleftarrow \emptyset$
  $cot \longleftarrow \emptyset$
  $P_4 \longleftarrow \emptyset$
  while $X \setminus Y \neq \emptyset$ **and** $P_4 = \emptyset$ **do**
    choose a vertex v in $X \setminus Y$
    $L \longleftarrow$ LABELING(cot,v)
    if CONDITION(cot,L) *is satisfied* **then**
      $Y \longleftarrow Y \cup \{v\}$
      UNMARK(cot)
      $cot \leftarrow$ UPDATE(cot,v)
    else
      $P_4 \longleftarrow$ FINDP4(cot,*nodes inducing a failure in* CONDITION)
end

3 A New Formulation of CPS-Algorithm

To design an incremental cograph recognition algorithm, the idea is to use the sketch presented on Algo. 1, with function LABELING computing the partition of nodes induced by the Modular Labeling of $cot(G_Y)$, and a function CONDITION to find whether this labeling follows the condition of Proposition 5. Unfortunately the complexity of these two functions would be too high, since to obtain

the expected linear time complexity for the whole algorithm, it is necessary to compute them in $O(|N(v)|)$. So, the reformulation of *CPS-Algorithm* presented here computes only nodes labeled In or $Failure$ by the Modular Labeling.

In the following subsections functions CPS-Labeling and CPS-Condition of *CPS-Algorithm* are presented, but modifications are introduced in a way to find efficiently a P_4 if there exists a $Failure$ node. Moreover, it will be shown that the condition checked by CPS-Condition on the Labeling computed by CPS-Labeling is equivalent to the property of the Modular Labeling presented in Proposition 5.
For complexity issues, the *CPS-Labeling* of nodes of $cot(G_Y)$ is computed by CPS-Labeling as follows:

U (for UnMarked): default case (at the beginning of CPS-Labeling, all nodes are Unmarked)
C (for Completely Marked):
- a leaf l is Completely Marked if and only if $l \in N(v)$.
- an internal node n is Completely Marked if and only if all its children are Completely Marked.

P (for Path): an internal node keeps the P mark iff it cannot be marked C and if there exists a node m such that
- m keeps the C mark,
- and $Father(m)$ cannot be marked C,
- and $Father(Father(m)) = n$ or $Father(Father(Father(m))) = n$.

B (for BeginPath): a node keeps the B mark if it cannot keep the C mark and it cannot keep the P mark, and it has at least one child marked C.

For complexity issues, the two following requirements are also needed:

- an $O(1)$ access from each node of $cot(G_Y)$ to some arbitrary leaf of its subtree,
- Check in $O(1)$ if a given node admits a Completely Marked (respectively UnMarked) child.

This can be achieved with the following data structure:
type T_Node **record**
- father, next, prev, F_child, L_child, Leaf : Ptr_Node;
- mark : (U,C,P,B);
- node_type : (Series, Parallel, leaf);
- vertex : T_Vertex;

endrecord

The direct access from each node of the cotree to some arbitrary leaf of its subcotree can be easily computed without additional cost in the Update function.

3.1 The Labeling of cot

The CPS-Labeling function presented in Algo. 2 takes $cot(G_Y)$ (all nodes are UnMarked), and a new vertex v belonging to $X \setminus Y$. It proceeds in two steps:

Algorithm 2: CPS-Labeling
Data: $v \in X \setminus Y$
$cot(G_Y)$
Result: L: list of nodes of $cot(G_Y)$ marked B or P
L_C: list of nodes of $cot(G_Y)$ marked C

```
begin
    L ⟵ ∅; L_C ⟵ ∅
    Heap ⟵ N_G(v) ∩ cot(G_Y)
    {first step:}
    while Heap ≠ ∅ do
        y ⟵ remove head from Heap
        L_C ⟵ L_C ∪ {y}
        y.mark ← C
        if y ≠ root then
            n ⟵ father of y in cot(G_Y)
            put y as the last child of n
            if n ∉ L then L ⟵ L ∪ {n}
            if all children of n are in L_C then Heap ⟵ Heap ∪ {n}
    {second step:}
    for n ∈ L do
        if n ∈ L_C then  L ⟵ L \ {n}
        else
            if n.father ≠ root then
                n.father.father.mark ← P
                if n.father.father ∉ L then L ⟵ L ∪ {n.father.father}
                if n.father.father ≠ root then
                    put n.father.father as the first child of
                    n.father.father.father
                put n.father as the first child of n.father.father
            if n ≠ root then
                n.father.mark ← P
                if n.father ∉ L then L ⟵ L ∪ {n.father}
                put n as the first child of n.father
            if n.mark = U then
                n.mark ← B
end
```

First, it computes each node of the cotree that need the C mark, putting the C mark to each leaf y of the cotree which belongs to the neighbourhood of v in G, and recursively mark C a node whose children are all marked C.

Secondly, CPS-LABELING takes the Unmarked fathers n of Completely Marked nodes, and gives them the B mark if they have not the P mark; and gives the P mark to their father and grandfather. When all these nodes are processed the labeling is complete.

CPS-LABELING returns the list L of nodes marked $BeginPath$ or $Path$, and the list L_c of Completely Marked nodes. With this two lists the UNMARK function which unmarks all nodes, can be computed in linear time.

As a by product, at the end of the function, for each internal node of the cotree one can find in its children list: first the child with mark BeginPath or Path, then the UnMarked children and finally the Completely Marked children.

The time complexity of CPS-LABELING is clearly in $O(|N(v)|)$.

3.2 The CPS-Condition

Let us consider the relationships between CPS-Labeling and the Modular Labeling. Proposition 6 shows that the following condition on CPS-LABELING is satisfied if and only if the condition on Modular Labeling stated in Proposition 5 is satisfied.

CPS-Condition *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$.*
The set of nodes of $cot(G_Y)$ marked B or P by CPS-LABELING *is a path $\mu = n_0, \ldots, n_l = root$, eventually empty, such that:*

1. *n_0 is marked B,*
2. *n_i $i \neq 0$ is marked P,*
3. *for all i $1 \leq i \leq l$,* **either** *n_i is a Parallel node and every child $c \neq n_{i-1}$ of n_i is such that L_c contains only nodes non adjacent to v,* **or** *n_i is a Series node and every child $c \neq n_{i-1}$ of n_i is such that L_c contains only nodes adjacent to v.*

We can remark that if nodes marked B or P computed by CPS-LABELING satisfy the CPS-Condition, then these nodes are exactly those given in the following formal definition:

- an internal node n is marked as BeginPath if and only if all children of n are either Completely Marked or UnMarked and both marks are present in the children set of n.
- an internal node n is marked as Path if and only if it admits a child marked Path or BeginPath.

Proposition 6. *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$. We have the following equivalence:*
The CPS-Condition is satisfied if and only if In nodes of the Modular Labeling of $cot(G_Y)$ form a path $\mu = n_0, \ldots, n_l = root$ eventually empty, such that the children of n_0 are leaves or nodes labeled $InOut$ or Out and such that there are at least a children labeled $InOut$ and a children labeled Out.

Algorithm 3: CPS-Condition
Data: L: the list of nodes of $cot(G_Y)$ marked B or P by CPS-Labeling
Result: *error* equals FALSE if CPS-Condition is satisfied,
TRUE otherwise

```
begin
    error ← False
    if L ≠ ∅ then
        n ← take a node marked B in L
        L ← L \ {n}; l ←| L |; k ← n; count ← 0
        if n ≠ root then
            repeat
                put n as the first child of n.father
                n ← n.father
                count ← count + 1
                if n.node_type = Series then
                    error ← (n.F_child.next.mark ≠ C)
                else
                    error ← (n.L_child.mark ≠ U)
                error ← (error or n ∉ L or n.mark = B)
            until n = root or error
        error ← (error or count ≠ l)
        if k.node_type = n.node_type then k ← k.father
    return error
end
```

Proof. $\Longrightarrow$ Let L be the set of nodes marked B or P by CPS-Labeling and satisfying the CPS-Condition.

Nodes in L are the nodes of $cot(G_Y)$ which are not marked C or U by CPS-Labeling. So they have at least a leaf adjacent to v and a leaf non adjacent to v in their subtree. So, they are the nodes labeled In or $Failure$ in Modular Labeling. Let us show that none of these nodes can be labeled $Failure$. If we assume n_0 is labeled $Failure$, then there exists a node n_i of μ which does not satisfy the CPS-Condition, a contradiction. So, n_0 is In. According to Property 4.a, n_i is also labeled In for all i, $1 \leq i \leq l$.

$\Longleftarrow$ - We first show that n_0 is labeled B by CPS-Labeling. n_0 is marked In, so it cannot be marked C. Considering the fact that a node n labeled Out or $InOut$ satisfies "either all nodes in L_n are adjacent to v, or any", then all children of n are marked C or U. So, n_0 cannot be marked P. n_0 is the lower node labeled In implies there exists at least a child c of n_0 marked C. So, n_0 is marked B.

- Then, we show that for each $n_i \in \mu$, $i \neq 0$, if n_i is a Series node, all its children $c \neq n_{i-1}$ are marked C, or if n_i is a Parallel node, all its children

$c \neq n_{i-1}$ are marked U. If n_i is a Series node. n_i is labeled In then all elements of $L_{n_{i-1}}$ are adjacent to all elements of L_c, for all children c of n_i, $c \neq n_{i-1}$. So, v must be adjacent to all elements of L_c, for all children c of n_i, $c \neq n_{i-1}$. So nodes c are marked C. If n_i is a Parallel node. n_i is labeled In then there is no element of $L_{n_{i-1}}$ adjacent to any element of L_c, for all children c of n_i, $c \neq n_{i-1}$. So, v is not adjacent to any element of L_c, for all c child of n_i, $c \neq n_{i-1}$. So nodes c are marked U.

- Then, we show by recurrence on i that each $n_i \in \mu$, $i \neq 0$ is marked P by CPS-LABELING algorithm. There is a child c of n_0 marked C and n_0 cannot be marked C. $n_1 = Father(Father(c))$, then n_1 is marked P. Considering a node n_k $2 \leq k \leq l$, if n_k is a Series node, n_{k-2} is a Series node. n_{k-2} has a child c marked C. n_{k-2} is not marked C, and $n_k = Father(Father(Father(c)))$. So, n_k is marked P. If n_k is a Parallel node, n_{k-1} is a Series node. n_{k-1} has a child c marked C. n_{k-1} is not marked C, and $n_k = Father(Father(c))$. So, n_k is marked P.

- Nodes of μ are the only ones marked P or B by CPS-LABELING algorithm, because according to the previous proofs, a partition of nodes of $cot(G_Y)$ is obtained with nodes of μ marked B or P, and all other nodes marked C or U. □

Corollary 7. *Let $G = (X, E)$ be an undirected graph, $Y \subseteq X$, $cot(G_Y)$ the cotree of G_Y and v a vertex of $X \setminus Y$.*
$G_{Y \cup \{v\}}$ is a cograph if and only if the CPS-Condition is satisfied by nodes marked B or P by CPS-LABELING.

So the function CPS-CONDITION can only check the CPS-Condition. This function is presented by Algo. 3.

Corollary 8. *Let G_Y be a cograph and $v \in X \setminus Y$, CPS-Algorithm recognizes if $G_{Y \cup \{v\}}$ is a cograph in $O(|N(v)|$. This yields a linear time complexity algorithm to recognize if an arbitrary graph is a cograph.*

3.3 UPDATE Function

The UPDATE procedure is not presented here because it is the same than in *CPS-Algorithm*. However using the informations collected by our algorithm it is very easy to present the idea of this algorithm.

The CPS-Condition is satisfied, so we have a path $\mu = n_0, \ldots, n_l = root$ of nodes such that v must be in their subtree. In $cot(G_{Y \cup \{v\}})$, v must be a leaf of the subtree rooted by n_0. Children c of n_0 are divided in two categories: nodes c such that L_c contains only neighbours of v, and nodes c such that L_c contains no neighbour of v. If the Series or Parallel property of n_0 is also considered, all the information needed to perform this updating is known (adjacency between v and the existing nodes). So, the updating is performed adding at most three new nodes to $cot(G_Y)$: v as a new leaf and at most two new internal nodes, to respect adjacencies.

Algorithm 4: FindP4
Data: $cot(G_Y)$
k, n: T_node
Result: P_4: a list of four vertices
begin
  $P_4 \leftarrow \emptyset$
  $l_1 \leftarrow$ FindMarkedLeaf(cot,k)
  if $k.mark = P$ **and** $k.node\_type = Parallel$ **then**
    $w \leftarrow$ the second child of n
    $l_2 \leftarrow$ FindUnMarkedLeaf(cot,w)
  else
    $l_2 \leftarrow$ FindUnMarkedLeaf(cot,k)
  if $n.node\_type = P$ **then**
    if $n.L\_child.mark = C$ **then**
      $l_3 \leftarrow$ FindMarkedLeaf(cot,$n.L\_child$)
    else
      $w \leftarrow$ the second child of n
      $l_3 \leftarrow$ FindMarkedLeaf(cot,w)
    $P_4 \leftarrow [l_3, v, l_1, l_2]$ (case 1 - Fig. 2)
  else
    $w \leftarrow$ the second child of n
    $l_3 \leftarrow$ FindUnMarkedLeaf(cot,w)
    $P_4 \leftarrow [l_2, l_3, l_1, v]$ (case 2)
end

4 Find a P_4

In this section we describe the function FindP4 (Algo. 4) called if the *CPS-Labeling* of $cot(G_Y)$ does not satisfy the CPS-Condition.

FindP4 receives as input a cotree and two nodes: k a node marked B which satisfies the CPS-Condition, and n which does not satisfy the CPS-Condition. It returns four vertices inducing a P_4. Using Proposition 5 and 6, we know that this P_4 exists. The problem is just to have an efficient way to find the three "good" vertices in $cot(G_Y)$. We need to find efficiently a Completely Marked leaf and (or) an UnMarked leaf in a subtree of $cot(G_Y)$ rooted by a node n. This explains we describe two extra functions:

1. FindMarkedLeaf(*cot: cotree, n: T_node such that* $n.mark \neq U$) : T_node;
2. FindUnMarkedLeaf(*cot: cotree, n: T_node such that* $n.mark \neq C$) : T_node;

Using this two functions, the algorithm is an implementation of the *only if* part of the proof of Characterization 2. In fact, the node k marked B which satisfies the CPS-Condition, is such that L_k and $L_k \cup \{v\}$ is not a module of $G_{Y \cup \{v\}}$.

The node n which does not satisfy the CPS-Condition, is the node which allows to detect the state of k.

For the computation of l_2, the case k is marked $Path$ and k is a Parallel node insures that k is the nearest ancestor of nodes l_1 and l_2 computed by FindMarkedLeaf and FindUnMarkedLeaf respectively.

There are some additional subcases for case 1 (n is a Parallel node and k is a Series node), if n contains at least one Completely Marked child, l_3 is obtained from this child. Otherwise n has two children marked B or P. The first is the child of n belonging to the path between $root$ and k, and the second belongs to another path between $root$ and a node k' of L ($k' \neq k$) previously processed. So, the second child of n satisfies the CPS-Condition. It has a marked leaf l_3 in its subcotree.

It can also be remarked, that if the function CPS-Condition fails with two Series nodes or two Parallel nodes n and k, we can find not only a P_4, but a undecomposable subgraph with fives vertices.

Algorithm 5: FindMarkedLeaf
Data: $cot(G_Y)$
n: T_node
Result: a T_node
begin
  if $n.node\_type = leaf$ **then** return n
  if $n.L\_child.mark = C$ **then** return $n.F\_child.Leaf$
  else return $n.F\_child.L\_child.Leaf$
end

Algorithm 6: FindUnMarkedLeaf
Data: $cot(G_Y)$
n: T_node
Result: a T_node
begin
  if $n.node\_type = leaf$ **then** return n
  while $n.F\_child.Leaf.mark = C$ **do**
    $n \leftarrow n.F\_child.Leaf$
    while $n.mark = C$ **do**
      $n \leftarrow n.father$
  return $n.F\_child.Leaf$
end

4.1 FindMarkedLeaf and FindUnMarkedLeaf

FindMarkedLeaf (Algo. 5) returns any marked leaf of the subcotree rooted by the node n. Since this procedure has been called with n (a node of the cotree), then either the last child of n is Completely Marked or no child of n is Completely Marked. In the first case, we just need to take any leaf of the subcotree rooted by the last child of n. Otherwise the first child of n keeps a B or P mark (it have at least a Completely Marked and an UnMarked child) , so we just need to return any marked leaf of the subcotree rooted by this new node.
FindUnMarkedLeaf (Algo. 6) returns any unmarked leaf of the subcotree rooted by the node n which is not Completely Marked.

4.2 Complexity of FindP4 Function

First one can notice that the FindMarkedLeaf function can be done in $O(1)$. The function FindUnMarkedLeaf can be done in at most the number of nodes containing the C mark in the subcotree rooted by n. It yields that the procedure FindP4 can be done in the number of Completely Marked nodes of the cotree. Since the number of Completely Marked nodes of the cotree is at most two times the number of neighbour of the new vertex v in the cotree, FindP4 can be done in linear time.

5 Conclusion

To prove our reformulation of *CPS-Algorithm* we have processed by transformation from a structural characterization of cographs in terms of modular decomposition. This implies that for any generalization of cographs, namely directed cographs [Cou93], labeled cographs [EGMS94], such a result will be available: a linear recognition algorithm that exhibits a minimal undecomposable structure in case of failure.

Furthermore, we suspect that our framework can be adapted to any class of graphs which are totally decomposable by some notion of tree-decomposition, and that the linearity of the algorithm can be preserved, *c.f.* Arnborg *et al.* [ALS] and Courcelle [Cou88] works.

References

[ALS] Stephan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340.

[CE80] W. H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canad. J. Math*, 32(3):734–765, 1980.

[CH93a] A. Cournier and M. Habib. An efficient algorithm to recognize prime undirected graphs. In E. W. Mayr, editor, *Lectures notes in Computer Science, 657.Graph-Theoretic Concepts in Computer Science. 18th International Workshop, WG'92. Wiesbaden-Naurod, Germany, June 1992. Proceding*, pages 212–224. Springer-Verlag, 1993.

[CH93b] A. Cournier and M. Habib. An $o(n + m \log n)$ algorithm for substitution decomposition. In *CODIGRAF'93*. Universitat Autònoma de Barcelona, 1993.

[CH94] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In S. Tison, editor, *Lectures notes in Computer Science, 787. Trees in Algebra and Programming-CAAP'94*, pages 68–84. Springer-Verlag, April 1994. 19th International Colloquium, Edinburgh, U.K., April 1994. Proceedings.

[Cha47] A. Chatelet. Algèbre des relations de congruence. *Ann. Sci. de l'école Normale Supérieure*, (66):332–368, 1947.

[Cou88] Bruno Courcelle. The monadic second order logic of graphs. iii. tree-width, forbidden minors, and complexity issues. Technical Report I-8852, Université de Bordeaux I, 1988.

[Cou93] A. Cournier. *Sur Quelques Algorithmes de Décomposition de Graphes*. PhD thesis, Université Montpellier II, 161 rue Ada, 34392 Montpellier Cedex 5, France, février 1993.

[CPS85] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM journal of computing*, (14):926–934, november 1985.

[EGMS94] A. Ehrenfeucht, H. N. Gabow, R. M. McConnel, and S. J. Sullivan. An $o(n^2)$ divide-and-conquer algorithm for prime tree decomposition of two-structures and modular decomposition of graphs. *Journal of Algorithm*, 16(2):283–294, March 1994.

[Hab81] M. Habib. *Substitution des structures combinatoires, théorie et algorithmes*. PhD thesis, Université Pierre et Marie Curie (Paris VI), 1981.

[HM79] M. Habib and M. C. Maurer. On the x-join decomposition for undirected graphs. *Discrete Applied Math*, (3):198–207, 1979.

[Kel85] D. Kelly. Comparability graphs. In I. Rival, editor, *Graphs and Orders*, pages 3–40. D. Reidel Pub. Comp., 1985.

[Ler71] H. Lerchs. On cliques and kernels. Technical report, Dept. of Comp. Sci. University of Toronto, 1971.

[Ler72] H. Lerchs. On the clique-kernel structure of graphs. Technical report, Dept. of Comp. Sci. University of Toronto, 1972.

[MS94] R. M. McConnell and J. Spinrad. Linear-time modular decomposition and efficient transitive orientation of undirected graphs, 1994. Proc. of the fifth Annual ACM-SIAM Symposium of Discrete Algorithms.

[Sei74] S. Seinsche. On a property of the class of n-colorable graphs. *JCT*, B(16):191–193, 1974.

[Spi83] J. Spinrad. Transitive orientation in $o(n^2)$ time. In *ACM Symposium on theory of complexity*, 1983.

[Spi92] J. Spinrad. P4-trees and substitution decomposition. *Discrete Applied Mathematics*, (39):263–291, 1992.

[ST94] K. Simon and P. Trunz. A cleanup on transitive orientation. In V. Bouchitté and M. Morvan, editors, *Lecture Notes in Computer Science*, number 831, pages 59–85. Springer-Verlag, 1994. Proceedings of Internationnal Workshop ORDAL'94 - Lyon.

This article was processed using the LaTeX macro package with LLNCS style