Fully dynamic recognition algorithm and certificate for directed cographs

Christophe Crespelle and Christophe Paul

CNRS - *Département Informatique*, LIRMM, Montpellier
`{crespell,paul}@lirmm.fr`

Abstract. This paper presents an optimal fully dynamic recognition algorithm for directed cographs. Given the modular decomposition tree of a directed cograph G, the algorithm supports arc and vertex modification (insertion or deletion) in $O(d)$ time where d is the number of arcs involved in the operation. Moreover, if the modified graph remains a directed cograph, the modular decomposition tree is updated; otherwise, a certificate is returned within the same complexity.

1 Introduction

Directed cographs is the family of digraphs recursively defined from the single vertex under the closure of the operations of *disjoint union*, *series* and *order* composition. Let $G_1, \ldots, G_k$ be a set of k disjoint digraphs. The *disjoint union* of the G_i's is the digraph whose connected components[1] is precisely the G_i's. The *series* composition of the G_i's is the union of these k graphs plus all possible arcs between vertices of different G_i's. The *order* composition of the G_i's is the union of these k graphs plus all possible arcs from G_i towards G_j, with $1 \leqslant i < j \leqslant k$. These operations define a unique tree representation of a directed cograph which corresponds to its modular decomposition tree [14]. The leaves are mapped to the vertices of the graph and the inner nodes are labelled by the different composition operations (see Fig. 1). Notice that by definition of the composition operations, the complement of a directed cograph is a directed cograph. Indeed, the term *cograph* [3] stands for *complement reducible graph.* Moreover the directed cograph family is *hereditary*: any induced subgraph of a directed cograph is also a directed cograph. It should also be noticed that directed cographs can be characterised by forbidden subgraphs (see Theorem 2 and Fig. 2).

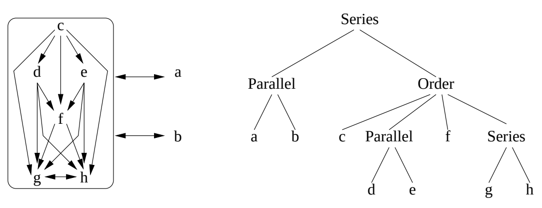


Fig. 1. A directed cograph and its modular decomposition tree. Since set $\{a, b\}$ is in series composition with the rest of the vertices, for any $x \notin \{a, b\}$ and $y \in \{a, b\}$, both arcs xy and yx exist.

Restricted to posets, directed cographs are the *series-parallel orders* [13] for which the recognition problem has been solved in linear time [17]. In the case of undirected graphs, the series composition and the order composition are equivalent. The family of undirected graphs defined from the single vertex graph by the closure of the series composition and the disjoint union is the family of *cographs.* The modular decomposition tree of a cograph is called a *cotree.* A number of linear time cograph recognition algorithms is now known: the first one was presented in [4] and the most recent one in [1].
The *dynamic recognition and representation problem* for a family $\mathcal{F}$ of graphs aims to maintain a characteristic

[1] In this paper, notion of connectivity of a digraph refers to connectivity of its underlying undirected graph.

representation of dynamically changing graphs as long as the modified graph belongs to $\mathcal{F}$. The input of the problem is a graph $G \in \mathcal{F}$ with its representation and a series of modifications. Any modification is of the following: adding a vertex (along with the arcs incident to it), deleting a vertex (and its incident arcs), adding or deleting an arc or two symmetric arcs (note that the insertion/deletion of only one of these symmetric arcs may not result in a graph of $\mathcal{F}$, while the insertion/deletion of both would). We consider only valid modification queries: any vertex or arc to be inserted must not previously exist in the graph, and similarly, any vertex or arc to be deleted must exist. Moreover, as pointed out by [12], if the property of belonging to $\mathcal{F}$ is no longer satisfied, providing a certificate would be highly desirable in practise (e.g. for debugging features). This paper considers this problem for the family of directed cographs. The representation we maintain is based on the modular decomposition tree.

Related works. The dynamic recognition and representation problem has been considered for various graphs families. [11] devised a fully dynamic recognition algorithm for chordal graphs which handles edge operations in $O(n)$ time. For proper interval graphs [10], each update can be supported in $O(d + \log n)$ where d is the number of edges involved in the operation. [6] presented a fully dynamic recognition algorithm for the class of permutation graphs which runs in $O(n)$ time per edge or vertex modification. Concerning cographs, a constant time algorithm for edge modification (insertion or deletion) has been designed in [16]. The undirected cograph recognition algorithm of [4] is incremental: given a cograph G, its cotree T and a vertex x, it modifies T iff $G + x$ is a cograph. Merging the results of [4] and [16] provides a fully dynamic recognition algorithm for cographs with $O(d)$ worst case time complexity per operation. Pushing further Algorithm of [4], if $G + x$ is not a cograph, it is possible, within the same complexity, to extract a certificate (namely a P_4, an induced path of 4 vertices).
The work of [4] has recently been extended for bipartite graphs. A new decomposition dedicated to bipartite graphs has been proposed in [8] and the family of bipartite graphs totally decomposable, as are the cographs for the modular decomposition, are defined: the weak-bisplit graphs. In [9], a linear time recognition algorithm for weak-bisplit graphs is given. It turns out that the incidence bipartite graph of a directed cograph is a weak-bisplit graph. As for cographs, the decomposition tree is built by adding the vertices one by one. But unfortunately, to get linear time complexity, the vertices have to be ordered with respect to their degree. It follows that the incremental aspect can not be guaranteed.

Our results. We present an optimal algorithm for the dynamic recognition and representation problem for the family of directed cographs. If needed, our algorithm is also able to find a certificate. Therefore, it extends the algorithms of [4, 16]. In the case of vertex insertion, we use a straightforward generalisation of the marking process of [4] to colour nodes of the tree representation (*di-cotree*) we use for directed cographs. As done in [4], we use the result of this marking step to determine whether the insertion results in a directed cograph. To that aim, we check, on the coloured di-cotree, that the conditions of Theorem 4 are satisfied. Theorem 4 gives a new characterisation of the augmented graph being a directed cograph, independently from the forbidden subgraph characterisation of the class (see further). Moreover, unlike the algorithm of [9] restricted to directed cographs, our algorithm supports arc modification and the dynamic aspect is guaranteed (that is the updates can be handled in arbitrary order). A summary of this work was previously given at [5].

Theorem 1. *The dynamic recognition and representation problem for directed cographs is solvable in $O(d)$ worst case time per update, where d is the number of edges involved in the updating operation. Moreover, if needed, a certificate that the modified graph is not a directed cograph is provided within the same time complexity.*

2 Preliminaries

We consider finite, loopless, simple and directed graphs $G = (V, E)$, with $|V| = n$ and $|E| = m$. The complement of a graph G is denoted by $\overline{G}$. If X is a subset of vertices, then $G[X]$ is the subgraph of G induced by X. Since the graphs are directed, the arc xy differs from yx. Let x be a vertex, then $N^+(x) = \{z \in V, xz \in E\}$, $N^-(x) = \{y \in V, yx \in E\}$ and $N(x) = N^-(x) \cup N^+(x)$ stand respectively for its *out-neighbourhood*, its *in-neighbourhood* and its neighbourhood. The non-neighbourhood of x, which is the

complement of its neighbourhood, will be denoted $\overline{N}(x)$. The degree $d(x)$ of a vertex x is the sum of its in-degree, $d^-(x) = |N^-(x)|$, and its out-degree, $d^+(x) = |N^+(x)|$. Let $G = (V, E)$ be a digraph, $x \notin V$ be a vertex and $N^-(x) \subseteq V$, $N^+(x) \subseteq V$ be two subsets of vertices of G. Then $G + x$ denotes the digraph $G' = (V \cup \{x\}, E \cup \{xz, z \in N^+(x)\} \cup \{yx, y \in N^-(x)\})$, in which $N^-(x)$, $N^+(x)$ are the in and out-neighbourhood of x. If $xy \in E$, $G - xy$ will be the graph $G' = (V, E \setminus \{xy\})$. $G - x$ and $G + xy$ are similarly defined.
As for the cograph family, directed cographs can be characterised by forbidden subgraphs. Unfortunately, such a characterisation does not help for an efficient recognition algorithm (even for a non-dynamic one). Nevertheless, these subgraphs will be useful to provide a certificate if the referred graph is not a directed cograph. This characterisation can be retrieved from a result of [7].

Theorem 2. *A digraph is a directed cograph iff it does not contain any graph of Fig. 2 as induced subgraph.*

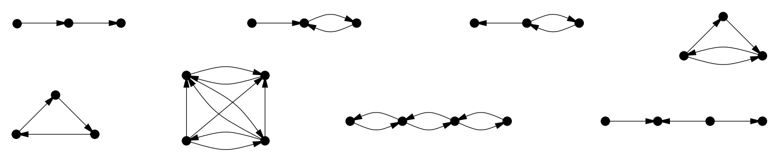


Fig. 2. The set of forbidden subgraphs for the directed cographs family. Note that this set is closed under complementation.

A *module* M is a set of vertices such that for any $x \notin M$ and $y \in M$, $xy \in E$ iff $\forall z \in M$, $xz \in E$ and $yx \in E$ iff $\forall z \in M$, $zx \in E$. The following claim is straightforward.

Claim. Let $G = (V, E)$ be a graph and $x \notin V$ a vertex to be inserted in G. Let $M \subseteq V$ such that $M \cup \{x\}$ is a module of $G + x$, then M is a module of G.

The modules of a graph are a potentially exponential-sized family. However, the sub-family of *strong* modules, the modules that overlap[2] no other module, has size $O(n)$. The inclusion order of this family defines the *modular decomposition tree*, which is enough to represent the modules family of a graph [14]. The root of this tree is the trivial module V and its n leaves are the trivial modules $\{x\}, x \in V$. The leaf corresponding to singleton $\{x\}$ will be denoted l_x. Any node p of the tree corresponds to a set of vertices $M(p)$, the set of the leaves in the subtree rooted at p, which is a module of G. To shorten the notations, the set $M(p)$ will be denoted by P. The set of children of a node p will be denoted $\mathcal{C}(p)$. We call sibling of a node p_1 in the tree, a node p_2 which has the same parent as p_1 has. In the case of directed cographs, the internal nodes are labelled by one of the three composition operations: *parallel, series* or *order* (see Fig. 1). Let us call the modular decomposition tree of a directed cograph, the *di-cotree.* In the proofs, we will often use the fundamental decomposition theorem for directed cographs, given below. We call *maximal strong module* of a graph $G = (V, E)$, a strong module of G different from V and maximal wrt. inclusion. A directed graph $G = (V, E)$ is a $k - order$, with $k \in \mathbb{N}^*$ (the set of strictly positive integers) if there exists a partition[3] $V_1 \sqcup \cdots \sqcup V_k$ of V such that for all $x \in V_i$ and for all $y \in V_j$, if $i < j$ then $xy \in E$ and $yx \notin E$. There exists a unique maximal k and a unique partition such that G is a $k - order$. The sets of this unique partition are called the *order components* of G. Note that the order components are naturally ordered, from the first V_1 to the last V_k. V_1 and V_k will also be referred as the *extremal order components* of G. A directed graph is said to be *co-connected* iff its complement is connected.

Theorem 3. *A directed cograph G is either:*

– *not connected, then its maximal strong modules are its connected components, or*

[2] A overlaps B if $A \cap B \neq \emptyset$, $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$
[3] The symbol $\sqcup$ denotes the union of disjoint sets.

– *not co-connected, then its maximal strong modules are its co-connected components, or*
– *connected and co-connected, then G is a $k-order$, for some $k \in \mathbb{N} \setminus \{0,1\}$, and its maximal strong modules are its order components.*

A set $S \subseteq V$ of vertices is *uniform* wrt. $x \notin S$ in G if $S \subseteq N^+(x)$ or $S \cap N^+(x) = \varnothing$, and $S \subseteq N^-(x)$ or $S \cap N^-(x) = \varnothing$. Equivalently, S is uniform iff S is a module of the graph $G[S \cup \{x\}]$. If S is not uniform, then it is *mixed*. We say that a node p is uniform (resp. mixed) wrt. x if P is. Finally, a set S of vertices (resp. a node p of the di-cotree) is *linked* to a vertex $x \notin S$ in G, if there exists $y \in S$ (resp. $y \in P$) st. $xy \in E$ or $yx \in E$. If S is uniform and linked, we say it is *uniformly linked*; and if S is uniform and not linked, we say that S is *uniformly not linked*. In the following, if no confusion is possible, we will omit to mention the graph in which the above notions are applied. The subtree of the di-cotree T, rooted at a node p will be denoted by T_p. The set of ancestors of node p in T will be denoted $Anc_T(p)$ and the set of its descendants will be denoted $Des_T(p)$. Note that p is considered as an ancestor and a descendant of itself, $p \in Anc_T(p) \cap Des_T(p)$. When there is no confusion, we omit the tree referred to and denote $Anc(p)$. The path between p and the root r of T will be denoted P^r_p. Finally, M_{xy} stands for the minimum (wrt. inclusion) module that contains vertices x and y. Since M_{xy} is not necessarily strong, it is a subset of $M(p_{xy})$ where p_{xy} is the least common ancestor in T of the leaves corresponding to x and y (denoted $lca(x,y)$) . A *factorising permutation* [2] τ is a permutation of the vertices such that any strong module M is a factor of τ (the vertices of M occur consecutively). A DFS of the modular decomposition tree orders the leaves as a factorising permutation. Maintaining a factorising permutation will be helpful to find a certificate.

3 Data structure

As we mentioned previously, the representation of a directed cograph we maintain along the algorithm is based on its di-cotree. We also maintain a factorising permutation. Note that it is not necessary for the recognition algorithm itself, but for finding a certificate within the desired complexity.
More precisely, as depicted on Fig. 3, each node q of the di-cotree stores 6 pointers:

– one pointer to its parent p in the di-cotree, and one pointer to its position in the list of children of p;
– one pointer to the first (resp. the last) element of its list of children in the di-cotree;
– one pointer to the first (resp. the last) vertex of Q in the factorising permutation.

The lists of children and the factorising permutation are doubly-linked lists (for sake of clearness, those lists are represented as simple lists in Picture 3). The list of children of any order node is ordered coherently with the order defined by the node, from the first order component to the last one. In addition to the list of its children, each node stores the number of its children.
Note that this data structure allows to answer adjacency queries on a pair x, y of vertices in $O(Max(d(x), d(y))$ time. To determine the adjacency relationship between x and y, we can first find $p_{xy} = lca(x, y)$. If its label is series or parallel, it is known; otherwise, we need to find which order component, the one of x or the one of y, is first in the order defined by p_{xy}. This two steps can be done in $O(Max(d(x), d(y))$ time. Indeed, the length of the path between any leaf l_x and the root r of the di-cotree is $O(d(x))$, because, on this path, of two consecutive nodes, at most one is labelled parallel. Moreover, the number of children of an order node q is $O(d(x))$, for any $x \in Q$.
The pointers from a node q to the factorising permutation allows to access in constant time to the list of vertices of Q, which is not possible in the di-cotree.

4 Dynamic vertex operations

This section deals with vertex modification, insertion or deletion. In the case of vertex deletion, the resulting graph $G - x$ is always a directed cograph and the algorithm consists in updating its di-cotree and the factorising permutation. Knowing how the di-cotree is modified under vertex deletion is helpful to characterise

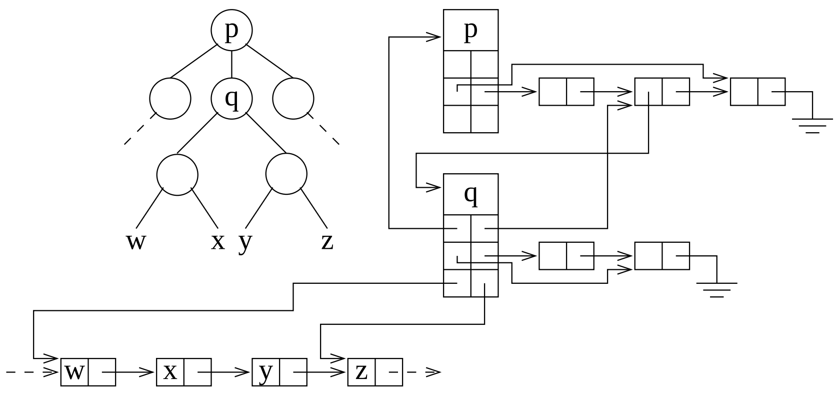


Fig. 3. The data structure maintained by the algorithm. Though it is not represented in the picture, the lists of children of a node as well as the factorising permutation are stored in doubly-linked lists. In addition, any node of the tree stores the number of its children.

the cases where the insertion of vertex x is possible. Theorem 4 is the basis of the insertion algorithm that either updates the di-cotree (and the factorising permutation) or finds a certificate that $G + x$ is not a directed cograph. For sake of simplicity, the certificate consists in a set of 4 vertices that induces a subgraph containing a forbidden subgraph of Fig. 2. Pushing further the algorithm, an exact forbidden subgraph can be found. The complexity of the deletion algorithm, as well as the insertion algorithm, is $O(d(x))$ where x is the vertex to be deleted or inserted.

4.1 Deleting a vertex

As already noticed, the deletion operation only requires to update the di-cotree T of G to obtain the di-cotree T' of G' (see fig. 4.1). It can be done in $O(d(x))$ as follows (see [16] for a similar algorithm). The case where x is the only vertex is trivial. Otherwise, let q be the parent node of x in T.

1. If x has at least 2 siblings, then x is removed from T.
2. Otherwise, let p be the sibling of x.
   (a) If q is the root of T or the label of $parent(q) = \tilde{q}$ is different from the one of p, nodes x and q are removed from T. If q is the root of T, then p becomes the root of T'. Otherwise, p is inserted in the children of $\tilde{q}$ in the exact place of q (it is crucial if $\tilde{q}$ is an order node).
   (b) If $label(\tilde{q}) = label(p)$, nodes x, q and p are removed from T. The children of p are inserted in the children of $\tilde{q}$, instead of q. If $\tilde{q}$ is an order node, then the relative order of the children of p has to be respected and they must be inserted in the children of $\tilde{q}$ as an interval in the exact place of q.

One can ensure that the new tree T' we built above is indeed the unique di-cotree of G' by checking that the following properties are satisfied: i) no node of T' has the same label as its parent, all the internal nodes have at least two children, and all the nodes are labelled series, order or parallel (i.e. T' is a valid di-cotree); ii) the adjacencies induced by T' are exactly the adjacencies of $G[V \setminus \{x\}]$.
For complexity issues, the case where p and $\tilde{q}$ have the same label (case 2b above) has to be handled carefully: only nodes containing neighbours of x can be touched. If q is not a parallel node, the children of p are linked to x. They can be disconnected from p and substituted for q in the children of $\tilde{q}$. If q is a parallel node, its siblings are linked to x. They can be disconnected from $\tilde{q}$ and reconnected as new children of p (at their right place if $\tilde{q}$ is an order node, see Fig.4.1). Finally p replaces $\tilde{q}$.
Updating the factorising permutation reduces to deleting x from it. Note that the only nodes remaining in T after the deletion of x that have to change their pointers are the ancestors of x for which x is an extremity of their corresponding segment in the factorising permutation. For these nodes, the pointer toward x has to be changed to a pointer on the previous (resp. next) vertex of the factorising permutation if x is the last

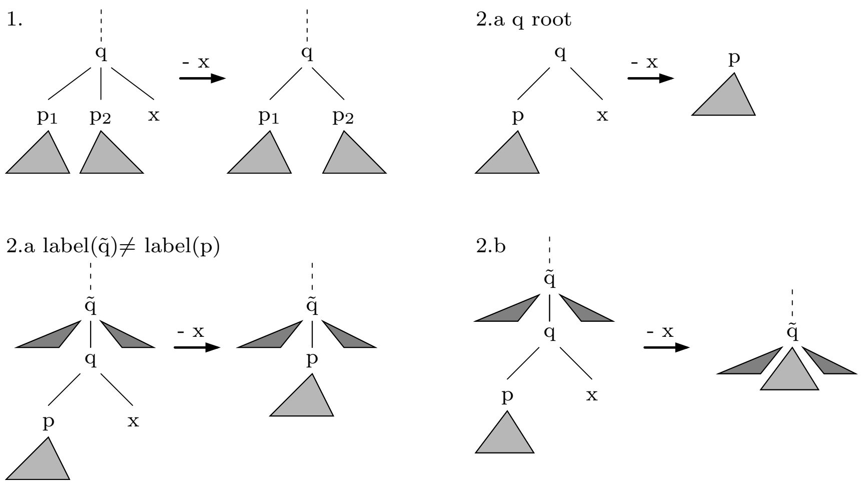


Fig. 4. Modifications of the modular decomposition tree under vertex deletion.

(resp. first) vertex of the segment corresponding to the considered node. Since the number of ancestors of x is $O(d(x))$, the complete update of pointers toward the factorising permutation can be done in $O(d(x))$ time. Only the parent or the grand-parent of x in T may have to update the number of their children. In case 1, $\mathcal{C}(q)$ is decreased by one; in case 2.b, $\mathcal{C}(p)$ is added to $\mathcal{C}(\tilde{q})$. This can be done in constant time.

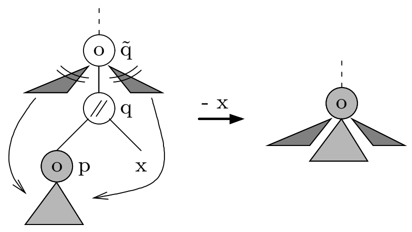


Fig. 5. Updating the di-cotree in $O(d)$ time under vertex deletion. Nodes are labelled O for order and // for parallel.

4.2 Adding a vertex

The main difficulty of the fully dynamic algorithm presented in this paper consists in maintaining a di-cotree under vertex insertion. Theorem 4 characterises the cases where given a directed cograph G, a vertex x and its neighbourhoods, the augmented graph $G + x$ remains a directed cograph. As done in [4], the algorithm first proceeds with a marking step of the di-cotree T of G. Then it tests whether the marks satisfy Theorem 4. In the positive, the di-cotree is updated; otherwise a certificate that $G+x$ is not a directed cograph is given.

Theorem 4. *Let $G = (V, E)$ be a directed cograph and T be its di-cotree. Let $x \notin V$ be a vertex and $N^-(x)$, $N^+(x)$ be its in and out-neighbourhoods. $G' = G + x$ is a directed cograph iff for any node p of T one of the following conditions holds:*

1. *P is uniform wrt. x;*
2. *P is mixed wrt. x and has a unique mixed child f such that $F \cup \{x\}$ is a module of $G'[P \cup \{x\}]$;*

3. P is mixed wrt.x, has no mixed children and either
   (a) there exists a unique non-empty set $\mathcal{S} \subsetneq \mathcal{C}(p)$ of children of p such that $S = \bigcup_{k \in \mathcal{S}} K$ is uniform wrt. x and $S \cup \{x\}$ is a module of $G'[P \cup \{x\}]$,
   (b) or there exists a non-empty set $\mathcal{S} \subsetneq \mathcal{C}(p)$ of children of p such that $S \cup \{x\}$, $(P \setminus S) \cup \{x\}$ are both modules of $G'[P \cup \{x\}]$, where $S = \bigcup_{k \in \mathcal{S}} K$.

A node of T satisfying condition 2 of Theorem 4 is called a *single mixed* node, and a *terminal mixed* node if it satisfies condition 3. It is worth to note that cases 3.(a) and 3.(b) of Theorem 4 are disjoint. In case 3.(b), p has to be an order node. Otherwise, p would be uniform wrt. x. Moreover, an order node p that satisfies Condition 3.(b) does not satisfy Condition 3.(a). Indeed, the unicity of the set $\mathcal{S}$ of Condition 3.(a) would not be satisfied: set $\mathcal{S}$ and set $\mathcal{C}(p) \setminus \mathcal{S}$ of Condition 3.(b) would both suit for condition 3.(a). Corollary 1 bellow shows that, if $G + x$ is a directed cograph, the mixed nodes cannot be spread anywhere in T and there is a unique terminal mixed node.

Theorem 4 is an equivalence. Corollary 1 follows from the direct implication and is useful to prove the converse implication of Theorem 4. Thus, we first prove that the conditions of Theorem 4 are necessary, then we prove that these conditions imply Corollary 1, and finally we prove the converse implication of Theorem 4.

Proof of the direct implication of Theorem 4: $\Longrightarrow$. Assume $G' = G + x$ is a directed cograph. Let T' be its modular decomposition tree and q' the parent of x in T'. Deleting x in G', we obtain, as described in Section 4.1 and Fig. 4.1, the modular decomposition tree T of $G' - x = G$. The transformation of T' in T establishes some correspondences between the strong modules of G' and the strong modules of G. We use these correspondences to show that the conditions of Theorem 4 are satisfied.
Before successively considering the 3 cases of Section 4.1, we first distinguish the case where q' is the root of T', which is a particular case of case 1 and case 2.a of Section 4.1. In this case, the strong modules of G are exactly the strong modules of G' which do not contain x. Namely, they are the strong modules of G' corresponding to the nodes of T' different from the root and l_x. Thus, they are all uniform and satisfy condition 1 of Theorem 4.
From now on, q' is supposed not to be the root of T'. Let $\tilde{q}'$ be its parent. After the deletion of x, $\tilde{q}'$ remains a node of the tree (see Fig. 4.1). We rename this node by $\tilde{q}$ in T. In other words, $\tilde{q}$ is the node of T corresponding to the strong module $\widetilde{Q}' \setminus \{x\}$ of G.

Claim. $\tilde{q}$ is mixed wrt. x.

Proof. Indeed, the labels of $\tilde{q}'$ and q' are different which implies that the vertices of $\widetilde{Q}' \setminus Q'$ and the vertices of $Q' \setminus \{x\}$ have not the same adjacencies with x. It follows that $\widetilde{Q} = \widetilde{Q}' \setminus \{x\}$ is mixed.

Claim. Let $k \in Anc_T(\tilde{q}) \setminus \{\tilde{q}\}$, k is single mixed.

Proof. Let k_m be the unique child of k being an ancestor of $\tilde{q}$. Since k_m is an ancestor of q, k_m is mixed. From section 4.1 (see Fig. 4.1) $K_m \cup \{x\}$ is a strong module of G', it is consequently a module of $G'[K \cup \{x\}]$. Thus k is single mixed.
Remark. It follows that any child f of k different from the unique mixed child k_m of k is uniform, and then its descendants are as well.

– If x has at least two siblings in T' (see case 1. of Section 4.1 and Fig. 4.1), then $Q' \setminus \{x\}$ is a strong module of G. We use q to denote its corresponding node in T. Let us examine $\tilde{q}$ and its descendants. Let u be a child of $\tilde{q}$ different from q, then U is a module of G' which does not contain x. It follows that u and its descendants are uniform. For the same reasons, the children of q are uniform. For $\tilde{q}$ and q, we have to distinguish 2 cases. If q is an order node and x is not an extremal component of q', let $\mathcal{S} = \{f \in \mathcal{C}(q') \mid f <_{q'} l_x\}$, where $<_{q'}$ is the order defined by q' on its children, and let $S = \bigcup_{f \in \mathcal{S}} F$. $S \cup \{x\}$ and $(Q \setminus S) \cup \{x\}$ are modules of $G'[Q \cup \{x\}] = G'[Q']$. Thus, by definition, q is terminal mixed, it satisfies condition 3.(b) of Theorem 4. As $Q \cup \{x\} = Q'$ is a module of $G'[\widetilde{Q} \cup \{x\}] = G'[\widetilde{Q}']$, $\widetilde{Q}$ is single mixed. Otherwise, if q' is a parallel or a series node or q' is an order node but x is an extremal component of q', then $Q = Q' \setminus \{x\}$ is a module of G' which does not contain x. Thus, Q is uniform, it satisfies condition 1 of Theorem 4. And since $Q \cup \{x\} = Q'$ is a module of $G'[\tilde{Q} \cup \{x\}] = G'[\widetilde{Q'}]$, $\tilde{q}$ is terminal mixed, it satisfies condition 3.(a) of Theorem 4 with $\mathcal{S} = \{q\}$. Such a subset $\mathcal{S}$ of children of $\tilde{q}$ is unique. Indeed, since S has to be uniform, from case 1 of Section 4.1 (see Fig 4.1), $S \subseteq Q$. Since $\mathcal{S}$ is a subset of children of $\tilde{q}$, then $\mathcal{S} = \{q\}$.

– If x has a unique sibling p' in T' and if p' and $\tilde{q}'$ have different labels (see case 2.(a) of section 4.1 and Fig. 4.1), then P' is a strong module of G. We denote p its corresponding node in T. Similarly to the previous case, the children of $\tilde{q}$ different from p are uniform. In addition, P is a strong module of G' and P does not contain x, then p is uniform. Finally, any descendant of $\tilde{q}$ is uniform. As $P \cup \{x\} = Q'$ is a module of $G'[\tilde{Q} \cup \{x\}] = G'[\widetilde{Q'}]$, and since $\tilde{q}$ is mixed then it is terminal mixed, it satisfies condition 3.(a) of Theorem 4 with $\mathcal{S} = \{p\}$. Again, such a set $\mathcal{S}$ is unique. Indeed, S has to be uniform, then $S \subseteq P'$ or $S \subseteq (\widetilde{Q'} \setminus Q')$. $S \cup \{x\}$ has to be a module of $G'[\tilde{Q} \cup \{x\}] = G'[\widetilde{Q'}]$, thus $S = P' = P$.

– If x has a unique sibling p' in T' and if p' and $\tilde{q}$ have the same label (see case 2.(b) of section 4.1 and Fig. 4.1), then, like above, all the descendants of $\tilde{q}$ are uniform. Let $\mathcal{S} = \mathcal{C}(p')$ and $S = P'$. $S \cup \{x\} = Q'$ is a module of $G'[\tilde{Q} \cup \{x\}] = G'[\widetilde{Q'}]$. As $\tilde{Q}$ is mixed, it follows that it is terminal mixed, it satisfies condition 3.(a) of Theorem 4 with $\mathcal{S} = \{p\}$. The proof of the unicity of such a set $\mathcal{S}$ in the current case is similar to the one of the previous case.

□

Corollary 1. *If $G + x$ is a directed cograph, there exists a unique mixed node q such that the set of mixed nodes of T is exactly $Anc(q)$. Node q is the unique terminal mixed node, the only mixed node without mixed children.*

Proof of Corollary 1: If $G + x$ is a directed cograph, then the conditions of Theorem 4 are satisfied. Assume there exist two distinct mixed nodes q_1 and q_2 of T such that $q_1 \notin Anc(q_2)$ and $q_2 \notin Anc(q_1)$. Then $p = lca(q_1, q_2)$ is different from both q_1 and q_2. Let p_1 (resp. p_2) be the unique node in $\mathcal{C}(p) \cap Anc(q_1)$ (resp. in $\mathcal{C}(p) \cap Anc(q_2)$). By definition of a mixed node, any ancestor of a mixed node is mixed. Since $p_1 \in Anc(q_1)$, p_1 is mixed, and similarly p_2 is mixed. Node p is mixed and has two mixed children, which refutes the conditions of theorem 4.
Let q be the lowest mixed node in T. All the mixed nodes belong to $Anc(q)$, and since any ancestor of a mixed node is mixed, the set of mixed nodes is exactly $Anc(q)$. □

Proof of the converse implication of Theorem 4: $\Longleftarrow$. Under the conditions of theorem 4, we build a di-cotree T' by inserting x in T. We show that the adjacencies induced by T' between the vertices of V are the adjacencies defined by E (also induced by T), and the adjacencies induced by T' between x and the vertices of V are the relations defined by $N^+(x)$ and $N^-(x)$. Since T' is built using only parallel, series and order nodes and since T' is a modular decomposition tree, T' is a di-cotree. Moreover, since the modular decomposition tree of a graph is unique, then T' is the modular decomposition tree of G' which is a directed cograph.
As shown in the proof of Corollary 1, if the conditions of Theorem 4 are satisfied, then there exists a mixed node q of T such that the mixed nodes of T are exactly the nodes on the path P_q^r from q to the root r of T.

Lemma 1. *$Q \cup \{x\}$ is a strong module of G'.*

Proof of Lemma 1: $\forall p \in Anc(q)$, p is mixed. If p is not the root, its parent $\tilde{p}$ is mixed and has a mixed child. Since the conditions of Theorem 4 are satisfied, $\tilde{p}$ is single mixed. Thus p is the unique mixed child of $\tilde{p}$ and $P \cup \{x\}$ is a module of $G'[\widetilde{P}]$. It follows by recursion that $Q \cup \{x\}$ is a module of G'. Let us show that $Q \cup \{x\}$ is strong. Suppose for contradiction that there exists a module M' of G' that overlaps $Q \cup \{x\}$. Necessarily, $M = M' \setminus \{x\} \neq \varnothing$. It follows that M is a module of G, and since Q is strong, M does not overlap Q. Since $M' \setminus (Q \cup \{x\}) \neq \varnothing$, then $M \not\subseteq Q$. It follows that $M \cap Q = \varnothing$ and $x \in M'$, or $Q \subseteq M$. In the first case, $(Q \cup \{x\}) \setminus M' = Q$ is a module of G'. In the latter case ($Q \subseteq M$), since M' overlap $Q \cup \{x\}$, then $x \notin M'$. Thus, $(Q \cup \{x\}) \cap M' = Q$ is a module of G'. As Q does not contain x, in both cases Q is uniform : contradiction. □

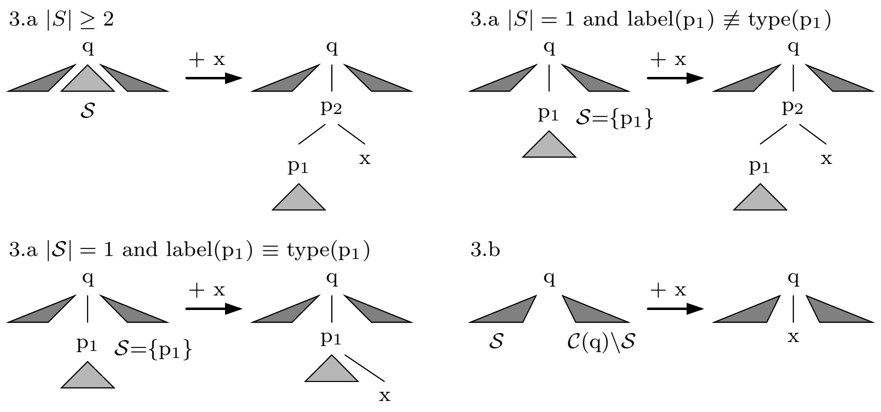
Consequently, building T' reduces to inserting x in T_q. In this way, we obtain the desired adjacencies between x and the vertices of $V \setminus Q$. We now discuss how to insert x in T_q (see Fig. 4.2).

– If q satisfies condition *3.(b) of theorem 4*, then $S \cup \{x\}$ and $(Q \setminus S) \cup \{x\}$ are modules of $G'[Q \cup \{x\}]$. From Claim 2, it follows that S and $Q \setminus S$ are modules of $G[Q]$ Thus, $\mathcal{S}$ and $\mathcal{C}(q) \setminus \mathcal{S}$ are composed of consecutive children in the order defined by q. Wlog., assume that $\mathcal{S}$ is the lower interval in this order. Since $S \cup \{x\}$ is a module of G', $Q \setminus S \subseteq N^+(x)$ and since $(Q \setminus S) \cup \{x\}$ is a module of G', $S \subseteq N^-(x)$. Therefore, inserting the leaf l_x as a child of Q between $\mathcal{S}$ and $\mathcal{C}(q) \setminus \mathcal{S}$, we obtain the wished adjacencies between x and the vertices of Q.
– If q satisfies condition *3.(a) of theorem 4*, then $S \cup \{x\}$ is a module of $G'[Q \cup \{x\}]$. From Claim 2, it follows that S is a module of $G[Q]$. Then, vertices of $Q \setminus S$ have the same adjacency relationship with x than the one they have with the vertices of S. For this reason, and because S is uniform, we can insert x as follows.
  - If $|\mathcal{S}| \geq 2$, we make the nodes of $\mathcal{S}$ children of a new node p_1 without changing their relative order if $label(q) = order$. p_1 is assigned the label of q. We create a new node p_2 which is assigned the label corresponding to the adjacency relationship between x and the vertices of S. Formally, the correspondence relationship between labels and types of nodes will be denoted $\equiv$ and is defined by: Series $\equiv$ InOut, Parallel $\equiv$ None, Order $\equiv$ In and Order $\equiv$ Out. Note that the label of p_2 is necessarily different from the one of p_1, otherwise $\mathcal{S}$ would not be unique as required by condition *3.(a)*. We make l_x and p_1 children of p_2, in the right order if $label(p_2) = order$. Finally, p_2 becomes a child of q. If q is an *order* node, the nodes of $\mathcal{S}$ are an interval of the order defined by q and p_2 has to be inserted in the children of q in the place of this interval.
  - If $|\mathcal{S}| = 1$, let p_1 be its unique element. If the adjacency relationship between x and the vertices of P_1 correspond to the label of p_1, then we insert l_x as a child of p_1, at the right place if p_1 is an *order* node. Otherwise, we create a new node p_2 which is assigned the label corresponding to the adjacency between x and the vertices of P_1. And we make l_x and p_1 children of this new node, in the right order if $label(p_2) = order$.

One can check that inserting x in T in this way, we did not change the adjacencies between the vertices of V and we obtained the wished adjacencies between x and the vertices of V. As we used only *parallel*, *series* and *order* nodes, we obtained a di-cotree which is the modular decomposition tree of G'. Thus, G' is a directed cograph. □

The marking process. The first step of our algorithm colours nodes of the modular decomposition tree T according to the neighbourhood of the vertex x to be inserted. This preliminary step is a straightforward extension of the marking process of [4].

Initially each leaf $l_y = \{y\}$, such that $y \in N(x)$, is coloured red. Depending on the adjacency relationship between y and x, these leaves are given a type: $type(l_y) = In$ if $yx \in E$ and $xy \notin E$; $type(l_y) = Out$ if $xy \in E$ and $yx \notin E$; or $type(l_y) = InOut$ if $xy \in E$ and $yx \in E$. The process is a bottom-up search: each red node p forwards its type to its parent node q and depending on the different types received by q, a colour is given to q. The first time an internal node receives a type, it is coloured black. A node q becomes red if all its children have the same type (ie. the corresponding set of vertices Q is uniformly linked to x). A red node receives the type of its children. Once a red node has forwarded its type to its parent, it becomes grey. Note that if it happens that the root of T becomes red, then it is coloured grey straight after. In order to prepare



3.a $|S| \geq 2$
q
+ x
q
$\mathcal{S}$
p_2
p_1
x
3.a $|S| = 1$ and label(p_1) $\not\equiv$ type(p_1)
q
+ x
q
p_1
$\mathcal{S}$={p_1}
p_2
p_1
x
3.a $|\mathcal{S}| = 1$ and label(p_1) $\equiv$ type(p_1)
q
+ x
q
p_1
$\mathcal{S}$={p_1}
p_1
x
3.b
q
+ x
q
$\mathcal{S}$
$\mathcal{C}$(q)\$\mathcal{S}$
x

Fig. 6. Modification of the di-cotree under vertex insertion. $\equiv$ denotes the correspondence relationship between labels and types of nodes.

the possible insertion of x, a list of the grey children is maintained for each node handled by the marking process. The process ends when there are no red nodes left.
For sake of simplicity, let us say that the default colour is *white*. Also notice that the absence of type can be considered as a non-adjacency type, we will use the notation $type(p) = None$. It is important to note, for complexity reasons, that all the nodes visited by the marking process will have a colour different from white, and a type different from *none* at the end of this step. This follows from the fact that only the leaves which are linked to x are parsed and coloured red at the beginning of the process.

It is worth to note that a marking technique similar to the one of [4] is used in [15] to update the modular decomposition tree of a graph under vertex insertion. The main difference between the two processes is that in the case of general graphs [15] the leaves which are not linked to x need to be typed and to forward their type to their parent, as the linked leaves do. This results in an $O(n)$ time complexity instead of $O(d(x))$ time for the cograph recognition problem [4]. The marking step of [15] has been appplied in [6] to design a fully dynamic algorithm for recognition of permutation graphs that runs in $O(n)$ time per update.

```
**Type**($G$, $T$, $\mathcal{R}$ a set of typed red leaves)
 1. While some red node $p$ exists Do
 2.       $colour(p) \leftarrow grey$
 3.       If $p$ is not the root of $T$ Then
 4.             Let $q$ be the parent node of $p$
 5.             Add $p$ to the list $greyChild(q)$
 6.             Increase $\#type(q, type(p))$ by one
 7.             If $\#type(q, type(p)) = \#child(q)$ Then
 8.                    $colour(q) \leftarrow red$ and $type(q) \leftarrow type(p)$
 9.             Else $colour(q) \leftarrow black$
10. End of while
```

Fig. 7. Marking process.

In our marking process, each node stores the list and the number of its grey children, and three counters $\#type(q, t)$ for any type $t \in \{In, Out, InOut\}$ (eg. $\#type(q, In)$ indicates the number of children of q whose

type is In).
We claim without proofs the following basic properties of the coloured tree T^c resulting from the marking process. They are necessary for understanding the insertion algorithm and the production of a certificate.

Claim. The nodes of T^c which are uniformly linked to x are exactly the grey nodes; the nodes of T^c which are uniformly not linked to x are white; and the black nodes are mixed.

Remark. A white node can be mixed. And a white node is mixed iff it is linked.

It follows that the mixed nodes are white or black. The set of black nodes will be denoted $\mathcal{B}$.

Lemma 2. *After the marking process, a white node which is linked to x has a black descendant.*

Proof of Lemma 2: Let w be a white node linked to x. Since w is white, it is not uniformly linked to x: it is mixed. Let v be a minimal (for inclusion) mixed node of T_w. The children of v are all uniform and since v is mixed, at least one of its children v_c is uniformly linked to x. v_c is grey and v is black. □

The running time of Routine `Type` (see Fig. 4.2) is $O(d(x))$, and the number of grey nodes and of black nodes are both bounded by $O(d(x))$. The part of the di-cotree T parsed by Routine `Type` is made of the black nodes, the grey nodes and the edges between them. First consider the tree restricted to grey nodes. This is a forest in which the leaves are linked to x, and the internal nodes have at least two children. Since the number of leaves is $O(d(x))$, so it is for the number of grey nodes. As a black node has at least one grey child and a node has at most one parent, then the number of black nodes is also $O(d(x))$. Each edge of the restriction of T to grey and black nodes is crossed at most once during Routine `Type` and each node is treated in constant time. It follows that the running time of Routine `Type` is $O(d(x))$.

Testing the insertion. In order to test whether the insertion of x is possible or not, Theorem 5 expresses the conditions of Theorem 4 in terms of coloured node. Our algorithm checks the conditions of Theorem 5 in T^c, considering only the colours of the nodes.

Theorem 5. *Let G be a directed cograph and T its di-cotree. $G + x$ is a directed cograph iff there exists a black node q of T such that:*

1. *every black node belongs to P^r_q,*
2. *any black node of $P^r_{parent(q)}$ is a single mixed series or order node,*
3. *any white node of $P^r_{parent(q)}$ is a parallel node and*
4. *q is a terminal mixed node.*

Clearly, the conditions of Theorem 5 are very close to the conditions of Theorem 4. Lemma 3 states the correspondences, when $G + x$ is a directed cograph, between the colour and the label of single mixed nodes of T^c. The main difference between the two theorems is that the conditions of Theorem 5 do not make explicitly the white parallel nodes of $P^r_{parent(q)}$ to be single mixed. Lemma 4 shows that under conditions of Theorem 5, the white parallel nodes of $P^r_{parent(q)}$ are single mixed. The fact that we do not have to check this condition is crucial for the complexity of $O(d(x))$ time per insertion.

Lemma 3. *Let p be a single mixed node of T^c. Node p is either a white parallel node, a black series node, or a black order node.*

Proof of Lemma 3: Let p be a series or order single mixed node, and f its unique mixed child. Then, $F \cup \{x\}$ is a module of $G'[P]$. It follows that any child h of p different from f is uniformly linked to x. Hence, h is grey and p is black.
Let now p be a parallel single mixed node. Then, $F \cup \{x\}$ is a module of $G'[P]$. It follows that any child h of p different from f is not linked to x. Hence, h is white. Since f is mixed, f is not grey, and then p is white. □

Lemma 4. *If there exists a black node q such that any black node belongs to P_q^r and any white node of $P_{parent(q)}^r$ is parallel, then the white nodes of P_q^r are single mixed nodes.*

Proof of Lemma 4: Let p be a white node of P_q^r. Since $p \in Anc(q) \setminus \{q\}$, p is mixed and has a mixed child $p_1 \in Anc(q)$. Let $h \in \mathcal{C}(p) \setminus \{p_1\}$. Assume h is linked to x, then, from Lemma 2, h has a black descendant f which is not on the path P_q^r: contradiction. Thus, h is not linked to x, as well as its descendants, which are all white (see Claim 4.2). Moreover, since p is a white node of P_q^r, p is a *parallel* node. Node p has a unique mixed child p_1 and its other children are not linked to x. It follows that $P_1 \cup \{x\}$ is a module of $G'[P \cup \{x\}]$: p is single mixed. $\square$

Thanks to the two lemmas above, we show the equivalence between the conditions of Theorem 4 and the conditions of Theorem 5, which prove Theorem 5.

Proof of Theorem 5: $\Longrightarrow$. If $G + x$ is a directed cograph, then the conditions of Theorem 4 are satisfied. From Corollary 1, the mixed nodes of T induce a path from the root to a certain mixed node q which is coloured black. Indeed, q is mixed and its children are uniform, thereby, at least one of its children is uniformly linked to x. This child is coloured grey and q is coloured black. Since the black nodes are mixed, and since, from Corollary 1, the mixed nodes lie on the path P_q^r, then the black nodes all belong to P_q^r. Condition 1 is satisfied. The conditions of Theorem 4 imply that the nodes of $P_{parent(q)}^r$ are single mixed nodes. Lemma 3 implies that the black single mixed nodes are series or order nodes, and the white single mixed nodes are parallel nodes. Conditions 2 and 3 are satisfied. Condition 4 is the same as condition 3 of Theorem 4.
$\Longleftarrow$. We show that if the conditions of Theorem 5 are satisfied, then the conditions of Theorem 4 are satisfied. Since q is terminal mixed, by definition, it satisfies Condition 3 of Theorem 4. Its descendants are uniform and satisfy Condition 1 of Theorem 4. From Condition 2 of Theorem 5, the black nodes of $P_{parent(q)}^r$ are single mixed. From Conditions 1 and 3 of Theorem 5, and since node q is black, Lemma 4 applies. Then the white nodes of $P_{parent(q)}^r$ are single mixed. Since $P_{parent(q)}^r$ contains only black or white nodes (all the nodes on $P_{parent(q)}^r$ are mixed), the nodes of $P_{parent(q)}^r$ satisfy Condition 2 of Theorem 4. It follows that the nodes of $T \setminus (Anc(q) \cup Des(q))$ are uniform, they satisfy Condition 1 of Theorem 4. $\square$

Routine `Check` (see Fig. 4.2) tests whether the conditions of Theorem 5 are satisfied or not. If these conditions are satisfied, the insertion of vertex x is handled by Routine `Insert`. If one of them is not satisfied, then a call to Routine `Find-Certificate` enables us to find a set Z of 3 vertices such that $G'[Z \cup \{x\}]$, with $G' = G + x$, contains one of the forbidden subgraphs of Fig. 2. Routines `Insert` and `Find-Certificate` are described further.
Let p be the current node in Routine `Check`. If p has already been visited (test Line 6), Condition 1 of Theorem 5 is not satisfied and G' is not a directed cograph. The tests of Line 7 and 8 check whether p satisfies Condition 2 or 3 of Theorem 5. Condition 4 is tested at Line 14.

Let us detail how to perform the test that a black series or order node is single mixed (Line 8), and the test that q is a terminal mixed node (Line 14).
Let p be a black series node or a black order node. For p to be a single mixed node, all but one of its children have to be coloured grey. If p is a series node, the children distinct from the only non-grey child q should be typed *InOut*. If p is an order node, the children that occur before (resp. after) q in the order defined by p have to be typed *In* (resp. *Out*).
Let q be the node tested by Routine `Check` at Line 14. There is no constraint on the label of q. For any $t \in \{In, Out, InOut\}$, we denote $\#type(t)$ the number of children of q typed t, and we denote $\#grey$ the number of children of q coloured grey. We can test whether q is terminal mixed as follows.

- if q is a parallel node (see Fig. 4.2): check that $\#type(In) = \#grey$ or $\#type(Out) = \#grey$ or $\#type(InOut) = \#grey$ (since in that case, if q is terminal mixed, any node of $\mathcal{S}$ is a grey node, where $\mathcal{S}$ is the set defined in condition 3.a of Theorem 4);
- if q is a series node (see Fig. 4.2): check that $\#type(In) + \#type(InOut) = |\mathcal{C}(q)|$ or $\#type(Out) + \#type(InOut) = |\mathcal{C}(q)|$ or $\#type(In) = \#type(Out) = 0$;

```
**Check**($G$, $T^c$, $\mathcal{B}$, $x$)
1.  $bottom \leftarrow r$, where $r$ is the root of $T^c$
2.  While some node $q$ in $\mathcal{B}$ exists Do
3.      $p \leftarrow q$ and remove $q$ from $\mathcal{B}$
4.      While $p \neq bottom$ Do
5.          $p \leftarrow parent(p)$
6.          If $p$ has been visited Then Find-Certificate($p$)
7.          If $p$ is a white non-parallel node Then Find-Certificate($p$)
8.          If $p \in \mathcal{B}$ is not a single mixed node Then Find-Certificate($p$)
9.          If $p \in \mathcal{B}$ Then Remove $p$ from $\mathcal{B}$
10.         Mark $p$ as visited
11.     End of while
12.     $bottom \leftarrow q$
13. End of while
14. If $q$ is a terminal mixed node Then Insert($x$, $q$, $T^c$)
15. Else Find-Certificate($q$)
```

Fig. 8. Testing the insertion of vertex x. T^c is the coloured di-cotree of G and $\mathcal{B}$ the set of black nodes of T^c.

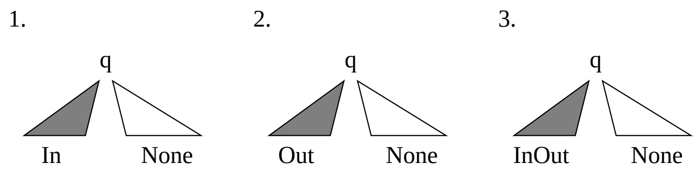


Fig. 9. The 3 cases where q is a parallel terminal mixed node.

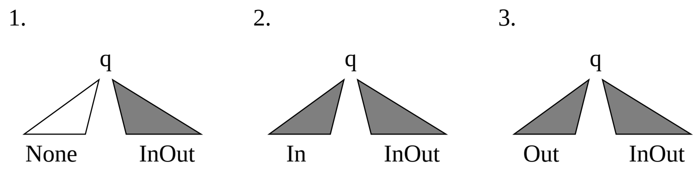


Fig. 10. The 3 cases where q is a series terminal mixed node.

– if q is an order node (see Fig. 4.2): first, test if either $\#type(InOut) + \#type(In) + \#type(Out) = |\mathcal{C}(q)|$ or $\#type(InOut) = 0$. Then check whether the first (wrt. the relative order of q) $\#type(In)$ children of q are typed In and the last $\#type(Out)$ are typed Out.

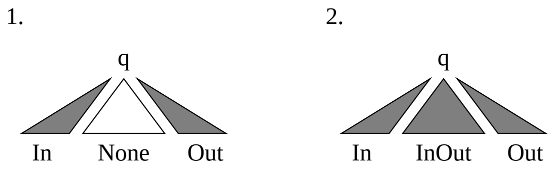


Fig. 11. The 2 cases where q is an order terminal mixed node.

Testing if a black series node p is single mixed is performed in constant time. If p is an order node, we need to parse its children only if $|\mathcal{C}(p)| = \#grey + 1$ (otherwise, p is not single mixed), then it takes $O(d(x))$ time. The test whether q is terminal mixed is performed in constant time when q is series or parallel. If q is an order node, the test uses two searches in the list of children of q. The first search checks whether the first $\#type(In)$ children of q are typed In and the second search checks whether the last $\#type(Out)$ children of q are typed Out. If a white node is encountered by any of the two searches, then the search stops. Consequently, only the grey children of q are visited plus eventually one white node. Since the number of grey nodes is $O(d(x))$, Routine `Check` runs in $O(d(x))$ time.

Inserting a vertex. When Routine `Check` determines that $G + x$ is a directed cograph, Routine `Insert` has to perform the insertion of x in the di-cotree T of G in order to obtain the di-cotree T' of $G' = G + x$. In this case, the conditions of Theorem 4 are satisfied. Let q be the only terminal mixed node (the *bottom* node in Routine `Check`). From Lemma 1, $Q \cup \{x\}$ is a module of G'. It follows that the modifications occur in the subtree T_q of T rooted at q. The update of T_q is made exactly as described in the proof of Theorem 4, and depicted in Fig. 4.2. Note that when the insertion is possible, set $\mathcal{S}$ of Theorem 4 is determined by Routine `Check`. As for the vertex deletion, to update the di-cotree in case 3.a when $|\mathcal{S}| \geq 2$ (see Fig. 4.2), we have to carefully handle the moving of non-neighbourhood of x. If the nodes of $\mathcal{S}$ have no type, then we disconnect the nodes of $\mathcal{C}(p) \setminus \mathcal{S}$ from p and reconnect them on a new node; otherwise, the disconnection-reconnection manipulation is applied to the nodes of $\mathcal{S}$.

At this point, we have to distinguish the di-cotree, which is a mathematical concept, from its representation in memory. The main difference between the two is that the list of children of a node p used in the representation induced an order on the children of p which is not relevant when p is a parallel or series node. In the previous paragraph, we described how to maintain the di-cotree (ie. the parent relationship) but we do not precise how to maintain a representation of the di-cotree as it is straightforward to imagine a way to do so. We will now show how to maintain a factorising permutation and the pointers from the nodes of the di-cotree toward it. Doing so, we will also update the representation of the di-cotree (the representation resulting from the update of the di-cotree described in the previous paragraph) so that we keep the property that the factorising permutation is the order in which we encounter the leaves of T' in the depth first search respecting the orders of the lists of children in the representation.

Updating the factorising permutation, the pointers from the node of the di-cotree toward it, and the di-cotree representation can be done in $O(d(x))$ time. We show, as an example, how to deal with this update in the case where the terminal mixed node q satisfies condition 3.a of Theorem 4 and $|\mathcal{S}| \geq 2$. The other cases are simpler since, in those cases, there is no need to sort the children of q before inserting x in the factorising permutation. In the following, p_1 and p_2 denotes the nodes introduced in the proof of the converse implication of Theorem 4 (page 9), and depicted in Fig. 4.2.

– q is a parallel node (see example depicted in Fig. 4.2), we move node p_2 to the beginning of the list of children of q. Then, for each child u of p_1, we cut its corresponding interval in the factorising permutation,

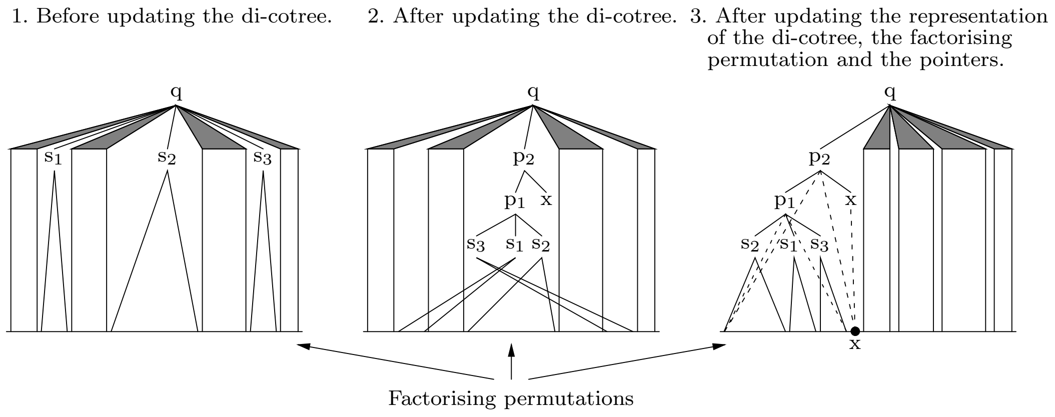

1. Before updating the di-cotree.
2. After updating the di-cotree.
3. After updating the representation of the di-cotree, the factorising permutation and the pointers.
q
s1
s2
s3
p2
p1
x
Factorising permutations

Fig. 12. Modification of the data structure under vertex insertion. q is the terminal mixed node and $\mathcal{S} = \{s_1, s_2, s_3\}$. The pointers are depicted as lines (dash lines for leaf x and nodes p_1 and p_2) from the nodes to the factorising permutation.

and move it to the beginning of the interval of q, and we move u to the beginning of the list of children of p_1. This guarantee the desired complexity since, in that case (q parallel), the children of p_1 are grey and thus, their number is $O(d(x))$. p_1 is assigned the first pointer of its last moved child and the second pointer of its first moved child.

- If p_2 is labelled order and x is its first child, then x is inserted in the factorising permutation before the first vertex of P_1, and p_2 is assigned pointers toward x and the last vertex of P_1. Vertex x is moved to the beginning of the list of children of p_2.
- Otherwise, x is inserted, in the factorising permutation, after the last vertex of P_1, and p_2 is assigned pointers toward the first vertex of P_1 and x. Vertex x is moved to the end of the list of children of p_2.

– If q is a series node, we proceed similarly. We move node p_2 to the beginning of the list of children of q. To guarantee the complexity, we cannot handle the children of p_1 which may be white. Instead, for each child of q different from p_2, we cut its corresponding interval in the factorising permutation, and move it to the end of the interval of q. The placement of x and the pointers of p_2 are the same as in the previous case.

– If q is an order node, the order of the children of q does not need to be changed. Node p_1 is assigned the first pointer of the first node of $\mathcal{S}$, and the second pointer of the last node of $\mathcal{S}$. Vertex x is inserted after the last vertex of P_1, and p_2 is assigned the first pointer of p_1 and a pointer toward x. Vertex x is moved to the end of the list of children of p_2.

Updating the number of children of the nodes of the di-cotree takes constant time. In case 3.a when $|\mathcal{S}| \geq 2$, if the nodes of $\mathcal{S}$ are linked to x, we know their number thanks to the counters used in the marking process. If the nodes of $\mathcal{S}$ are not linked to x, the nodes of $\mathcal{C}(q) \setminus \mathcal{S}$ are linked and their number is known. In both cases, by difference, we deduce both $|\mathcal{S}|$ and $|\mathcal{C}(q) \setminus \mathcal{S}|$. Thus we can set the number of children of q, p_1 and p_2 to their right values. The other cases are even simpler.

Finding a certificate. To avoid a heavy case by case analysis, we present a version of Routine `Find-Certificate`(p) which does not provide an exact certificate, but a set of 4 vertices which contains a forbidden graph of Fig. 2.

Lemma 5. *If $G' = G + x$ is not a directed cograph, a set $Z = \{a, b, c\}$ of 3 vertices can be found in $O(d(x))$ time such that $G'[Z \cup \{x\}]$ contains one of the graphs of Fig. 2.*

For each call to `Find-Certificate`, we show on 2 examples how to find a minimal certificate from the set Z returned. One can complete the case analysis and ensure that it is always possible to find a minimal

forbidden subgraph within the same complexity.
Assume Routine `Find-Certificate`(p) is also given the parameters P^r_{bottom} and P^p_q where *bottom* and q are the nodes respectively defined at Line 12 and 2 of Routine `Check`. Thanks to the lists of grey children for each node of T^c and the factorising permutation, the search is processed in $O(d(x))$ time. The call to `Find-Certificate` occurs: at Line 6, if the current node p has already been visited before; at Lines 7 and 8, if node p is not a single mixed node; at Line 15, if the last visited node q is not terminal mixed. For each call, we show how to find the 3 vertices a, b, c of Z.

The black nodes do not induce a path from the root (Line 6). In that case, p has to be a parallel node, otherwise, `Check` would have found out that p is not a single mixed node, since p has at least two mixed children. These two mixed children have already been visited by Routine `Check`. They are the child h of p on the path P^r_{bottom}, and the child h' of p on the path P^p_q. Nodes h and h' are black, otherwise, `Check` would have stopped before since they are not parallel. Thus, they both received a type from a grey child, say k and k' respectively. Let a be a vertex of K and b be a vertex of K'. Finally, since h' is mixed and k' is uniform, a vertex $c \in H' \setminus K'$ such that $type(c) \neq type(b)$ exists. See examples on Fig. 4.2.

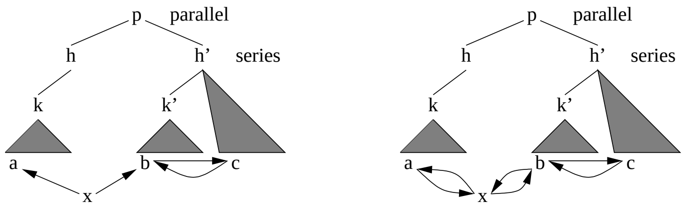


Fig. 13. Since p is a parallel node, h' is either a series or an order node. Assume that h' is a series node, therefore bc and cb exist. In the first example, the certificate is induced by $\{b, c, x\}$, in the second by $\{a, b, c, x\}$.

The current node is not single mixed (Line 7 or 8). Let p be the node currently visited by Algorithm `Check`. If we are in the first occurrence of the internal loop (Lines 4 - 11), then p is the parent of the node q we chose among the black nodes remaining in $\mathcal{B}$, at Line 2. Thus, p has a black child. If we are not in the first occurrence of the internal loop then, during the previous occurrence of the loop, a node h has been visited. If h is not black, then it is a white parallel node, otherwise the algorithm would have stopped while visiting h. It follows that h has a black child. Indeed, if h has a visited child h', h' is series, since h is parallel, and h' is black, since the algorithm did not stop while visiting h'. Otherwise, h has been visited during the first occurrence of the internal loop and has a black child which is the node q chosen among the nodes of $\mathcal{B}$ at Line 2 to initialise the internal loop. We proved that p either has a black child or a black grand-child, denoted $\tilde{h}$. Let b be a vertex of a grey child of $\tilde{h}$. And let c be a vertex of H such that $type(c) \neq type(b)$.
If p is not single mixed, then there exists $k \in \mathcal{C}(p) \setminus \{q\}$ such that $type(k)$ does not correspond to the label of p. More precisely, if p is a parallel node, $type(k) \neq None$; if p is a series node, $type(k) \neq InOut$; and if p is an order node, $type(k) \neq In$ (resp. $type(k) \neq Out$) and k is before (resp. after) q in the relative order of $\mathcal{C}(p)$. Let a be a vertex of K whose type does not correspond to the label of p. Note that if p is a parallel node, k is uniformly linked to x and any vertex $a \in K$ suits. See examples on Fig. 4.2.

The bottom node is not terminal mixed (Line 15). At this stage of the algorithm, we checked that the black nodes are on a path P^r_q from a node q, the bottom node, to the root r of T. Since q is the lowest black node, its children are grey or white. Lemma 2 implies that the white children of q are uniform. The call to $Find-certificate$ occurs when q is not terminal mixed.

– If q is a *parallel* or *series* node
  Then q has two children k and k' of different types, such that their types are different from the one corresponding to $label(q)$.

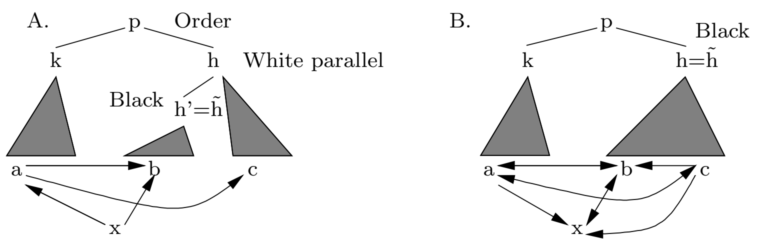


Fig. 14. In Case `A.`, since *type*$(c) \neq$ *Out*, $G'[\{a, c, x\}]$ is a certificate. In Case `B.` $G'[\{a, b, c, x\}]$ is a certificate.

– If q is an *order* node
  If q has a grey child typed *InOut* and a white child, let k and k' respectively be these children. Otherwise, q has two children k and k' with $k < k'$ and one of the following conditions is true:
  - *type*$(k) =$ *Out* and *type*$(k') =$ *In*
  - *type*$(k) =$ *Out* and *type*$(k') =$ *none* or *InOut*
  - *type*$(k) =$ *none* or *InOut* and *type*$(k') =$ *In*

We choose $a \in K$ and $b \in K'$. For this call to find certificate, c is not necessary, since $G'[\{a, b, x\}]$ is an exact forbidden graph of Fig. 2. See examples on Fig. 4.2.

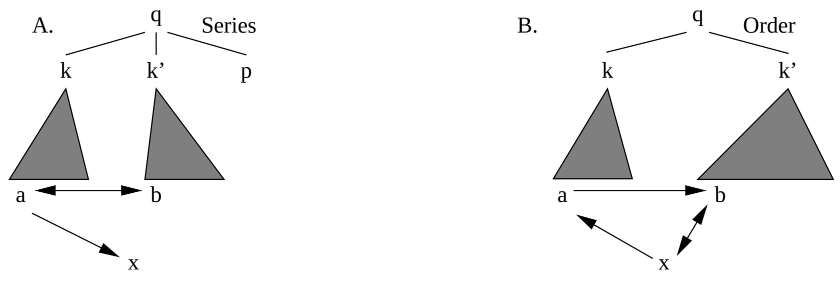


Fig. 15. In Case `A.`, since *type*$(k) =$ *In* and *type*$(k') =$ *None*, $G'[\{a, b, x\}]$ is a certificate. In Case `B.`, since *type*$(k) =$ *Out*, *type*$(k') =$ *InOut* and $k < k'$ in the order defined by q, then $G'[\{a, b, x\}]$ is a certificate.

For each call to `Find-Certificate`, the set Z described above can be found in $O(d(x))$ time. We parse the tree from the node p on which the call occurs, and we consider a constant number of nodes which are children or grand-children of p. For each node considered, we make at most one search in its grey children, and at most one search in its corresponding segment in the factorising permutation. This latter search could threaten the required complexity of $O(d(x))$. Fortunately, for any search in the factorising permutation, we look for a vertex whose type is different from a specified type which can be *In*, *Out* or *InOut*, but not *None*. It follows that all the nodes visited by this search, but eventually the last one, are linked to x. Note that the pointers from the nodes to the factorising permutation are useful to grant access to the set of vertices represented by any node in constant time. As announced, the complexity of Routine `Find-Certificate` is $O(d(x))$ time.

5 Dynamic arc operations

In this section, we show how to handle arc modifications in $O(1)$ time. We only present how to handle arc deletion. Completing and adapting the argument of [16], we obtain, from the deletion algorithm, an insertion algorithm having the same complexity. Since the family of directed cographs is closed under complementation, the graph $G + xy$ is a directed cograph iff the graph $\overline{G} - xy$ is. Similarly, a certificate that $\overline{G} - xy$ is not a directed cograph, is a certificate for $G + xy$. The di-cotree of the complement of a directed cograph G is obtained from the di-cotree of G by changing parallel nodes into series nodes, and conversely, and reversing the order of the children of each order node. The data structure we use allows to make these changes in constant time for each node of the di-cotree. Since our arc deletion algorithm is based only on the di-cotree, it can be adapted to handle arc insertion within the same complexity.

5.1 Deleting an arc.

Two types of arc based modifications should be distinguished. The first one concerns the simultaneous removal of two symmetric arcs, say xy and yx. This modification can be compared to the deletion of an edge in an undirected cograph, see [16]. The proof of Theorem 4 in [16] is perfectly adaptable to the more general case of directed cographs (and to the case of deletion thanks to the discussion above). Let q_x (resp. q_y) be the child of p_{xy} containing x (resp. y). Recall that p_{xy} is defined as the lca of x and y in T.

Theorem 6. *The graph $G' = G - \{xy, yx\}$ is a directed cograph iff $|Q_x| = 1$ and $Q_y \setminus \{y\} \subseteq \overline{N}(y)$ or $|Q_y| = 1$ and $Q_x \setminus \{x\} \subseteq \overline{N}(x)$.*

Theorem 7 extends Theorem 6 so that any valid arc modification of a directed cograph can be characterised. Recall that M_{xy} is the minimum module of G containing x and y ($M_{xy} \subseteq P_{xy}$).

Theorem 7. *The graph $G' = G - xy$ is a directed cograph iff*

1. *p_{xy} is an order node, $M_{xy} = Q_x \cup Q_y$, and:*
   (a) *either $|Q_x| = 1$ and $Q_y \setminus \{y\} \subseteq \overline{N}(y)$,*
   (b) *or $|Q_y| = 1$ and $Q_x \setminus \{x\} \subseteq \overline{N}(x)$.*
2. *p_{xy} is a series node and:*
   (a) *either $|Q_x| = 1$ and $Q_y \setminus \{y\} \subseteq N^+(y) \setminus N^-(y)$,*
   (b) *or $|Q_y| = 1$ and $Q_x \setminus \{x\} \subseteq N^-(x) \setminus N^+(x)$.*

Proof of Theorem 7: We consider only the subgraph $G[M_{xy}]$ of G induced by M_{xy}. Indeed, since M_{xy} contains both x and y, then M_{xy} is also a module in $G' = (V, E')$, where $E' = E \setminus \{xy\}$. It follows that $G' = (V, E')$ is a directed cograph iff $G'[M_{xy}]$ is.
$\Longrightarrow$. Since the modules of G' containing both x and y are exactly the modules of G containing both x and y, then M_{xy} is also the minimum module of G' containing both x and y. It follows that the maximal strong module Q'_x of $G'[M_{xy}]$ containing x and the maximal strong module Q'_y containing y are distinct. We denote p'_{xy} for the root of the modular decomposition tree of $G'[M_{xy}]$.
If p_{xy} is an order node, $M_{xy} = \bigcup_{q_x \le h \le q_y} H$ where $\le$ denotes the order on the children of p_{xy}. After the deletion of the arc xy, x and y are not linked. Since Q'_x and Q'_y are distinct strong modules of $G'[M_{xy}]$, from Theorem 3, it follows that Q'_x and Q'_y are connected components of $G'[M_{xy}]$. Then, p'_{xy} is a parallel node and its children are exactly q'_x and q'_y, since M_{xy} is the minimal module of $G'[M_{xy}]$ containing x and y. We now show that, in fact, q_x and q_y are the only children of p_{xy}. Assume for contradiction that $\exists h \in \mathcal{C}(p_{xy})$ such that $q_x < h < q_y$. Let $u \in H$, both xu and uy belong to E, and so to E', which refutes that Q'_x and Q'_y are distinct connected components of $G'[M_{xy}]$. Thus, $M_{xy} = Q_x \cup Q_y$. Let $u \in Q_x \setminus \{x\}$. uy belongs to E, and so to E', then u belongs to the connected component Q'_y of y in $G'[M_{xy}]$. It follows that $u \in \overline{N}(x)$. Similarly, any $v \in Q_y \setminus \{y\}$ belongs to Q'_x and $v \in \overline{N}(y)$. Finally, assume there exist both $u \in Q_x \setminus \{x\}$ and $v \in Q_y \setminus \{y\}$. As we showed above, in $G'[M_{xy}]$, u is in the connected component Q'_y of y and v in the connected component Q'_x of x, which are distinct. But uv belongs to E, then uv belongs to E'. This refutes that Q'_x and Q'_y are distinct connected components of $G'[M_{xy}]$. Thus, if G' is a directed cograph, $Q_x \setminus \{x\} = \emptyset$ or $Q_y \setminus \{y\} = \emptyset$.
If p_{xy} is a series node, then $M_{xy} = Q_x \cup Q_y$. After the deletion of the arc xy, x and y are linked by the arc

yx. Since Q'_x and Q'_y are distinct strong modules of $G'[M_{xy}]$, from Theorem 3, it follows that Q'_x and Q'_y are order components of $G'[M_{xy}]$. Q'_y is the first, and Q'_x the last, in the order induced by p'_{xy}. Let $u \in Q_x \setminus \{x\}$, since both uy and yu belong to E, and so to E', then u is in the order component Q'_y of y. It follows that $u \in N^-(x)$. Similarly, any $v \in Q_y \setminus \{y\}$ belongs to Q'_x and then $v \in N^+(y)$. Finally, assume there exist both $u \in Q_x \setminus \{x\}$ and $v \in Q_y \setminus \{y\}$. As we showed above, in $G'[M_{xy}]$, u is in the order component Q'_y of y and v in the order component Q'_x of x, which are distinct. But uv and vu belong to E, then uv and vu belongs to E'. This refutes that Q'_x and Q'_y are distinct order components of $G'[M_{xy}]$. Then, if G' is a directed cograph, $Q_x \setminus \{x\} = \emptyset$ or $Q_y \setminus \{y\} = \emptyset$.
$\Longleftarrow$. We show that under conditions of Theorem 7, $G'[M_{xy}]$ is a directed cograph (see Fig. 5.1).
If p_{xy} is an *order* node and $|Q_x| = 1$, $\{x\}$ is an order component of $G[M_{xy}]$. Since $Q_y \setminus \{y\} \subseteq \overline{N}(y)$, after the deletion of xy, y is disconnected from x and $\{y\}$ becomes a connected component of $G'[M_{xy}]$. The other connected component is $\{x\} \cup (Q_y \setminus \{y\}) = Q'_x$ and $\{x\}$ is in order composition with $Q_y \setminus \{y\}$. Since $G'[Q'_x \setminus \{x\}] = G[Q_y \setminus \{y\}]$ is a directed cograph, then $G'[M_{xy}]$ is a directed cograph. The case where $|Q_y| = 1$ is similar.
If p_{xy} is a *series* node and $|Q_x| = 1$, $\{x\}$ is a co-connected component of $G[M_{xy}]$. Since $Q_y \setminus \{y\} \subseteq N^+(y) \setminus N^-(y)$, after the deletion of xy, x belongs to $N^+(y) \setminus N^-(y)$ and $\{y\}$ becomes an order component of $G'[M_{xy}]$. The other order component is $\{x\} \cup (Q_y \setminus \{y\}) = Q'_x$ and $\{x\}$ is in series composition with $Q_y \setminus \{y\}$. Since $G'[Q'_x \setminus \{x\}] = G[Q_y \setminus \{y\}]$ is a directed cograph, then $G'[M_{xy}]$ is a directed cograph. The case where $|Q_y| = 1$ is similar. □

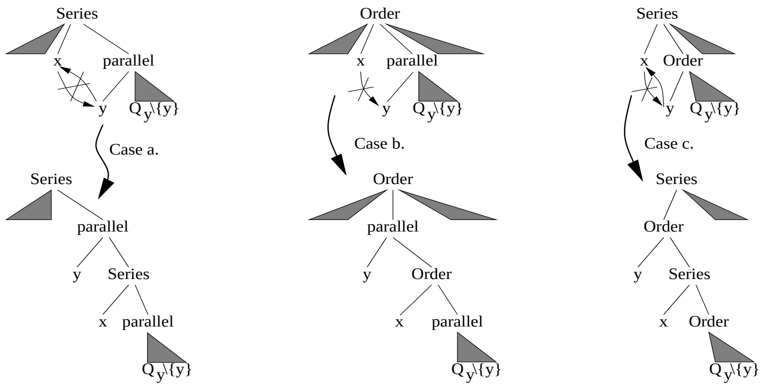


Fig. 16. Case `a.` illustrates the modification implied by the simultaneous removal of two symmetric arcs (see Theorem 6); cases `b.` and `c.` illustrate the removal of the arc xy described in Theorem 7. Depending on the number of siblings of y, the resulting di-cotrees may contain fewer nodes than depicted above.

It is straightforward from Theorem 6 and 7 that the deletion test can be done in $O(1)$. Indeed, x and y have to be either the child and the grand-child of p_{xy}, or two children of p_{xy}. Then, it suffices to check the label of p_{xy} and eventually of its child q_y which is the parent of y. At last, if p_{xy} is an order node, its children q_x and q_y have to be consecutive in the order defined by p_{xy}. If the deletion is possible, the modifications of the di-cotree are carried out in constant time, as depicted in Fig. 5.1.

5.2 Finding a certificate.

Assume the test of the xy deletion (or the deletion of symmetric arcs xy and yx) fails. As done for the vertex certificate, our algorithm returns a small subgraph containing one of the graphs of Fig. 2. Thanks to the factorising permutation, the vertices of this subgraph can be found in constant time. If an exact certificate is wished, it can be found in $O(Max(d(x), d(y)))$ time.

Lemma 6. *If $G' = G - xy$ is not a directed cograph, a set Z of at most 6 vertices can be found in $O(1)$ such that $G'[Z \cup \{x, y\}]$ contains one of the graphs of Fig. 2.*

The same set Z of vertices, described bellow, also provides a certificate in the case where the deletion of symmetric arcs xy and yx fails. In the following, we do not detail this case which is very similar to the case of deletion of a single arc xy.

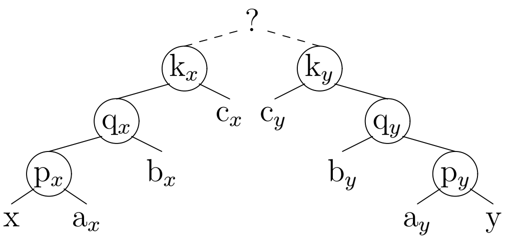
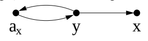


Fig. 17. A possible configuration for Z.

Let us describe how the set Z is defined. Let p_x (resp. p_y) be the parent of x (resp. y) in T. If $p_x \neq r$ (resp. $p_y \neq r$), let q_x (resp. q_y) be the parent of p_x (resp. p_y) in T. If $q_x \neq r$ (resp. $q_y \neq r$), let k_x (resp. k_y) be the parent of q_x (resp. q_y) in T. Let us define 6 vertices, namely a_x, b_x, c_x and a_y, b_y, c_y. Vertex a_x belongs to $P_x \setminus \{x\}$ and if p_x is an order node and if l_x is not the last order component, then choose for a_x a vertex in the order component immediately following l_x in the order defined by p_x. Vertices b_x and c_x belong respectively to $Q_x \setminus P_x$ and $K_x \setminus Q_x$ if these sets exist. The last 3 vertices a_y, b_y, c_y are similarly defined wrt. y. If possible, a_y should be picked in the order component immediately preceding y in the order defined by p_y. Note that, even if they exist, these vertices may not be all distinct (e.g. it may happen that $a_x = c_y$).

The table in Fig. 5.2 synthesis the case by case analysis to find out a forbidden graph of Fig. 2, in the case where $lca(x, y)$ is a series node. The case where $lca(x, y)$ is an order node is similar. Let us examine in details, as an example, the case where $lca(x, y)$ is a series node, where neither x nor y is a child of $lca(x, y)$, and where p_x is a parallel node (first line of the table). In this case, since p_x is parallel, there are no arcs between x and a_x. There are arcs in both directions between a_x and y, because $lca(a_x, y) = lca(x, y)$ is labelled series. And after the deletion of arc xy, x and y are linked by an arc from y to x. Thus, vertices a_x, x, y induced the following forbidden graph: 

.
Obviously, vertices $a_x, b_x, c_x, a_y, b_y, c_y$ can be found in constant time. If an exact certificate is wished, we need to find the lca of x and y. Since it may happen that this node is not among $p_x, q_x, k_x, p_y, q_y, k_y$ (cf. Fig 5.2), it may take $O(Max(d(x), d(y)))$ time to find it. Once $lca(x, y)$ has been found, examining the labels of p_x, q_x, k_x, p_y, q_y, k_y and $lca(x, y)$, it is possible to determine in constant time a subset $\widetilde{Z}$ of Z such that $G'[\widetilde{Z} \cup \{x, y\}]$ is a minimal forbidden subgraph.

References

1. A. Bretscher, D.G. Corneil, M. Habib, and C. Paul. A simple linear time lexbfs cograph recognition algorithm. In H. Bodlaender, editor, *29th International Workshop on Graph Theoretical Concepts in Computer Science (WG'03)*, number 2880 in Lecture Notes in Computer Science, pages 119–130, 2003.
2. C. Capelle and M. Habib. Graph decompositions and factorizing permutations. In *Fifth Israel Symposium on the Theory of Computing Systems (ISTCS'97)*, IEEE Computer Society, pages 132–143, 1997.
3. D.G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(1):163–174, 1981.
4. D.G. Corneil, Y. Perl, and L.K. Stewart. A linear time recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

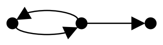
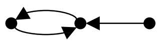
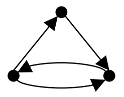
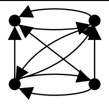
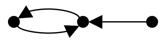
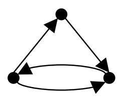
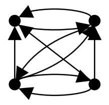
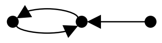
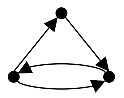
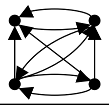
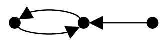
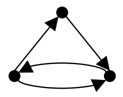
x, y, lca(x,y)	p_x, q_x, p_y, q_y	k_x, k_y	Forbidden subgraph
Neither x nor y is a child of lca(x,y)	One of p_x, q_x, p_y, q_y is parallel		 or 
	p_x or q_x is order and p_y or q_y is order		 or 
x is a child of lca(x,y) but y is neither a child nor a grand child of lca(x,y)	p_y or q_y is parallel		
	p_y is series and q_y is order		 or 
	p_y is order and q_y is series	k_y is parallel	
		k_y is order	 or 
y is a child of lca(x,y) but x is neither a child nor a grand child of lca(x,y)	Idem than for the previous case		
x is a child of lca(x,y) but y is neither a child nor a grand child of ppca(x,y)	p_y is parallel		
	p_y is order but y is not its first child		

Fig. 18. Case analysis to find out a minimal forbidden subgraph when $lca(x, y)$ is a series node.

5. C. Crespelle and C. Paul. Fully dynamic recognition algorithm and certificate for directed cographs. In J. Hromkovic, M. Nagl, and B. Westfechtel, editors, *30th International Workshop on Graph Theoretical Concepts in Computer Science (WG'04)*, number 3353 in Lecture Notes in Computer Science, pages 93–104, 2004.
6. C. Crespelle and C. Paul. Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. In Dieter Kratsch, editor, *31th International Workshop on Graph Theoretical Concepts in Computer Science (WG'05)*, number 3787 in Lecture Notes in Computer Science, pages 38–48, 2005.
7. A. Ehrenfeucht and G. Rozenberg. Primitivity is hereditary for 2-structures. *Theoretical Computer Science*, 70(3):343–359, 1990.
8. J.-L. Fouquet, V. Giakoumakis, and J.-M. Vanherpe. Bipartite graphs totally decomposable by canonical decomposition. *International Journal of Foundation of Computer Science*, 10(4):513–533, 1999.
9. V. Giakoumakis and J.-M. Vanherpe. Linear time recognition and optimizations for weak-bisplit graphs, bi-cographs and bipartite p_6-free graphs. *International Journal of Foundation of Computer Science*, 14(1):107–136, 2003.
10. P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM Journal on Computing*, 31(1):289–305, 2002.
11. L. Ibarra. Fully dynamic algorithms for chordal graphs. In *10th ACM-SIAM Annual Symposium on Discrete Algorithm (SODA'03)*, pages 923–924, 1999.
12. D. Kratsch, R.M. McConnell, K. Mehlhorn, and J.P. Spinrad. Certifying algorithm for recognition of interval graphs and permutation graphs. In *14th ACM-SIAM Annual Symposium on Discrete Algorithm (SODA'03)*, pages 153–167, 2003.
13. E.L. Lawler. Graphical algorithm and their complexity. *Mathematical center tracts*, 81:3–32, 1976.
14. R.H. Möhring and F. J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19:257–356, 1984.
15. J.H. Muller and J.P. Spinrad. Incremental modular decomposition algorithm. *Journal of the Association for Computing Machinery*, 36(1):1–19, 1989.
16. R. Shamir and R. Sharan. A fully dynamic algorithm for modular decomposition and representation of cographs. *Discrete Applied Mathematics*, 136(2-3):329–340, 2004.
17. J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11:298–313, 1982.