Available online at www.sciencedirect.com
SCIENCE DIRECT®
Discrete Applied Mathematics 145 (2005) 183–197
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

A simple linear time algorithm for cograph recognition

Michel Habib, Christophe Paul

CNRS, Université Montpellier 2, LIRMM, 161 rue Ada, 34392 Montpellier Cedex 2, France

Received 22 March 2002; received in revised form 19 November 2002; accepted 16 January 2004

Abstract

In this paper, we describe a new simple linear time algorithm to recognize cographs. Cographs are exactly the P_4-free graphs (where P_4 denotes the path with 4 vertices). The recognition process works in two steps. First, we use partition refinement techniques to produce a factorizing permutation, i.e., an ordering of the vertices in which the strong modules appear consecutively. Then a very simple test algorithm is provided to check whether the given graph is a cograph, using a single sweep of the permutation obtained in the first step.

Keywords: Modular decomposition; Graphs; Algorithms

1. Introduction

The class of cographs has been intensively studied since their definition by Seinsche [21]. Cographs are exactly the P_4-free graphs. It is well known that any cograph has a canonical tree representation, called the cotree. This tree decomposition scheme of cographs is a particular case of the modular decomposition [9] that applies to arbitrary graphs. Indeed, algorithm which computes in linear time the modular tree decomposition of an arbitrary graph, can also recognize cographs without additive complexity cost. In 1994, linear time modular decomposition algorithms were designed independently by Cournier and Habib [5] and by McConnell and Spinrad [18]. More recently, Dahlhaus et al. [7] proposed a simpler algorithm. Unfortunately, because they build the decomposition tree, all these algorithms are either complicated or need to maintain complicated data structures. Therefore, to find a simple modular decomposition algorithm is still an open problem.

The design of a new recognition algorithm for cographs[1] is also an interesting problem. The first linear time algorithm by Corneil et al. [4] incrementally builds a cotree, starting from a single vertex and adding a new vertex at each step of the computation. The complication of this algorithm is mainly due to the linear time complexity. In fact, each time a vertex x is added, the cotree has to be updated using at most $O(|N(x)|)$ elementary operations, where $N(x)$ denotes the neighborhood of x, which is far from being obvious. It should be mentioned that Dahlhaus [6] proposed a nice parallel cograph recognition algorithm.

The new algorithm we propose in this paper is not incremental, and instead of building directly the cotree, it first computes a special ordering of the vertices, namely a factorizing permutation, using the very efficient partition refinement techniques via two elementary refinement rules. In our point of view, the bottleneck with respect to simplicity for all these algorithms is the decomposition tree computation. In 1997, Capelle [2] introduced the concept of factorizing permutation that can roughly be seen as an ordering of the leaves of the decomposition tree.

E-mail address: habib@lirmm.fr (M. Habib), paul@lirmm.fr (C. Paul).

[1] Recently, a simple Lex-BFS based cograph recognition algorithm, using the duality on G and $\overline{G}$, has been proposed [1].

0166-218X/$ - see front matter © 2004 Published by Elsevier B.V.
doi:10.1016/j.dam.2004.01.011

Our algorithm really avoids complicated data structures because it never computes the decomposition tree. It is a two step algorithm. The first step only computes a permutation of the vertices, that is a factorizing permutation if the input graph is a cograph. The second step tests the result: the computed permutation has a certain property iff the input graph is a cograph. It roughly consists of a left to right scan of the computed permutation. Both steps of the algorithm need linear time and the main step is based on the powerful paradigm of partition refinement and vertex splitting; thus this algorithm can be included in a wide pool of graphs algorithms including modular decomposition, transitive orientation, interval graph recognition algorithms. The interested reader can refer to [10,11] for more examples. In [8,11] a $O(n + m \log n)$ version of the first step was proposed. Our algorithm can also be seen as the first step towards a simple linear modular decomposition algorithm.

Section 2 presents in more detail the structure of cographs and some definitions. The algorithm that computes a factorizing permutation of a cograph is explained in Section 3. Data-structures and complexity analysis are discussed in Section 4. Finally, the recognition test is detailed in Section 5.

2. Definitions

2.1. Cographs and factorizing permutations

Throughout this paper we consider only finite undirected simple (with no multiple edges) graphs.

Definition 1. The class of cographs is the smallest class of graphs containing the single vertex graph and closed under series and parallel composition.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two arbitrary graphs. A graph $G = (V, E)$ is the *parallel composition* of G_1 and G_2 if $V = V_1 \cup V_2$ and $E = E_1 \cup E_2$. A graph G is the *series composition* of G_1 and G_2 if $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup \{(x_1, x_2)$ s.t. $x_1 \in E_1$ and $x_2 \in E_2\}$.

Therefore, to each cograph can be associated several composition formulas using series and parallel operations. Such a formula can be written as a tree whose leaves are the vertices of the graph, and the internal nodes are labeled *series* or *parallel* depending of their corresponding operation. Among those tree-decompositions, for each graph there exists a canonical one, the so-called *cotree* [4] in which on every path, the labels series and parallel strictly alternate. Fig. 1 shows an example of a cograph and its cotree.

Remark 2. In a cotree, the internal nodes of a path from a leaf to the root are alternatively labeled series and parallel.

Remark 3. In a cograph, two vertices x and y are adjacent iff their least common ancestor (denoted by $\mathrm{LCA}(x, y)$) in the cotree is a series node.

Definition 4. Let us denote by $\leqslant_T$ the usual partial order of the nodes of T (i.e., $n_1 \leqslant_T n_2$ iff n_1 is a descendant of n_2 in T. Equality holds when $n_1 = n_2$.)

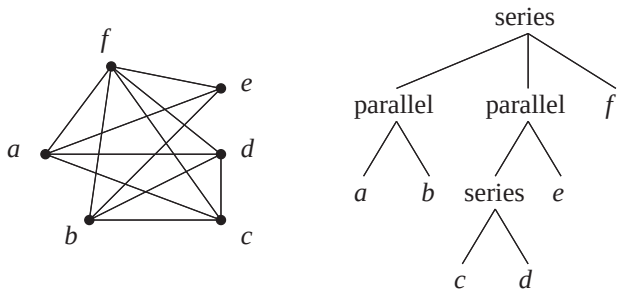

Fig. 1. A cograph and its cotree.

Notation 1. Let n be an internal node of the cotree T of a given cograph G. Let us denote by T_n the subtree of T rooted at n.

Let M be the set of vertices that are leaves of some subtree T_n for some n. It follows from the second remark that any pair of vertices in M have the same neighborhood outside M; such a set is called a *module* and plays an important role in the cograph recognition algorithm. More formally:

Definition 5. A set of vertices M of a graph G is a *module* iff for any z and t in M, $N(z)\backslash M = N(t)\backslash M$. A module M is a *strong module* iff for any module M' either $M' \subseteq M$ or $M \subseteq M'$ or $M \cap M' = \emptyset$.

Remark 6. For any strong module M, there is an internal node n of the cotree T such that M is exactly the set of leaves of T_n.

The algorithm we present computes a *factorizing permutation*, that can be seen as a postorder traversal of the leaves of the cotree. Let us define this permutation more precisely.

Definition 7. A *factorizing permutation* of a graph $G = (V, E)$ is a permutation σ of the vertex set V such that the vertices of any strong module of G appears consecutively in σ.

In particular if G is a prime graph (i.e., has no nontrivial module) then any permutation of the vertices is a factorizing permutation. Let us now examine the relationships between cotrees and factorizing permutations. Although a given cograph G has a unique cotree $T(G)$, a cotree admits several plane representations (drawings) in which root is on top and where the left-right ordering of the children of each node is fixed. We first need a definition.

Definition 8. Let x, y, z be three different vertices. Then *x separates y and z* if either $xy \in E$ and $xz \notin E$ or $xy \notin E$ and $xz \in E$.

Lemma 9. *Factorizing permutations are in one-to-one correspondence with plane representations of a cotree.*

Proof. Let A be plane representation of a cotree, the left-right ordering of the leaves yields a factorizing permutation. Let us prove the converse by induction on the size of G. If G has only one vertex the result is obvious. Now let σ be a factorizing permutation of G. If G is prime its cotree $T(G)$ has only one internal node and the result is also obvious. Else G admits a minimal non trivial strong module M. By definition M defines a factor of σ. The result is obtained by contracting M to a single vertex and applying induction. □

Most of the proofs of this paper can easily been understood geometrically when considering the plane representation associated with a given factorizing permutation. Furthermore, for cographs since adjacency between two vertices is completely determined by their least common ancestor in the cotree, we can deduce some necessary conditions.

Corollary 10. *Let x, y, z be distinct vertices appearing in that order in a factorizing permutation σ. If x separates y and z then* $\mathrm{LCA}(x, y) <_T \mathrm{LCA}(y, z)$.

Proof. Let us consider $A(\sigma)$ the plane representation associated with σ, then z is a leaf of $A(\sigma)$ that lies right to y which lies right to x. Trivially least common ancestors are the same in the cotree and in $A(\sigma)$. Let us consider the unique path in $A(\sigma)$ joining y to the root r of $A(\sigma)$. $\mathrm{LCA}(x, y)$ and $\mathrm{LCA}(y, z)$ are two nodes of this path. If $\mathrm{LCA}(x, y) \geqslant_T \mathrm{LCA}(y, z)$ this implies that $\mathrm{LCA}(x, y) = \mathrm{LCA}(x, z)$ which contradicts the fact that x separates y and z. □

Corollary 11. *Let x, y, z be distinct vertices appearing in that order in a factorizing permutation σ such that $xy \notin E$ and $yz \in E$. Then $xz \in E$ iff* $\mathrm{LCA}(x, y) <_T \mathrm{LCA}(y, z)$ (see Fig. 2).

Proof. If $xz \in E$, then x separates y and z and Corollary 10 applies. If $xz \notin E$, then z separates x and y. Same proof than for Corollary 10 shows that $\mathrm{LCA}(x, y) >_T \mathrm{LCA}(y, z)$. □

Corollary 12. *Let t, x, y, z be distinct vertices appearing in that order in a factorizing permutation σ and $tx \notin E$, $xy \in E$ and $xz \in E$. If t separates y and z, then necessarily $ty \notin E$ and $tz \in E$* (see Fig. 2).

Proof. Suppose the contrary: $ty \in E$ and $tz \notin E$. Corollary 11 applied to triples t, x, y and t, x, z, respectively, shows that $\mathrm{LCA}(x, t) <_T \mathrm{LCA}(x, y)$ and $\mathrm{LCA}(x, z) <_T \mathrm{LCA}(x, t)$. It follows that $\mathrm{LCA}(x, z) <_T \mathrm{LCA}(x, y)$ which would lead to a crossing in $A(\sigma)$, a contradiction. □

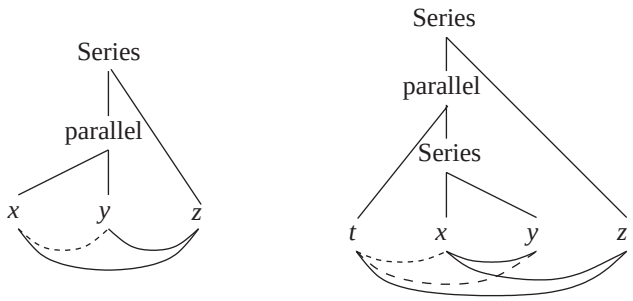

Fig. 2. Illustrations of Corollaries 11 and 12.

The proof of lemma 9 leads to an algorithm which computes $A(\sigma)$ from σ. In fact this bijection for cographs between plane representations of cotrees and factorizing permutations can be generalized to a bijection between plane representations of the unique modular tree decomposition of a given graph and its factorizing permutations. In this general setting Capelle et al. [3] obtained a linear algorithm to compute $A(\sigma)$ from σ. Indeed in many applications of the modular decomposition, the factorizing permutation is enough [14,2,12]. For the particular case of cographs a very simple algorithm is provided in Section 5.

2.2. Partition refinement and vertex splitting

A *partition* $\mathscr{P}$ of a set V is a set of disjoint subsets of V, called *parts* of $\mathscr{P}$, $\{\mathscr{X}_1, \ldots, \mathscr{X}_k\}$ whose union is exactly V. Let $\mathscr{P}$ and $\mathscr{Q}$ be two partitions of V. If for each part $\mathscr{X}$ of $\mathscr{P}$ there exists a part $\mathscr{Y}$ of $\mathscr{Q}$ such that $\mathscr{X} \subseteq \mathscr{Y}$, then we say that $\mathscr{P}$ is *thinner* than $\mathscr{Q}$ (or $\mathscr{Q}$ is *coarser* than $\mathscr{P}$).

The algorithms we develop here deal with *ordered partitions*. Let $\mathscr{P}$ be the ordered partition $[\mathscr{X}_1, \ldots, \mathscr{X}_k]$ of the set V. And let $u \in \mathscr{X}_i$ and $v \in \mathscr{X}_j$ be two arbitrary elements of different parts. Then $u <_{\mathscr{P}} v$ iff $i < j$. For sake of simplicity, we will also say that $X_i <_{\mathscr{P}} X_j$.

Let $\mathscr{P}$ and $\mathscr{Q}$ be two partitions of V, then $\mathscr{P}$ is *compatible* with $\mathscr{Q}$, denoted by $\mathscr{P} \preccurlyeq \mathscr{Q}$ (and $\mathscr{P} \prec \mathscr{Q}$ if $\mathscr{P} \preccurlyeq \mathscr{Q}$ and $\mathscr{P} \neq \mathscr{Q}$) iff :

- $\mathscr{P}$ is thinner than $\mathscr{Q}$ and,
- let x and y be two elements of V such that $x <_{\mathscr{P}} y$, then $x \leqslant_{\mathscr{Q}} y$.

Clearly $\preccurlyeq$ is a partial ordering on the partitions of a given ground set V.

Definition 13. A set S *strictly intersects* another set S' iff $S \cap S' \neq \emptyset$ and $S' - S \neq \emptyset$.

It should be noticed that the strict intersection relation is not symmetric: set S can be included in set S'. Refining a partition $\mathscr{P}$ with a pivot set S consists in replacing each part $\mathscr{X} \in \mathscr{P}$ by $[\mathscr{X}_b, \mathscr{X}_a]$ (in that order, recall we deal with ordered partitions) where $\mathscr{X}_a = \mathscr{X} \cap S$ and $\mathscr{X}_b = \mathscr{X} \setminus S$. The new partition obtained using this refinement operation will be denoted by *Refine*$(\mathscr{P}, S)$. A partition $\mathscr{P}$ is *stable* with respect to S if S strictly intersects no part of $\mathscr{P}$ (i.e., $\mathscr{P} = \mathit{Refine}(\mathscr{P}, S)$). A set S *strictly refines* a partition $\mathscr{P}$ if *Refine*$(\mathscr{P}, S) \prec \mathscr{P}$.

In the following we deal with partitions of the vertex set of a graph, and we use neighborhood sets as pivot sets to refine these partitions. When the neighborhood of a vertex x is used as a pivot set to refine the partition, then x is called a *pivot*.

A vertex x *splits* a part $\mathscr{C}$ if $N(x)$ strictly intersects $\mathscr{C}$. Then x is called a *splitter for* $\mathscr{P}$. Notice that a vertex x is a splitter for $\mathscr{P}$ iff x separates at least two vertices of some part of $\mathscr{P}$.

Starting from initial partition $[V]$ of the vertex set of a graph $G = (V, E)$ and using vertex splitting operations the following algorithms will produce a final partition $[\{x_1\}, \ldots, \{x_n\}]$ which can be considered as an ordering or a permutation of the vertices.

3. Computing a factorizing permutation

In this section, the graphs we consider are supposed to be cographs. So the existence of a cotree is assumed. The first algorithm describes how the basic ideas for the computation of a factorizing permutation can be applied. The second algorithm is a refinement of the first: based on two new properties it can be implemented in linear time.

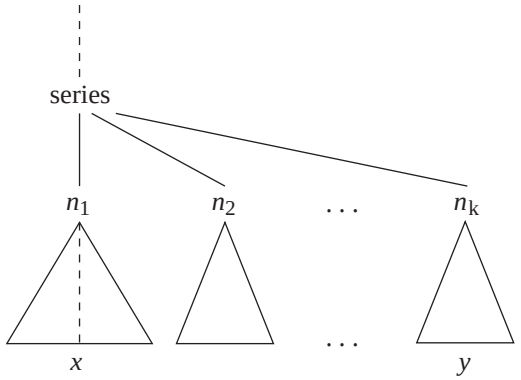

Fig. 3. If n is a series node, then $T_{n_2}, \ldots, T_{n_k}$ are inserted on the right of T_{n_1}.

3.1. A kind of "brute force" algorithm

Lemma 14 (*Habib et al. [11]*). *Let x be an arbitrary vertex of a cograph, then there exists a factorizing permutation compatible with partition $\mathscr{P} = [\overline{N}(x), \{x\}, N(x)]$.*

Proof. Let T be the cotree of G. As noticed in Remark 3, two vertices are adjacent in G if and only if their least common ancestor (LCA) in T is a series node. The set $\{\mathrm{LCA}(x, y)|$ s.t. $y \neq x\}$, is exactly the set of all ancestors of x in the cotree T. Let n be one of these nodes and let $n_1, \ldots, n_k$ be its sons. Without loss of generality, we can assume that x is a leaf of T_{n_1}. Then insert the subtrees $T_{n_2}, \ldots, T_{n_k}$ on the right of T_{n_1} iff n is a series node (see Fig. 3). Applying downward this rule to any internal node on the path between x and the root of T, produces a drawing of T where the leaf corresponding to a given vertex y is on the right of x iff x and y are adjacent. □

This lemma will be used as a refinement rule in the algorithms as follows:

Refinement rule 1 (*Initialization rule*). Let $\mathscr{C}$ be a partition part, then pick an arbitrary vertex $x \in \mathscr{C}$, hereafter called the *origin* of $\mathscr{C}$, and refine $\mathscr{C}$ into $[\overline{N}(x) \cap \mathscr{C}, \{x\}, N(x) \cap \mathscr{C}]$.

We will now explain how an initial partition $[\overline{N}(x), \{x\}, N(x)]$ can be refined into a factorizing permutation. In order to introduce Lemma 15 we need to fix a notation.

Notation 2. Let n be an ancestor of leaf x in the cotree. Let us denote by

$$M(n, x) = \{w \in V \mid \mathrm{LCA}(x, w) = n\}.$$

Therefore $M(n, x)$ is a subset of the leaves of the cotree, but not necessarily a strong module.

Lemma 15. *Let n be an arbitrary ancestor of a given vertex x. Then the set of vertices $M(n, x)$, is a module.*

Proof. Without loss of generality let us assume that n is a series node. It should be noticed that $M(n, x)$ is included in $N(x)$. If $y \in \overline{N}(x)$ is adjacent to some vertex $w \in M(n, x)$, then n is an ancestor of $\mathrm{LCA}(y, x)$. If y is non-adjacent to some vertex $w' \in M(n, x)$, then $\mathrm{LCA}(y, x)$ is an ancestor of n. Thus in these cases, y cannot split $M(n, x)$. Now let us consider a vertex $y \in N(x) \backslash M(m, x)$. Then $\mathrm{LCA}(x, y)$ is a series node distinct from n that is either a descendant or an ancestor of n. Therefore, y is adjacent to any vertex of $M(n, x)$. □

Having the initial partition $[\overline{N}(x), \{x\}, N(x)]$, the remaining problem is to refine $N(x)$ and $\overline{N}(x)$ into subparts corresponding to the sets $M(n, x)$ for any ancestor n of x. The following lemmas will be helpful. They are based on Remark 2.

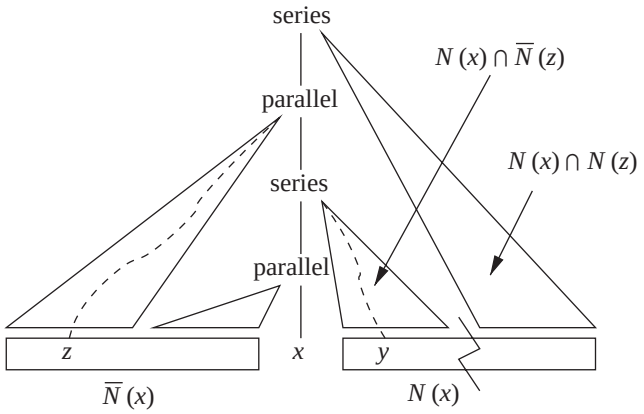

Fig. 4. Vertex z splits the part formed by $N(x)$ (dotted lines represent path in the cotree).

Lemma 16 (*Habib et al. [11]*). *Let y and z be two vertices of a cograph such that $y \in N(x)$, $z \in \overline{N}(x)$ and let $\mathscr{P}$ be a partition thinner than $[\overline{N}(x), \{x\}, N(x)]$ such that there exists a factorizing permutation compatible with $\mathscr{P}$.*

- *If y splits a part $\mathscr{C} \subseteq \overline{N}(x)$ then there exists a factorizing permutation compatible with $\mathscr{P}'$ that is obtained from $\mathscr{P}$ by refining $\mathscr{C}$ into $[\mathscr{C} \cap \overline{N}(y), \mathscr{C} \cap N(y)]$.*
- *If z splits a part $\mathscr{C} \subseteq N(x)$ then there exists a factorizing permutation compatible with $\mathscr{P}'$ that is obtained from $\mathscr{P}$ by refining $\mathscr{C}$ into $[\mathscr{C} \cap \overline{N}(z), \mathscr{C} \cap N(z)]$* (see Fig. 4).

Proof. Without loss of generality let us consider the second case (a similar proof holds for the first). Since z splits $\mathscr{C}$, there exists $u \in \mathscr{C}$ adjacent to z and $v \in \mathscr{C}$ non-adjacent to z. Since $z \in \overline{N}(x)$, $\mathrm{LCA}(x, z)$ is a parallel node. Then Corollary 11 applied to triple z, x, v and z, x, u, respectively, shows that $\mathrm{LCA}(x, v) <_T \mathrm{LCA}(x, z) <_T \mathrm{LCA}(x, u)$. Thus z and $w \in \mathscr{C}$ are non-adjacent iff $\mathrm{LCA}(z, w) =_T \mathrm{LCA}(x, z)$. So using the neighborhood of z, we can separate $M(\mathrm{LCA}(x, z), x)$ from the other vertices of $\mathscr{C}$. And by Corollary 12, $M(\mathrm{LCA}(x, z), x) \cap \mathscr{C}$ has to occur before $\mathscr{C} \setminus M(\mathrm{LCA}(x, z), x)$ in any factorizing permutation compatible with $\mathscr{P}$. By assumption there exists a factorizing permutation compatible with $\mathscr{P}$. Recall that from $\mathscr{P}$ to $\mathscr{P}'$ only part $\mathscr{C}$ has been splitted into $[\mathscr{C} \cap \overline{N}(z), \mathscr{C} \cap N(z)]$. Since Corollary 12 means that there is no factorizing permutation of G thinner than $\mathscr{P}$ in which a vertex of $\mathscr{C} \cap N(z)$ could be placed left to a vertex of $\mathscr{C} \cap \overline{N}(z)$, it exists a factorizing permutation of G compatible with $\mathscr{P}'$. □

The previous lemma allows us to express a simple refinement rule, which will be useful for the computation of a factorizing permutation.

Refinement rule 2. If a vertex $y \notin \mathscr{C}$ separates two vertices of a part $\mathscr{C}$, then refine $\mathscr{C}$ into $[\mathscr{C} \cap \overline{N}(y), \mathscr{C} \cap N(y)]$.

So using the neighborhood of each vertex of $\overline{N}(x)$ to refine the partition, $N(x)$ can be partitioned into $[M(n_1, x), \ldots, M(n_k, x)]$, where $n_1, \ldots, n_k$ are the series nodes on the path from x up to the root of the cotree. Since the $M(n_i, x)$ sets are modules by Lemma 2, then to refine $\overline{N}(x)$ in the same manner, using a single vertex per partition subpart of $N(x)$ is sufficient.

The next lemma explains how to launch again the refinement process into any non-singleton part. It shows that the same ideas can be recursively applied:

Lemma 17. *Let $\mathscr{P}$ be a partition that can be refined into a factorizing permutation and $\mathscr{C}$ be a part of $\mathscr{P}$ that is a module* (*not necessarily a strong module*). *Let $\mathscr{P}'$ be the partition obtained from $\mathscr{P}$ by using rule* 1 *on $\mathscr{C}$ with an arbitrary vertex of $x \in \mathscr{C}$. Then there exists a factorizing permutation that is compatible with $\mathscr{P}'$.*

Proof. Since $\mathscr{C}$ is a module any vertex $y \neq x$ of $\mathscr{C}$ behaves like x with respect to the vertices in $V \setminus \mathscr{C}$. It means that part $\mathscr{C}$ can be refined independently from the rest of the partition as a whole cograph, and Lemma 14 can be applied. Therefore to launch the process in $\mathscr{C}$, rule 1 can be applied. □

Algorithm 1: A kind of "brute force" algorithm

Input: Let $G = (V, E)$ be a cograph

Output: A factorizing permutation $\mathscr{P}$ of G

begin

Set the initial partition $\mathscr{P}$ to $[V]$

while *there exist non-singleton parts in* $\mathscr{P}$ **do**

Choose any non-singleton partition part $\mathscr{C}$ in $\mathscr{P}$

Choose an arbitrary vertex $x \in \mathscr{C}$ as the origin

1 Refine $\mathscr{C}$ into $[\mathscr{C} \cap \overline{N}(x), \{x\}, \mathscr{C} \cap N(x)]$ "Rule 1";

2 Iteratively refine $\mathscr{C} \cap N(x)$ using the neighborhoods of the vertices of $\mathscr{C} \cap \overline{N}(x)$: $\forall y \in \mathscr{C} \cap \overline{N}(x)$, refine each subpart $\mathscr{C}'$ of $\mathscr{C} \cap N(x)$ into $[\mathscr{C}' \cap \overline{N}(y), \mathscr{C}' \cap N(y)]$ "Rule 2";

3 Pick an arbitrary vertex y in each subpart of $\mathscr{C} \cap N(x)$ and use $N(y)$ as pivot set to refine each subpart $\mathscr{C}'$ of $\mathscr{C} \cap \overline{N}(x)$ into $[\mathscr{C}' \cap \overline{N}(y), \mathscr{C}' \cap N(y)]$ "Rule 2";

Return $\mathscr{P}$

end

Invariant of algorithm 1. *There exists a factorizing permutation compatible with the current ordered partition* $\mathscr{P}$.

Proof. Initially true, this invariant is proved for step 1 using Lemma 14, and for step 2 using Lemma 16. After step 2, any part $\mathscr{C} \subseteq N(x)$ corresponds to a set $M(n, x)$ for some ancestor n of x. So by Lemma 15, it is a module. Therefore during step 3, the refining rule (2) can be applied just to one vertex per subpart of $N(x)$. By Lemma 16 the invariant is preserved.

Applying the same argument, we prove that after steps 2 and 3, necessarily all non-singleton parts of $\mathscr{P}$ are modules of G. Since we only refine the partition when needed, a part which cannot be separated is necessarily maximal with this property and therefore by Lemma 17 step 1 can be recursively processed. □

The correctness of Algorithm 1 follows from the above invariant that states:

Theorem 18. *If G is a cograph, then Algorithm* 1 *ends up with a final partition* $\mathscr{P}$ *which is a factorizing permutation of G.*

3.2. A linear time algorithm

Clearly the complexity of the above algorithm 1 is not linear since a given vertex can be used $O(n)$ times as a pivot to refine the partition: it implies a time-complexity larger than $O(nm)$ or $O(mlogn)$ if the part $\mathscr{C}$ is chosen via a cleverer rule [19].

To achieve linear time complexity we need to use only O(1) time the neighborhood of each vertex. The problem is now to choose the pivot vertices in an appropriate ordering. Indeed, as shown in Fig. 5, an arbitrary choice may give a vertex whose neighborhood does not refine the partition.

The idea of the algorithm is to use only one vertex per part as long as possible. When any part has a pivot that has been used, we have to find a way to relaunch the refinement process. The next lemma explain how it can be achieved using rule 1 can be used again. In the following, we will denote by $\mathscr{C}_y$ the part containing a given vertex y.

Lemma 19. *Let* $G = (V, E)$ *be a cograph and* $\mathscr{P}$ *be a partition with vertex x as* $Origin$, *that can be refined into a factorizing permutation. Let* $y \in \mathscr{C}_y$ *be a pivot. Let us assume that any vertex z such that* $\mathrm{LCA}(x, z)$ *is a descendant of* $\mathrm{LCA}(x, y)(\mathrm{LCA}(x, z) <_T \mathrm{LCA}(x, y))$ *belongs to a singleton part. Let* $\mathscr{P}'$ *be the partition obtained from* $\mathscr{P}$ *by splitting* $\mathscr{C}_y$ *into* $[\overline{N}(y) \cap \mathscr{C}_y, \{y\}, N(y) \cap \mathscr{C}_y]$. *Then there exists a factorizing permutation compatible with* $\mathscr{P}'$.

Proof. Without loss of generality, let us assume that $y \in N(x)$. Any vertex $v \in N(x)$ such that $v \notin M(\mathrm{LCA}(x, y), x)$ is a neighbor of y. Indeed $\mathrm{LCA}(y, v)$ is a series node that is either a descendant or an ancestor of $\mathrm{LCA}(x, y)$. If $\mathrm{LCA}(x, v) <_T \mathrm{LCA}(x, y)$, by assumption v belongs to a singleton part and thus does not belong to $\mathscr{C}_y$. If $\mathrm{LCA}(y, v) >_T \mathrm{LCA}(x, y)$, then v may belong to $\mathscr{C}_y$ or not. The simple case holds when $\mathscr{C}_y = M(\mathrm{LCA}(x, y), x)$ and is proved by Lemma 17. But by assumption there may also exist

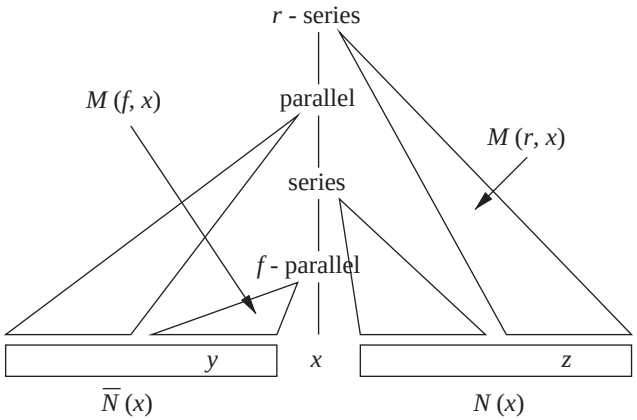

Fig. 5. Let r be the root of the cotree and f be the father of x. Let us consider arbitrary vertices $y \in M(f, x)$ and $z \in M(r, x)$. Since $N(x) \subseteq N(y)$ and $\overline{N}(x) \subseteq \overline{N}(z)$, y and z do not split any part of the partition.

some neighbors $z \in \mathscr{C}_y$ of y such that $\mathrm{LCA}(x, y) <_T \mathrm{LCA}(x, z)$. But notice that for any factorizing permutation σ compatible with $\mathscr{P}$, we have to have $x <_\sigma y <_\sigma z'$ where $z' \in N(y)$ and $\mathrm{LCA}(x, y) <_T \mathrm{LCA}(x, z')$. Indeed it will be the case in $\mathscr{P}'$ since $\mathscr{C}_y$ is splitted into $[\overline{N}(y) \cap \mathscr{C}_y, \{y\}, N(y) \cap \mathscr{C}_y]$. □

If we know the $Origin$ of the current partition, we can easily find a vertex y as describe in Lemma 19. We have to consider the first two pivots, respectively, on the left and on the right that belong to non-singleton parts. Then by Corollary 11, we can determine the good one by a simple adjacency test. An example of execution of the algorithm is given in Appendix (see Fig. 7).

Algorithm 2: Computing a factorizing permutation of a cograph

Input: A graph $G = (V, E)$ and an empty stack Q of vertices

Output: A permutation of V that is a factorizing permutation if G is a cograph

begin

  $\mathscr{P} = [V]$

1 Choose an arbitrary vertex x of G as $Origin$

  if *$Origin$ is an isolated vertex or a universal vertex* **then**

2 recurse on $G[V \setminus \{Origin\}]$

  while *there exist some non-singleton parts* **do**

  if *$\mathscr{C}_{Origin}$ is not a singleton* **then**

3 Use rule **??** on $\mathscr{C}_{Origin}$ with $Origin$ as pivot

  Set $\overline{N}(Origin) \cap \mathscr{C}_{Origin}$ and $N(Origin) \cap \mathscr{C}_{Origin}$ as unused parts

  while *there exist unused parts* **do**

  Pick an arbitrary unused part $\mathscr{C}$ and an arbitrary vertex $y \in \mathscr{C}$

  Set y as the pivot of $\mathscr{C}$

  Refine the parts $\mathscr{C}' \neq \mathscr{C}$ of $\mathscr{P}$ with rule **??** using the pivot set $N(y)$

  Mark $\mathscr{C}$ as used and the new created subparts without pivot as unused

  Let z_l and z_r be the pivots of the nearest non singleton parts to $Origin$ respectively on its left and on its right

4 **if** *z_l is adjacent to z_r* **then** $Origin \leftarrow z_l$ **else** $Origin \leftarrow z_r$

  return $\mathscr{P}$

end

The invariant of Algorithm 2 will be proved in two steps. Roughly speaking, the first property is that each time a new origin is chosen all its neighbors which are not singletons are on its left side. This will ensures the validity of the refinement rules. And

thus at any step, there is a factorizing permutation compatible with the current partition. Let us introduce some notations:

- Let x_0 be the first origin chosen at step 1 and x_i be the $i+1$th vertex chosen as the new origin of the partition (step 4).
- The part containing x_i and the partition at the step in which x_i becomes the current origin are denoted, respectively, by $\mathscr{C}_i$ and $\mathscr{P}_i$.
- The invariant deals with subsets V_i of the vertex set V and cotrees T_i of the subgraphs induced by V_i. We define $V_0 = V$ and $V_i = V_{i-1} \backslash M_i$, $i > 0$, where $M_i = M(n_i, x_{i-1})$ with n_i the son of $\mathrm{LCA}(x_i, x_{i-1})$ in the cotree T_i. Let us remark that by Lemma 15, M_i is a module for the subgraph induced by V_{i-1}.

Invariant of algorithm 2. *Let $\mathscr{P}$ be the current partition with origin x_i. If G is a cograph, then*:

a: *Let y be a vertex of V_i. Then $yx_i \in E$ iff $x_i <_{\mathscr{P}} y$.*
b: *there exists a factorizing permutation compatible $\mathscr{P}$.*

Proof.

- *Invariant a*. The property is clearly true in the case $i = 0$, since the initial partition $\mathscr{P}_0$ is $[\overline{N}(x_0), \{x_0\}, N(x_0)]$ (see step 1). Let us assume by induction that invariant A holds for $i \geqslant 0$. Let us consider a vertex $y \in V_{i+1}$. Let us remark that $\mathrm{LCA}(y, x_i) \geqslant_{T_i} \mathrm{LCA}(x_{i+1}, x_i)$. Therefore, $\mathrm{LCA}(y, x_i) = \mathrm{LCA}(y, x_{i+1})$. Since $yx_i \in E$ iff $y >_{\mathscr{P}_i} x_i$, we also have $yx_{i+1} \in E$ iff $y >_{\mathscr{P}_1} x_{i+1}$.
- *Invariant b*. In the following, all the partition we deal with, are issued from the refinement process. *Invariant B* is initially true by Lemma 14 for $\mathscr{P}_0$. Let us consider any partition $\mathscr{P}$ that is coarser than $\mathscr{P}_1$. $\mathscr{P}$ is obtained by successive application of rule 2 and thus Lemma 16 ensures that the invariant is preserved. Let us assume by induction invariant B for any $\mathscr{P}$ strictly coarser than $\mathscr{P}_i$ with $i \geqslant 1$. Let us first consider $\mathscr{P}_i$. By Corollary 11, the the new origin x_i checks (see step 4) the hypothesis of Lemma 19. Thus the refinement process can be relaunched on $\mathscr{C}_i$ using rule 1. So *invariant B* is verified for $\mathscr{P}_i$. Now recall that by the choice of x_i, M_i is composed by singleton parts. It means that the problem of computing a factorizing permutation on the subgraph induced by M_i is solved. Since M_i is a module (that contains x_{i-1}) for the subgraph induced by V_{i-1}, $\mathscr{P}_i$ is stable with respect to the neighborhood of any vertex of M_i. In other words, M_i can be removed to end the refinement process. Let $\mathscr{P}'_i$ be the partition of V_i obtained from $\mathscr{P}_i$ by removing the vertices of all the M_j, $j \leqslant i$. Now there exists a factorizing permutation of the induced subgraph $G[V \backslash M_i]$ compatible with $\mathscr{P}'_i$. Since by *invariant A*, $\mathscr{P}'_i$ is thinner than $[\overline{N}(x_i), \{x_i\}, N(x_i)]$, exactly the same arguments than those used for the initial case while there are some unused parts, shows that the property also holds for the current partition $\mathscr{P}$ at any step of the refinement process. □

The above invariant proves the following theorem that states the correctness of Algorithm 2:

Theorem 20. *Algorithm* 2 *computes a factorizing permutation if the input graph is a cograph*.

Proof. When any part is a singleton part, since *invariant B* has been preserved by any refinement step, the partition $\mathscr{P}$ is a factorizing permutation. □

4. Data-structures and complexity issues

In this section, we describe the data-structures and the key points of algorithm 2. The complexity of these operations will be proved. Then the complexity analysis of the whole algorithm follows.

A partition $\mathscr{P}$ of a set E is represented as shown in Fig. 6. The elements of E are stored in a sorted list and each element has a pointer to its part. Since the elements of a given part are consecutive in the sorted list, each part can be represented with pointers to its first and last elements. The parts of the partition are stored in two different sorted lists depending on their status: one list of the *used* parts and one list for the *unused* parts. To each used part, we have to store its vertex that has been used as a pivot. That vertex will be used once more at step 3 of algorithm 2.

Lemma 21. *The neighborhood of each vertex is used at most once by procedure* 3, *that can be processed in* $O(|N(x)|)$ *times*.

Proof. A vertex in a used part $\mathscr{C}$ can be used once more by procedure 3 iff $\mathscr{C}$ is split into subparts. Since the vertex x used in procedure 3 is the only member of a new used part, it will never be used again. Procedure 3 can clearly be achieved in $O(|N(x)|)$ time since we mainly have to move the neighbors of x in the list of vertices. □

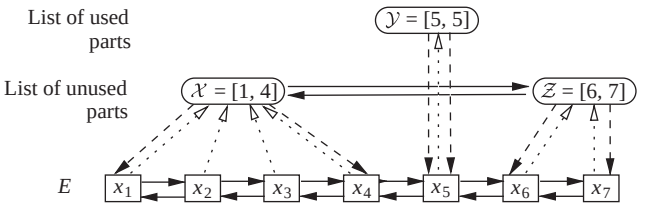

Fig. 6. Partition data-structure.

Procedure 3: *Init*($\mathscr{C}$)

Input: a part $\mathscr{C}$ of the current partition $\mathscr{P}$

Output: a partition $\mathscr{P}'$ compatible with $\mathscr{P}$

begin

  if *no vertex of* $\mathscr{C}$ *has been already used as a pivot* **then**

    Choose an arbitrary vertex x

  else let x be the already used pivot Remove $N(x)$ from $\mathscr{C}$ and insert it on the right

  Remove x from $\mathscr{C}$ and insert it on the right

  $\overline{N}(x) \cap \mathscr{C}$ and $N(x) \cap \mathscr{C}$ are unused parts, $\{x\}$ is a used part

end

The procedure 3 is an implementation of rule 1. Let us have a look at the implementation of rule 2 (i.e., refines a partition $\mathscr{P}$ using a pivot set S) and updates the lists of parts. While splitting a used part into two subparts, a new unused part is created.

Procedure 4: *Refine*($\mathscr{P}$, S)

Input: a partition $\mathscr{P}$ and a pivot set S

Output: a partition $\mathscr{P}'$ compatible with $\mathscr{P}$ and stable with respect to S

begin

  let M be the set of parts strictly intersected by S

  foreach *part* $\mathscr{X} \in M$ **do**

    let $\mathscr{X}_a$ be the elements of $\mathscr{X}$ belonging to S

    remove $\mathscr{X}_a$ from $\mathscr{X}$ and insert $\mathscr{X}_a$ immediately on the right of $\mathscr{X}$

    if $\mathscr{X}$ *is an unused part* **then**

      $\mathscr{X}_a$ and $\mathscr{X} \setminus \mathscr{X}_a$ are also unused parts

    else

      let x be the used pivot of $\mathscr{X}$

      the new subpart that does not contain x is an unused part

end

Lemma 22. *Refining a partition of the vertices with the neighborhood of a vertex x can be done in* $O(|N(x)|)$ *times.*

Proof. Each element of S can be moved from its position to the beginning (or the end) of its part in constant time. Then counting the moves in each touched part, allows us to create new parts containing the elements of S. Let $\mathscr{X}$ be a part strictly overlapped by S. A new record for the new part $\mathscr{X}_a = \mathscr{X} \cap S$ is created. The record corresponding to the old part $\mathscr{X}$ now represents the part $\mathscr{X}_b = \mathscr{X} \setminus S$. At most $|S|$ new records are created, and exactly $|S|$ elements are moved. So this operation can be achieved in $O(|S|)$ time using an appropriate data structure. □

In order to implement efficiently step 4 of algorithm 2, each time a vertex of a singleton part $\mathscr{C}$ is used as pivot with rule 2, the part $\mathscr{C}$ is removed from the lists of parts. Also when the origin of the partition changes, the part containing the part of the old origin is removed from the lists of parts. Therefore, to choose the new origin, we just have to look at the pivots of the two parts adjacent to the part containing the origin.

Lemma 23. *The neighborhood of each vertex is used at most* 3 *times to refine the partition*.

Proof. The neighborhood of a given vertex x can be used to refine the partition with rule 2. It can also be used for the adjacency test at step 4 of algorithm 2. To test whether the two candidates are connected or not, it suffices to scan the smallest neighborhood and this search could be charged to the chosen new origin. Thus in the whole any neighborhood can be charged at most once at step 4. The next use of $N(x)$ is to refine its partition part with rule 1 when it becomes the new origin, with procedure 3. □

Theorem 24. *The algorithm* 2 *computes a factorizing permutation of a cograph in* $O(n+m)$ *time*.

Proof. By Lemma 23, during the whole refining process each neighborhood is used O(1) times. So the whole complexity is $O(\sum_{x\in V}|N(x)|) = O(n+m)$. □

5. A very simple recognition test

Let us now consider the testing problem, i.e. to test whether the output permutation of algorithm 2 is a factorizing permutation or not. This work can easily be done in linear time using the following simple algorithm that scans the given permutation from left to right.

It is well known that any cograph admits a twin-elimination ordering of its vertex set defined as following : $\sigma = x_1, \ldots, x_n$ such that for any vertex x_i, $1 \leqslant i < n$, there exists j, $i < j \leqslant n$ such that x_i and x_j are either *true twins* or *false twins*.

Definition 25. Two vertices x and y of a graph G are false (respectively true) twins iff $N(x)=N(y)$ (resp. $N(x)\cup\{x\}=N(y)\cup\{y\}$).

Algorithm 5: Recognition test

Input: Let $\sigma = x_1, \ldots, x_n$ be a permutation of the vertex set of a graph G, σ is represented as a doubly linked list.

Output: σ a list of vertices

begin

Let x_0 and x_{n+1} be added to σ (these vertices are dummies which are not twins with any other vertex)
Let z be the current vertex, initially $z \leftarrow x_1$
Let $succ(z)$ (resp. $prec(z)$) be the vertex following (resp. preceding z) in σ
while $z \neq x_{n+1}$ **do**
  if *z and $prec(z)$ are twins (true or false) in $G(\sigma)$* **then**
    remove $prec(z)$ from σ
  else
    if *z and $succ(z)$ are twins (true or false) in $G(\sigma)$* **then**
      $z \leftarrow succ(z)$
      remove $prec(z)$ from σ
    else $z \leftarrow succ(z)$
if $|\sigma - \{x_0, x_{n+1}\}| = 1$ **then return** G is a cograph **else return** $G(\sigma)$ contains a P_4

end

Clearly two vertices x and y are twins iff they are brothers in the cotree (true twins if their parent node $\mathrm{LCA}(x, y)$ is a series node, false otherwise). By definition, many brothers will occur consecutively in a factorizing permutation. The natural idea is to

scan the computed permutation from left to right. The description of the testing process is given by algorithm 5. An example of execution on the cograph of Fig. 7 is given in Appendix.

Theorem 26. *A permutation σ computed by algorithm* 2 *is a factorizing permutation of a cograph iff algorithm* 3 *ends up with σ reduced to a single vertex.*

Proof. If algorithm 5 ends up with a single vertex, then a twin vertex elimination ordering has been found and thus the corresponding graph is a cograph. Let us now prove that if G is a cograph algorithm 5 ends with a single vertex in σ. Clearly algorithm 5 maintains as invariants the following properties:

Invariant 1. If G is a cograph σ is a factorizing permutation of $G(\sigma)$, where $G(\sigma)$ denotes the subgraph induced by the vertices in σ.

Let us denote by z_k (respectively σ_k) the current vertex z (respectively the permutation) after k steps of the while loop, in particular $z_0 = x_1$. We now prove by induction that:

Invariant 2. For any $k \geqslant 1$, the subsequence $\sigma_k([z_0, z_k[)$ does not contain any twins vertices in $G(\sigma)$.

For $k = 1$, the property is obviously true, since $\sigma([z_0, z_1[)$ contains at most one vertex. Let us now execute step $k + 1$ of the while loop; three cases have to be considered corresponding to the algorithm.

(1) z_k and $prec(z_k)$ are twins in $G(\sigma)$. But then $prec(z_k)$ is deleted from σ_k and $z_{k+1} = z_k$, $\sigma_{k+1}([z_0, z_{k+1}[)$ is included in $\sigma_k([z_0, z_k[)$, and therefore invariant 2 is trivially true.
(2) z_k and $succ(z_k)$ are twins in $G(\sigma)$. But then z_k is deleted from σ_k and $z_{k+1} = succ(z_k)$, $\sigma_{k+1}([z_0, z_{k+1}[) = \sigma_k([z_0, z_k[)$, and therefore invariant 2 is trivially true.
(3) In the last case, we move right on the circular list, and $z_{k+1} = succ(z_k)$, $\sigma_{k+1}([z_0, z_{k+1}[) = \sigma_k([z_0, z_k])$.

Using the induction hypothesis it suffices to show that z_k has no twin in $\sigma_k([z_0, z_k])$. Let us suppose the contrary, i.e. z_k admits a twin $z_h \in \sigma_k([z_0, z_k[)$.
If we consider $\sigma_k(]z_h, z_k[)$ it corresponds to a factorizing permutation of a cograph and therefore by induction the algorithm must have reduced it to a single vertex$z$ in σ_k. But this vertex is equal to $prec(z_k)$ a twin of z_k, a contradiction.

Therefore, if G is a cograph, using the two previous invariants, we can prove that necessarily algorithm 5 ends up with σ such that: $|\sigma - \{x_0, x_{n+1}\}| = 1$. □

Theorem 27. *The recognition of cographs can be done in* $O(n + m)$ *using algorithm* 2 *and* 5.

Proof. By Theorem 24 the computation of the factorizing permutation can be done in linear time. Let us analyze the complexity of the test (algorithm 5). Assuming that the neighborhoods are all given in the same sorted order, to test whether two vertices u and v are twins in σ can be done in $O(\min(|N(u)|, |N(v)|))$. Let us consider an execution of a step of the while loop.

- If some twins are detected, then z can remain the current vertex, but this step can be charged to the eliminated vertex.
- If no twins are detected, then z will be no longer be the current vertex and this step can be charged to z in $O(|N(z)|)$.

Therefore in the whole, the neighborhood of a vertex can be at used at most twice, once as the current vertex and another time as an eliminated vertex. So the whole complexity can be done in $O(n + m)$. □

When a cograph has been recognized, if the cotree is needed, one can easily build a binary series-parallel tree from the twin-elimination ordering. To transform this tree into a canonical cotree, it suffices to merge neighbor series (respectively parallel) nodes.

6. Conclusions

We have liked to see a graph algorithm as a function applied on the graph that provides a permutation σ of the vertices that contains all the required information. Such a framework includes many graph algorithms such as depth-first search, (lexicographic) breadth first search, chordal graph recognition.

It may turn out that even if the input graph is not a cograph algorithm 2 may output a factorizing permutation. Just consider the case of the P_4 for which any permutation of its vertex set is a factorizing permutation. So to recognize cographs, the recognition test has to be performed after algorithm 2. For those reasons the presented algorithm can be considered as a robust algorithm [20]:

- If the test fails, then it produces a certificate, namely the subgraph $G(\sigma)$, that shows the input graph is not a cograph. To be a little more precise, if T_G is the modular decomposition tree of G, then $T_{G(\sigma)}$ is the tree obtained from T_G by recursively deleting all series and parallel nodes whose children are only leaves. If G is not a cograph, then $T_{G(\sigma)}$ contains a prime node and so $G(\sigma)$ contains a P_4.
- But also since it is possible to extract in linear time the modular tree decomposition out of a factorizing permutation [2], the same algorithm may be used to recognize more general graph classes: for example graphs having with few P_4 [13,15–17].

Of course, another natural generalization of these ideas would be to apply them to modular decomposition. It has still to be done, since the algorithm developed in [11] has an extra *logn* factor.

Acknowledgements

The authors wish to thank the anonymous referees for their careful reading and their useful remarks which greatly help us to improve and simplify the paper.

Appendix A. An example

A.1. Computation of a factorizing permutation of the graph of Fig. 7

a. Vertex 0 is the first origin. Vertices 1 and 2 have been used as pivot, but do not refine any part of the partition.
b. The innermost pivot with respect to the current origin, namely 0, is 1. Then 1 is the new origin and its part is refined using rule 1.

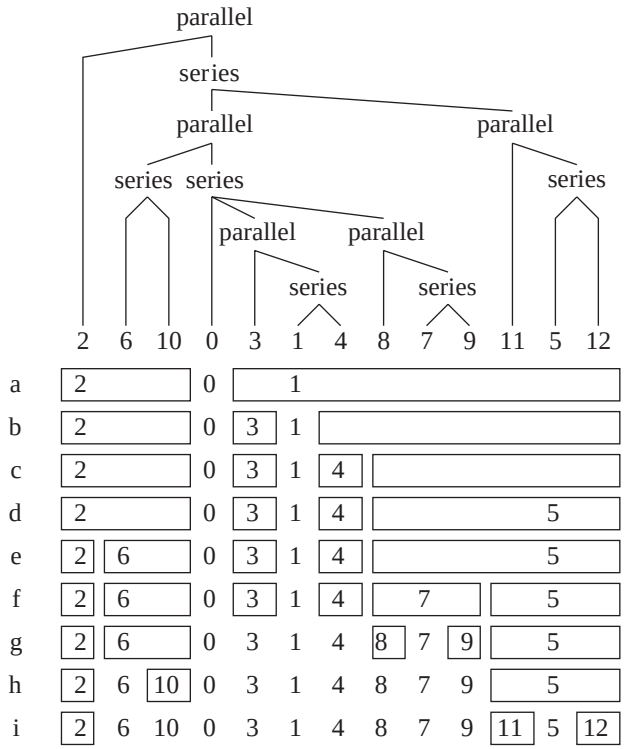

Fig. 7. An execution of the algorithm.

c. Vertex 3 splits the rightmost part into [4] and [8, 7, 9, 11, 5, 12]. Then vertex 4 can be used but it does not refine anything.
d. Vertex 5 is used and splits the part containing 2 into [2] and [6, 10].
e. Vertex 6 is used and splits the part containing 5 into [8, 7, 9] and [11, 5, 12].
f. Vertex 7 is used but refines nothing.
g. All the parts have been used. The innermost pivot with respect to the current origin, namely 1, is 7. The part containing 7 is refined using rule 1 into [8], [7] and [9]. Vertices 8 and 9 can be used but refines nothing.
h. All the parts have been used. The innermost pivot with respect to the current origin, namely 7, is 6. The part containing 6 is refined using rule 1 into [6] and [10]. Vertex 10 can be used but refines nothing.
i. All the parts have been used. The innermost pivot with respect to the current origin, namely 6, is 5. The part containing 5 is refined using rule 1 into [11][5] and [10]. Now all parts are singletons, we are done.

A.2. The recognition test

- $\sigma = [x_0, 2, 6, 10, 0, 3, 1, 4, 8, 7, 9, 11, 5, 12, x_{14}]$
  - $z = 2$: 2 and x_0 nor 2 and 6 are twins.
  - $z = 6$: 6 and 2 are not twins but 6 and 10 are. Thus set $z = 10$ and 6 is removed.
- $\sigma = [x_0, 2, 10, 0, 3, 1, 4, 8, 7, 9, 11, 5, 12, x_{14}]$
  - $z = 10$: 10 and 2 nor 10 and 0 are twins.
  - $z = 0$: 0 and 10 nor 0 and 3 are twins.
  - $z = 3$: 3 and 0 nor 3 and 1 are twins.
  - $z = 1$: 1 and 3 are not twins but 1 and 4 are. Thus set $z = 4$ and 1 is removed.
- $\sigma = [x_0, 2, 10, 0, 3, 4, 8, 7, 9, 11, 5, 12, x_{14}]$, $z = 4$: 4 and 3 are twins. Thus 3 is removed.
- $\sigma = [x_0, 2, 10, 0, 4, 8, 7, 9, 11, 5, 12, x_{14}]$, $z = 4$: 4 and 0 are twins. Thus 0 is removed.
- $\sigma = [x_0, 2, 10, 4, 8, 7, 9, 11, 5, 12, x_{14}]$
  - $z = 4$: 4 and 10 nor 4 and 8 are twins.
  - $z = 8$: 8 and 4 nor 8 and 7 are twins.
  - $z = 7$: 7 and 8 are not twins but 7 and 9 are. Thus set $z = 9$ and 7 is removed.
- $\sigma = [x_0, 2, 10, 4, 8, 9, 11, 5, 12, x_{14}]$, $z = 9$: 9 and 8 are twins. Thus 8 is removed.
- $\sigma = [x_0, 2, 10, 4, 9, 11, 5, 12, x_{14}]$, $z = 9$: 9 and 4 are twins. Thus 4 is removed.
- $\sigma = [x_0, 2, 10, 9, 11, 5, 12, x_{14}]$, $z = 9$: 9 and 10 are twins. Thus 10 is removed.
- $\sigma = [x_0, 2, 9, 11, 5, 12, x_{14}]$
  - $z = 9$: 9 and 2 nor 9 and 11 are twins.
  - $z = 11$: 11 and 9 nor 11 and 5 are twins.
  - $z = 5$: 5 and 11 are not twins but 5 and 12 are. Thus set $z = 12$ and 5 is removed.
- $\sigma = [x_0, 2, 9, 11, 12, x_{14}]$, $z = 12$: 12 and 11 are twins. Thus 11 is removed.
- $\sigma = [x_0, 2, 9, 12, x_{14}]$, $z = 12$: 12 and 9 are twins. Thus 9 is removed.
- $\sigma = [x_0, 2, 12, x_{14}]$, $z = 12$: 12 and 2 are twins. Thus 2 is removed.
- $\sigma = [x_0, 12, x_{14}]$
  - $z = 12$: 12 and x_0 nor 12 and x_{14} are twins.
  - $z = x_{14}$: End of the algorithm, *G is a cograph*.

References

[1] A. Bretscher, D.G. Corneil, M. Habib, C. Paul, A simple linear time lexbfs cograph recognition algorithm, in: Graph-Theoretic Concepts in Computer Science - WG'03, number 2880 in Lecture Notes in Computer Science, 2003, pp. 119–130.
[2] C. Capelle, Decomposition de graphes et permutations factorisantes, Ph.D. Thesis, Univ. de Montpellier II, 1997.
[3] C. Capelle, M. Habib, F. de Montgolfier, Graph decompositions and factorizing permutations, Discrete Math. Theoret. Comput. Sci. 5 (1) (2002) 55–70.
[4] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (4) (1985) 926–934.
[5] A. Cournier, M. Habib, A new linear algorithm for modular decomposition, in: S. Tison (Ed.), 19th International ColloquiumTrees in Algebra and Programming, CAAP'94, Vol. 787, Lecture notes in Computer Science, Springer, Berlin, 1994, pp. 68–82.
[6] E. Dahlhaus, Efficient parallel algorithms of cographs and distance hereditary graphs, Discrete Appl. Math. 57 (1995) 29–54.
[7] E. Dahlhaus, J. Gustedt, R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, J. Algorithms 41 (2) (2001) 360–387.
[8] G. Damiand, M. Habib, C. Paul, A simple paradigm for graph recognition: application to cographs and distance hereditary graphs, Theoret. Comput. Sci. 263 (2001) 99–111.

[9] T. Gallai, Transitiv orientierbarer graphen, Acta Math. Acad. Sci. Hung. 18 (1967) 25–66.
[10] M. Habib, R. McConnell, C. Paul, L. Viennot, Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, Theoret. Comput. Sci. 234 (2000) 59–84.
[11] M. Habib, C. Paul, L. Viennot, Partition refinement techniques: an interesting algorithmic tool kit, Internat. J. Found. Comput. Sci. 10 (2) (1999) 147–170.
[12] M. Habib, C. Paul, L. Viennot, Linear time recognition of p_4-indifference graphs, Discrete Math. Theoret. Comput. Sci. 4 (2) (2001) 173–178.
[13] C.T. Hoàng, Perfect graphs, Ph.D. Thesis, School of Computer Science, McGill University, Montréal, 1985.
[14] Hsu, W.L., Ma, T.Z., Substitution decomposition on chordal graphs and applications, in: Proceedings of the 2nd ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation, number 557 in Lecture Notes in Computer Science, Springer, Berlin, 1991.
[15] B. Jamison, S. Olariu, A new class of brittle graphs, Stud. Appl. Math. 81 (1989) 89–92.
[16] B. Jamison, S. Olariu, p_4-reducible graphs—a class of uniquely representable graphs, Stud. Appl. Math. 81 (1989) 79–87.
[17] B. Jamison, S. Olariu, A unique tree representation of p_4-sparse graphs, Discrete Appl. Math. 35 (1992) 115–129.
[18] R.M. McConnell, J.P. Spinrad, Linear-time modular decomposition and efficient transitive orientation of comparability graphs, in: Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms Arlington, VA, ACM, New York, 1994, pp. 536–545.
[19] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (6) (1987) 973–989.
[20] V. Raghavan, J. Spinrad, Robust algorithms for restricted domains, in: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, Vol. 2, 2001, pp. 460–467
[21] S. Seinsche, On a property of the class of n-colorable graphs, J. Combin. Theory (B) (1974) 191–193.