Algorithmica
DOI 10.1007/s00453-008-9273-0

Fully Dynamic Algorithm for Recognition and Modular Decomposition of Permutation Graphs

Christophe Crespelle · Christophe Paul

Received: 5 June 2006 / Accepted: 22 December 2008

Abstract This paper considers the problem of maintaining a compact representation ($O(n)$ space) of permutation graphs under vertex and edge modifications (insertion or deletion). That representation allows us to answer adjacency queries in $O(1)$ time. The approach is based on a fully dynamic modular decomposition algorithm for permutation graphs that works in $O(n)$ time per edge and vertex modification. We thereby obtain a fully dynamic algorithm for the recognition of permutation graphs.

Keywords Dynamic algorithms · Permutation graphs · Modular decomposition

1 Introduction

Finding efficient graph representations is a central question of algorithmic graph theory. How to store a graph to make its manipulation easier? Compact graph representations often rest on the combinatorial structures of the considered graphs (see for example [22]). Thereby interesting graph encodings can be found when restricted to special graph classes. This paper deals with dynamic graphs and considers the *dynamic recognition and representation problem* (see e.g. [21]), which, for a family $\mathcal{F}$ of graphs, aims to maintain a characteristic 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

This paper is a full version of the extended abstract appeared in [5].

C. Crespelle (✉)
Université de Montpellier 2, LIRMM, Montpellier, France
e-mail: crespell@lirmm.fr

C. Paul
CNRS, LIRMM, Montpellier, France
e-mail: paul@lirmm.fr

Published online: 15 January 2009

is of the following: inserting or deleting a vertex (along with the edges incident to it), inserting or deleting an edge. In this paper, we consider the dynamic recognition and representation problem for the class of permutation graphs.

1.1 Related Works

As witnessed by recent results, decomposition methods are very useful for the design of dynamic graph representations. For example, in the case of planar graphs, [10] devised a fully dynamic algorithm whose amortised cost per edge modification is $O(\sqrt{n})$. Their algorithm is based on a decomposition of the graph into sparse subgraphs. This decomposition is not related to the planar representations of the graph considered. On the other hand, [8] designed an incremental algorithm based on the *SPQR*-tree decomposition that allows to represent in $O(n)$ space all the embeddings of a planar graph, whose number can be exponential. They obtain an algorithm whose amortised cost per edge insertion is $O(\log n)$. Unfortunately, their algorithm cannot update the representation it uses under edge deletion.

The modular decomposition (or substitution decomposition) also revealed its efficiency for various graph classes. Roughly speaking the modular decomposition aims to recursively partition a graph into subgraphs induced by subsets of vertices (called modules). A module is a set of vertices that behaves uniformly with respect to the adjacency with the rest of the vertices (see Sect. 2.1 for a proper definition). Some graphs turn out to be indecomposable for the modular decomposition, they are the *prime* graphs. The decomposition of any graph can be represented by a tree (the modular decomposition tree). The inner nodes of that tree correspond to the sets of the partitions used to recursively decompose the graph. The leaves of the tree correspond to the vertices of the graph. Beside the tree, with each inner node is associated a *quotient* graph, also called *representative* graph. Each internal node is assigned a label depending on its representative graph: *parallel* if its representative graph is empty, *series* if its representative graph is complete, and *prime* if its representative graph is prime. Concerning the computation of the modular decomposition tree of an arbitrary graph, a purely incremental algorithm is presented in [19]. It runs in $O(n)$ time per vertex insertion. Unfortunately, it is based on a partial representation of the representative graphs (N-representation), compromising the possibility of any vertex deletion.

The totally decomposable graphs for the modular decomposition are called *cographs*. In [21], the dynamic recognition and representation problem has been considered for cographs. The algorithm is based on [4]'s incremental algorithm, where any modification (edge or vertex) is supported in $O(d)$ time, d being the number of edges involved in the modification. This result has recently been generalised in two different ways: in [6], directed cographs are considered, while [20] deals with P_4-sparse graphs. For these latter graph classes, the same complexity than for cographs has been obtained. For all of these three graph families, the modular decomposition tree is strongly constrained. In the case of cographs and directed cographs, none of its internal nodes is labelled prime, and in the case of P_4-sparse graphs, the prime representative graphs are very specific (a *spider*, see [2]) and the corresponding prime nodes have at most one non-leaf child.

In [13], a fully dynamic algorithm for the family of proper interval graphs has been designed. Each update is supported in $O(d + \log n)$ time, where d is the number of edges involved in the operation. Though the authors did not adopt the modular decomposition point of view, the representation they maintain for a proper interval graph is closely related to its modular decomposition tree. Moreover, as for the classes cited above, the modular decomposition tree of a proper interval graph is strongly constrained: its root r is a parallel node; every non-leaf children of r is either a prime node, or a series node with at most one non-leaf child, which has to be a prime node; the non-leaf children of any prime node are series nodes having only leaf children. Note that these conditions imply that the height of the tree is at most four. In [13], the authors maintain a representation of the prime graph associated with the node of each connected component of the graph, and, from their representation, it is straightforward to obtain the modular decomposition tree of the graph.

Chordal graphs should also be mentioned as a class whose dynamic maintenance has been considered. [14] devised a fully dynamic recognition algorithm for chordal graphs which handles edge operations in $O(n)$ time. In the same paper, the author obtains a fully dynamic recognition algorithm for split graphs that runs in constant time per edge modification.

1.2 Our Contribution

Permutation graphs are intersection graphs of segments between two parallel lines. Thereby any permutation graph can be represented by assigning, to each vertex, two integers describing the position of its segment extremities. Such a representation is called a *realiser* (see Sect. 2.2 for a formal definition). A permutation graph is a comparability graph the complement of which is also a comparability graph (see e.g. [22]), and this is a characterisation. It follows that any permutation graph G and its complement $\overline{G}$ can be transitively oriented. The best known permutation graph recognition algorithm [15, 16] makes use of this property. The links between transitive orientation and modules of graphs are well understood since Gallai's work [11]. Indeed, to compute a transitive orientation of a graph, a modular decomposition preliminary step is required. Let us briefly sketch the (non-dynamic) linear time permutation graph recognition algorithm of [15, 16]: 1) a preprocessing step first computes the modular decomposition of the input graph G; 2) then a transitive orientation algorithm is applied to compute a first linear ordering of the vertices, which defines a transitive orientation of G whenever G is comparability; 3) a second linear ordering is computed, corresponding to a transitive orientation of $\overline{G}$ if it exists. It turns out that the input graph G is a permutation graph if and only if these two linear orderings define a realiser of G, which can be tested in linear time. As the three steps described above also require linear time, the whole algorithm is linear. As far as we know, no linear time (non-dynamic) recognition algorithm for permutation graphs avoiding the computation of the modular decomposition has been designed.

It is clear from the discussion above that in context of non-dynamic graphs, the modular decomposition of permutation graphs is pretty well understood. However in the case of dynamic graphs very little is known on modular decomposition and permutation graphs. A natural question is for example, how the modular decomposition of a graph is affected by a vertex or an edge modification? Our fully dynamic

algorithm maintains an $O(n)$ space canonical representation of permutation graphs including both its modular decomposition tree and a realiser. It enables us to answer adjacency queries between any pair of vertices in $O(1)$ time. Both insertion and deletion operations of an edge or a vertex (along with the edges incident to it) are supported. The cost of maintenance is $O(n)$ time per operation. Note that a modification of the input graph may lead to $O(n)$ changes in the modular decomposition tree (see Fig. 2). Therefore our algorithm does not present any complexity extra cost in the maintenance of the modular decomposition tree. Furthermore, within the same complexity, we also maintain a realiser of the graph, as long as it remains a permutation graph. The $O(n)$ space structure we use even represents all realisers of the considered graph, whose number can be up to $\Omega(n!)$.

The family of permutation graphs is hereditary and closed under substitution composition (see Sect. 2.1). It follows that a graph is a permutation graph iff the prime representative graphs associated with the prime nodes of its modular decomposition tree are permutation graphs. Therefore, any modular decomposition tree is the tree of some permutation graph. In other words, the constraint to be a permutation graph does not apply on the modular decomposition tree itself but only on the representative graphs. Our algorithm is the first one that fully dynamically maintains the modular decomposition tree for a graph class for which the modular decomposition tree is not constrained. Let us recall that [19]'s algorithm, which applies to arbitrary graphs, only support vertex addition and does not maintain a whole representation allowing efficient adjacency test. As the one of [19], our algorithm performs in $O(n)$ time per operation, and supports insertion as well as deletion of vertices and edges. Moreover, while the approach of [19] is purely algorithmic, when a vertex x is inserted in a graph G, we give a structural characterisation of the modular decomposition of the augmented graph $G + x$ which is general to arbitrary graphs (Theorem 2). In the case of a permutation graph G, we also give necessary and sufficient conditions under which the graph $G + x$ remains a permutation graph (Theorem 3).

2 Preliminaries

The paper deals with undirected loop-less simple graphs. If x is a vertex of a graph $G = (V, E)$, $N(x)$ denotes the neighbourhood of x and $\overline{N}(x)$ its non-neighbourhood. The graph $\overline{G}$ is the complement graph of G. If S is a subset of vertices of a graph G, then $G[S]$ is the subgraph of G induced by S. Two sets of vertices S and S' are *adjacent* iff any vertex of S is adjacent to any vertex of S', and *non-adjacent* iff any vertex of S is not adjacent to any vertex of S'. Two sets S and S' *overlap* iff $S \cap S' \neq \emptyset$ and $S \setminus S' \neq \emptyset$ and $S' \setminus S \neq \emptyset$ (we note $S \perp S'$). Two vertices x and y are *twins* iff $N(x) = N(y)$ or $N(x) \cup \{x\} = N(y) \cup \{y\}$.

2.1 Modular Decomposition

Modular decomposition theory has been independently rediscovered in many fields. Gallai [11] may be the first to deal with this concept in the context of graph theory. For a complete survey on the topic, refer to [18]. Let us give the standard definitions and basic results we will need.

A subset $S \subsetneq V$ of vertices of a graph G is *uniform* with respect to vertex $x \in V \setminus S$ iff $S \subseteq N(x)$ or $S \subseteq \overline{N}(x)$ (otherwise S is *mixed*). A *module* of G is a subset of vertices $M \subseteq V$ which is uniform with respect to any vertex $x \in V \setminus M$. It follows from definition that the whole vertex set V and the singleton sets $\{x\}$, for any $x \in V$, are modules of G. These modules are called the *trivial modules*. A graph is *prime* if all its modules are trivial. A *submodule* of a module M of G is a subset $S \subseteq V$ that is a module of $G[M]$. Note that a submodule is necessarily a module of the graph G itself. Let us state a very simple observation.

Observation 1 *Let M be a module of G and S a subset of vertices. Then $M \cap S$ is a module of the induced subgraph $G[S]$.*

A module M is *strong* if it does not overlap any module M'. Connected components of G and connected components of $\overline{G}$, also called *co-connected components* of G, are examples of strong modules. The strong modules of a graph can be organised into an inclusion tree, called the *modular decomposition tree* T_G. There is a one-to-one mapping between nodes of T_G and strong modules of G. Thereby the root of T_G represents the whole vertex set and the leaves are mapped to the singleton trivial modules of G. For an internal node p of T_G, the corresponding strong module is the vertex set $V(p) = P$ resulting from the union of modules mapped to its children. In the following, T_p denotes the subtree of T_G rooted at node p (i.e. the modular decomposition tree $T_{G[P]}$). The set of children of a node p in T_G will be denoted by $\mathcal{C}(p)$. In all the paper, we use lowercase letters, such as p, q, to denote nodes of the modular decomposition tree, and we use capital letters, P, Q, to denote the corresponding strong modules. We may also refer to these strong modules as $V(p) = P$ and $V(q) = Q$.

A *maximal strong module* of a graph $G = (V, E)$ is a strong module of G maximal with regard to inclusion and distinct from V. If G (resp. $\overline{G}$) is disconnected, the connected components (resp. co-connected components) of G are precisely its maximal strong modules. If both G and $\overline{G}$ are connected, the maximal strong modules of G are the maximal modules of G distinct from V (in other words, the maximal modules of G distinct from V do not overlap).

Thanks to the well-known modular decomposition theorem (see [18] for references), any non-leaf node p of the modular decomposition tree is labelled as follows: *parallel* if $G[P]$ is not connected; *series* if $\overline{G}[P]$ is not connected; and *prime* otherwise (the three cases are disjoint). The label of node p is denoted by $label(p)$. The series and parallel nodes (and their corresponding strong modules) are also called *degenerate* nodes. It should be noticed that any module is either a strong module or the union of maximal strong submodules of a degenerate strong module.

A partition $\mathcal{P} = \{M_1, \ldots M_k\}$ of the vertex set is a *congruence partition* iff any part M_i $(1 \leqslant i \leqslant k)$ is a module. Congruence partitions play an important role in the theory of modular decomposition [18] as well as in most of decomposition algorithms (see [9] for example). Notice that it follows from the definition of a module that any pair of distinct parts M_i and M_j of a congruence partition is either adjacent or non-adjacent. Thereby it is possible to define the quotient graph $G/\mathcal{P}$ of a congruence

partition as the graph whose vertices are the elements of $\mathcal{P}$ and where two vertices are adjacent iff their corresponding modules are adjacent.[1]

An important property of congruence partitions and quotient graphs is the following.

Lemma 1 *Let $\mathcal{P}$ be a congruence partition of G. Then $\mathcal{X} \subseteq \mathcal{P}$ is a non-trivial strong module of $G/\mathcal{P}$ iff $X = \bigcup_{M \in \mathcal{X}} M$ is a non-trivial strong module of G.*

Proof Actually, it is proved in [18], that $\mathcal{X} \subseteq \mathcal{P}$ is a module of $G/\mathcal{P}$ iff $X = \bigcup_{M \in \mathcal{X}} M$ is a module of G. So let $\mathcal{X}$ be a non-trivial module of $G/\mathcal{P}$.

$\Leftarrow$ Assume $\mathcal{X}$ is not a strong module of $G/\mathcal{P}$. There exists a module $\mathcal{Y} \subseteq \mathcal{P}$ of $G/\mathcal{P}$ such that $\mathcal{X} \perp \mathcal{Y}$. From above, $Y = \bigcup_{M \in \mathcal{Y}} M$ is a module of G. Thereby $Y \perp X$ and X is not a strong module of G.

$\Rightarrow$ Assume X is not a strong module of G. Thereby it is the union of maximal strong submodules of a degenerated strong module Z of G. For $\mathcal{P}$ to be a congruence partition, there must exist $\mathcal{Y} \subseteq \mathcal{P}$ such that $Y = \bigcup_{M \in \mathcal{Y}} M = Z \setminus X$. As Z is degenerate, Y is a module of G and $\mathcal{Y}$ is a module of $G/\mathcal{P}$. Now by assumption $\mathcal{X}$ is non-trivial. Thereby there exists $\mathcal{S} \subsetneq \mathcal{X}$ such that $S = \bigcup_{M \in \mathcal{S}} M$ is the union of some maximal strong submodules of Z. Again as Z is degenerate, $Y \cup S$ is a module of G, in other words $\mathcal{X}$ and $\mathcal{Y} \cup \mathcal{S}$ are also modules of $G/\mathcal{P}$. It follows that $\mathcal{X}$ is not a strong module of $G/\mathcal{P}$ as by definition of $\mathcal{S}$, $\mathcal{X} \perp \mathcal{Y} \cup \mathcal{S}$. □

The set $\mathcal{MSM}(G)$ of maximal strong modules of G is a congruence partition and by the way define a quotient graph. Similarly a quotient graph, denoted G_p and called *representative graph* can be associated to each node p of the modular decomposition tree. The children $p_1 \dots p_k$ of p are mapped to the maximal strong modules of $G[P]$. The representative graph G_p is therefore the quotient graph $G[P]/\mathcal{MSM}(G[P])$. G_p is a clique if p is a series node and a stable if p is a parallel node. It follows that if together with each prime node p of the modular decomposition tree, its representative graph G_p is stored, then adjacency queries between any pair of vertices x, y can be answered by a search in T_G and in the representative graphs. Note that the size of the modular decomposition tree T_G is $O(n)$. If the representative graphs are added, it yields an $O(n + r)$ space representation with r being the sum of the size of the encoding of representative graphs. For arbitrary graph $r = n + m$, but for special graph families, like permutation graphs, it can be $r = n$.

If $\mathcal{F}$ is a family of disjoint modules but does not necessarily define a partition of the vertex set, then we abusively use the notation $G/\mathcal{F}$ to design $G/\mathcal{P}$ with $\mathcal{P} = \mathcal{F} \cup \{\{x\} \mid x \in V \setminus \bigcup_{M \in \mathcal{F}} M\}$. The inverse of the quotient operation is called the *substitution composition*. Given a vertex x of a graph $G = (V_G, E_G)$ and a graph $H = (V_H, E_H)$, substituting H for x in G results in the graph $G_{x \to H} = ((V_G \setminus \{x\}) \cup V_H, (E_G \setminus \{xy \mid y \in V_G\}) \cup E_H \cup \{zy \mid z \in V_H, xy \in E\})$.

[1] The quotient graph can also be seen as the induced subgraph $G[S]$ with $S \subseteq V$ and $\forall i, |M_i \cap S| = 1$. The vertex of $M_i \cap S$ is called the *representative vertex* of M_i.

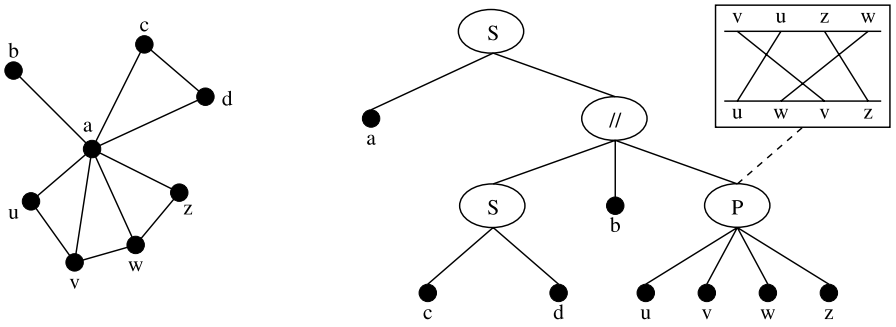

Fig. 1 A permutation graph and its full modular representation

2.2 Permutation Graphs

If π is a linear order on the vertices of a graph G, $\pi(x)$ denotes the rank of vertex x in π while $\pi^{-1}(i)$ is the vertex at rank i. Permutation graphs (see [12] for a detailed introduction) are those graphs for which there exists a pair (π_1, π_2) of linear orders on the vertex set such that x and y are adjacent iff $(\pi_1(x) - \pi_1(y))(\pi_2(x) - \pi_2(y)) < 0$. For a permutation graph G, such a pair $R = (\pi_1, \pi_2)$ is a *realiser* of G. Let us denote $\overline{\pi}$ the reverse order of π. It should be noticed that if $R = (\pi_1, \pi_2)$ is a realiser of G, then (π_2, π_1), $(\overline{\pi}_1, \overline{\pi_2})$ and $(\overline{\pi}_2, \overline{\pi_1})$ are also realisers of G. In the following, these four realisers are considered as the same. It follows from the definition that the family of permutation graphs is hereditary. The restriction of R to a subset $S \subseteq V$ of vertices is denoted $R[S]$ and is a realiser of the induced subgraph $G[S]$.

A graph is a comparability graph iff its edges can be transitively oriented. Permutation graphs are precisely comparability graphs whose complement is also a comparability graph. It follows from a result of [11] on comparability graphs that the permutation graph family is closed under the substitution composition. Actually if G and H are permutation graphs having realisers $R_G = (\pi_1, \pi_2)$ and $R_H = (\tau_1, \tau_2)$, then the permutation graph $G_{x \to H}$ has a realiser R where τ_1 has been substituted for x in π_1 and τ_2 for x in π_2. Moreover from [11, 17], a graph is a permutation graph iff the representative graphs of the prime nodes of its modular decomposition tree are permutation graphs. And it is known [17] that any prime permutation graph has a unique realiser. It follows that associating the modular decomposition tree T_G with the realiser of each of its prime nodes provides an $O(n)$ space canonical representation of a permutation graph G. If p is a node of the modular decomposition tree T_G, then R_p denotes the realiser of its representative graph G_p.

Definition 1 The *full modular representation* of a permutation graph G is the pair $MD(G) = (T_G, \mathcal{R}_G)$ where T_G is the modular decomposition tree of G and $\mathcal{R}_G$ is the set of realisers R_p with p a prime node of T_G (see example in Fig. 1).

The realisers of degenerate nodes do not need to be stored as they are trivial: (π, π) for parallel nodes and $(\pi, \overline{\pi})$ for series nodes, where π is any linear order on their children. Note that a realiser of the whole graph G can be retrieved in $O(n)$

time by composing the realisers of the nodes of the full modular representation. As our dynamic algorithm works in $O(n)$ time per operation, a realiser of G can be maintained without any extra cost. That guarantees the possibility of answering at any time adjacency queries in $O(1)$ time.

It has recently been remarked [1, 3] that the strong modules of a permutation graph can be retrieved from its realiser. An *interval* of a linear order π on V is a set of consecutive elements of V in π. Let $(a, b) \in V^2$, the interval $\{y \in V \mid a \leq y \leq b\}$ of π is denoted $[a, b]$. Given a pair (π_1, π_2) of linear orders, a *common interval* [23] is a set I that is an interval of π_1 and of π_2. In [23], an $O(n + K)$ algorithm computing all common intervals of a pair of linear orders has been proposed, K being the number of common intervals. A common interval is *strong* if it does not overlap any other common interval. Clearly common intervals of a realiser $R = (\pi_1, \pi_2)$ of a permutation graph G are modules of G. The converse is false in general, but true for strong modules.

Proposition 1 ([7]) *The strong modules of a permutation graph $G = (V, E)$ are exactly the strong common intervals of any of its realisers.*

In [1, 3], the algorithm of [23] has been revisited so that the inclusion tree of the strong common intervals can be computed in $O(n)$ time if the pair of linear orders is given. By Proposition above, that inclusion tree is the modular decomposition tree of the corresponding permutation graph.

It is worth to notice that the full modular representation is a representation of all the realisers of a permutation graph G. Indeed, any realiser can be obtained from $MD(G)$ by choosing a realiser of the representative graph of each node of the tree and composing them together. For that purpose, all the four different realisers of a prime node, that we usually consider as a single one, since they are equivalent, are useful. The realisers (π_1, π_2) of a degenerate node p are obtained by choosing an arbitrary order on its children for π_1 and the same order for π_2 if p is parallel, the reverse order if p is series. It follows that a degenerate node with k children admits $k!$ different realisers. It is easy to see that composing together the realisers we chose for all the nodes of the tree results in a realiser of G. Conversely, if R is a realiser of G, from Proposition 1, the maximal strong modules of G are strong common intervals of R. Contracting, in R, these strong common intervals into single vertices gives a realiser of the representative graph of the root. Recursively applying the process on $R[M]$, where $M \in \mathcal{MSM}(G)$, we obtain realisers of the representative graphs of the children of the root, and finally of all nodes of the tree. By construction, the substitution composition of all these realisers results in R.

2.3 Dynamic Arc Operations

Unfortunately an edge modification may imply $O(n)$ changes in the modular decomposition tree (Fig. 2). As we propose an $O(n)$ time algorithm for the vertex insertion and for the vertex deletion operations, inserting or deleting an edge e incident to vertex x will be handled by first removing x and then inserting x again with the updated neighbourhood.

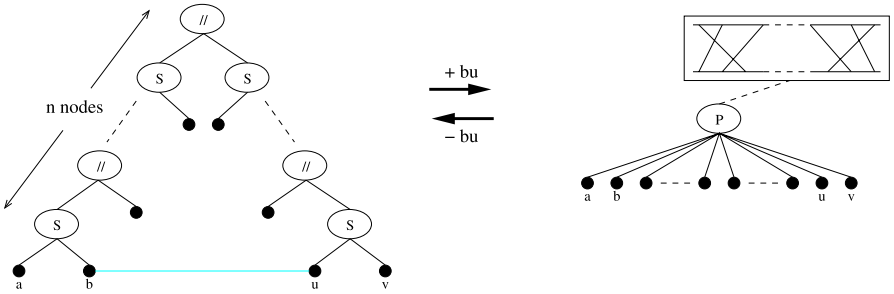

Fig. 2 An edge insertion/deletion implying $O(n)$ changes in the full modular representation. When the edge bu is inserted in the cograph whose decomposition tree is given on the left, it becomes prime and remains a permutation graph. The number of nodes in its modular decomposition tree switches from $2n-1$ to 1. Conversely, when the edge bu is deleted in the prime permutation graph represented on the right, it becomes a cograph

3 Vertex Deletion

Let $G' = G - x$ be the graph resulting from the deletion of a vertex x in the permutation graph G. The family of permutation graphs is *hereditary*, that is every induced subgraph of a graph in the family belongs to the family. Consequently, removing x reduces to update the full modular representation of G' from the one of G. We shall distinguish the case where the parent node p of x in T_G (if it exists) is a prime node from the case where p is a degenerate node.

Degenerate Case An algorithm was proposed in [21] to handle vertex deletion in dynamic cographs. Under a slight modification (case 4(b) below), it also applies in the case of permutation graphs when a vertex x child of a degenerate node is removed. Let us generalise [21]'s algorithm.

1. If T_G contains leaf l_x only, then $T_{G'}$ is empty.
2. If p has at least three children, then $T_{G'}$ is obtained by deleting from T_G the leaf l_x.
3. If p has only two children, namely l_x and l, where l is a leaf of T_G, then $T_{G'}$ is obtained from T_G by deleting l_x and replacing p with l.
4. If p has only two children, namely l_x and q_1, where q_1 is an internal node of T_G:
   (a) If p is the root of T_G, then $T_{G'}$ is the subtree of T_G rooted at q_1.
   (b) Assume p is not the root of T_G. Let q_2 be the parent node of p.
   - If q_1 and q_2 are both series nodes or both parallel nodes, then $T_{G'}$ is obtained by deleting from T_G the leaf l_x and the nodes p and q_2, and connecting the children of q_2 to q_1.
   - Otherwise, $T_{G'}$ is obtained by deleting from T_G the leaf l_x and replacing p with q_1.

Replacing a node p with a node q in the modular decomposition tree only consists in two operations: p's parent becomes q's parent and p is deleted. Node q saves all

of its original children. It can clearly be done in $O(1)$ time. Notice that any case but case 4(b), can be handled in constant time. The number of operations in case 4(b) is clearly bounded by $O(n)$.

Finally to obtain $MD(G')$, it remains to update (if needed) the set $\mathcal{R}_G$ of realisers of $MD(G)$. The only case to manage occurs when p's parent is a prime node. Then q replaces p in the representative realiser R_p. To prove the correctness of the algorithm, it is easy to check that the adjacency between any pair of vertices of $V \setminus \{x\}$ is preserved by the operations.

Prime Case If p is a prime node, then deleting x may generate modules in the representative graph of p. It follows that the algorithm has to deal with only two cases:

1. If $G_p - x$ is a prime graph, then $T_{G'}$ is simply obtained by deleting from T_G the leaf l_x. Similarly, removing x from the realiser R_p gives the representative realiser of the updated prime node.
2. Otherwise, let $MD(G_p - x)$ be full modular representation of $G_p - x$. $MD(G')$ is obtained by first replacing in T_G the node p with the root of $T_{G_p - x}$ and then replacing any leaf of $T_{G_p - x}$ with the corresponding child of p. $\mathcal{R}_{G'}$ is the union of $\mathcal{R}_G \setminus \{R_p\}$ and of the set of realisers of $MD(G_p - x)$.

As for the degenerated case, it can easily be argued that adjacencies between non-deleted vertices are preserved by these operations. Thereby we reduce the problem to the computation of $MD(G')$, where $G' = G - x$ and G is a prime graph. As already mentioned, using algorithms of [1, 3], the modular decomposition tree of a permutation graph can be computed in $O(n)$ time if a realiser is given. Moreover removing x from the realiser of G provides a realiser of G'. Therefore algorithms of [1, 3] can be used for our purposes. Notice that in $O(n)$ time, the realisers of all the prime nodes can be recursively extracted by a bottom-up process along the tree. We therefore obtain the following theorem.

Theorem 1 *Updating the full modular representation of a permutation graph under vertex deletion costs $O(n)$ time.*

Let us continue the study of the vertex deletion operation. The following property indicates that a somehow simpler algorithm than ones of [1, 3] can be designed as the modules of $G - x$, if G is prime, have a very restricted structure. Moreover this property will be useful for latter proofs.

Lemma 2 *Let $G = (V, E)$ be a prime permutation graph and x be a vertex. The non trivial strong modules of $G' = G - x$ can be partitioned in two families (possibly empty) totally ordered by inclusion.*

Proof Let us denote $R = (\pi_1, \pi_2)$ a realiser of G and R' the realiser of G' obtained by deleting x in R. From Proposition 1, it is equivalent to show that the strong common intervals of R' can be divided in two families (possibly empty) totally ordered by inclusion. Still from Proposition 1, any common interval of R is non-trivial. Let I

be a non-trivial common interval of R'. Since I is not a common interval of R, x is between two vertices of I in at least one of the two orders π_1, π_2 of R. In other words, $I \cup \{x\}$ is an interval of π_1 (or π_2) and x is not a bound of this interval. We say that I surrounds x in π_1 (or π_2). Since the strong common intervals of a realiser do not overlap, those of R' which surround x in π_1 of R (resp. in π_2) are totally ordered by inclusion. □

As there are at most two non-trivial maximal strong modules, the root of $T_{G'}$ has at most two non-leaf children, and each internal node of $T_{G'}$ have at most one non-leaf child. Moreover, it can be shown that any degenerated node of $T_{G'}$ has at most two leaf children. Therefore the number of modules (not necessarily strong) of G' is $O(n)$, and there are also $O(n)$ common intervals of the realiser of G'.

4 Vertex Insertion

Given a graph $G = (V, E)$, a vertex $x \notin V$ and a subset $N(x) \subseteq V$, we define $G' = G + x$ as the graph on vertex set $V \cup \{x\}$ with edge set $E \cup \{xy \mid y \in N(x)\}$). In order to maintain the full modular representation of a permutation graph under vertex insertion, we will: 1) update the modular decomposition tree T_G to get $T_{G'}$; 2) test whether $G' = G + x$ is a permutation graph; and 3) in the positive, compute the set of realisers $\mathcal{R}_{G'}$ from $\mathcal{R}_G$ and T_G. The algorithm we propose has an $O(n)$ complexity per vertex insertion. The presentation of this section follows these three steps.

4.1 Modular Decomposition Tree of $G + x$

[19] proposed an incremental algorithm that updates the modular decomposition tree under the insertion of a vertex x. While the approach of [19] is algorithmic, in this subsection, we give a mathematical description of the modular decomposition tree of the augmented graph, independently from its computational aspects. The results we obtain in this section applies to arbitrary graphs and are not restricted to permutation graphs. The main interest of our approach is to separate the problem of maintaining the modular decomposition tree from the problem of maintaining the representation chosen for the representative graphs of the prime nodes. To perform the maintain of the tree in $O(n)$ time under vertex insertion, it is sufficient that this representation allows to determine whether x has a twin in the representative graph of a prime node p in $O(|\mathcal{C}(p)|)$ time. This is true for the N-representation of [19]. Nevertheless, in [19], as the structure associated with a prime node (N-representation) is only a partial representation, it does not allow to maintain the tree under vertex deletion. Indeed, not all edges of a prime graph can be retrieved from its N-representation. In our case, the fact that we consider permutation graphs family will give us a complete representation of the representative graphs that allows both to determine whether x has a twin in the quotient of a prime node p in $O(|\mathcal{C}(p)|)$ time and to maintain the tree under vertex deletion (as we showed in the previous section). Whatever may be the representation chosen for the quotients of the prime nodes, the results we give on the structure of the modular decomposition tree of the augmented graph remain valid and entirely determine this tree.

Each node p of T_G is assigned a type with respect to x : *linked* (resp. *notlinked*) if $P = V(p)$ is uniform with respect to x and $P \subseteq N(x)$ (resp. $P \subseteq \overline{N}(x)$), and *mixed* otherwise. The types we assign to nodes correspond to the marks 1, -1 and 0 given in the algorithm of [19]. $\mathcal{C}_l(p)$ (resp. $\mathcal{C}_{nl}(p)$) stands for the set of children of p which are typed *linked* (resp. *notlinked*) and $\mathcal{C}_m(p)$ for the set of children of p which are typed *mixed*. For $t \in \{m, l, nl\}$, we denote $F_t(p) = \bigcup_{f \in \mathcal{C}_t(p)} V(f)$.

Insertion Node We show that inserting x in G reduces to update a certain subtree of the modular decomposition tree. Let us first identify the root of that subtree.

Definition 2 A node p of T_G is a *proper* node iff either p is uniform with respect to x, or p is a mixed node with a unique mixed child f such that $F \cup \{x\}$ is a module of $G'[P \cup \{x\}]$, with $F = V(f)$ and $P = V(p)$. Otherwise p is a *non-proper* node.

Lemma 3 *Let $G = (V, E)$ be a graph and x a vertex to be inserted in G. If all the nodes of T_G are proper, then V is uniform with respect to x.*

Proof We prove it by contrapositive. If V is mixed, then the root of T_G is mixed. Any mixed node of T_G always enjoys a mixed descendant p having only uniform children. By Definition 2, a node like p is non-proper. □

As a consequence of Lemma 3, in the case where there are no mixed nodes in T, x is either a universal vertex or an isolated vertex. Therefore the modular decomposition tree is easy to update in constant time. That case will not be considered anymore in the following. From now on, we assume that there is at least one non-proper node in T.

Observation 2 *The least common ancestor (lca for short) q of non-proper nodes of T is a non-proper node.*

Proof Let p_1 and p_2 be two non-proper nodes. If $lca(p_1, p_2) \in \{p_1, p_2\}$, then the property holds. Otherwise, since any ancestor of a mixed node is mixed, $lca(p_1, p_2)$ has at least two mixed children. Thereby $lca(p_1, p_2)$ is non-proper. □

Lemma 4 *If q is the least common ancestor of non-proper nodes of T, then $Q' = Q \cup \{x\}$ is a strong module of $G' = G + x$.*

Proof *Q' is a module of G'.* We state by induction that for any ancestor p_1 of q, Q' is a module of $G[P_1']$, with $P_1' = P_1 \cup \{x\}$. Indeed, Q' is a module of $G'[Q']$. Let p_1 be an ancestor of q such that Q' is a module of $G'[P_1']$. If $p_1 \neq r$ then let p_2 be the parent of p_1. Since q is mixed and p_2 is an ancestor of q, p_2 is a mixed proper node (by definition of q). It follows that p_1 is the unique mixed child of p_2 and P_1' is a module of $G'[P_2']$. Since Q' is a module of $G'[P_1']$ and P_1' is a module of $G'[P_2']$, thus, by definition of a module, Q' is a module of $G'[P_2']$. This ends the induction and shows that Q' is a module of G'.

Q' does not overlap any other module M' of G'. Let M' be a module of G'. Assume M' overlaps Q'. If $x \notin M'$, then M' is a module of G. Since Q is a strong module of G, M' and Q do not overlap. Thereby, since M' overlaps Q', then $Q \subsetneq M'$. Q is not uniform with respect to x, so neither M' is. This is a contradiction with the fact that M' is a module of G' which does not contain x. If $x \in M'$, then $M = M' \setminus \{x\}$ is a module of G (see Observation 1). Since Q is a strong module of G, it does not overlap M. Assume $M \cap Q = \varnothing$. Since M' overlaps Q', $Q' \setminus M' = Q$ is a module of G'. Which is a contradiction since $x \notin Q$ and Q is not uniform with respect to x. It follows that $M \cap Q \neq \varnothing$, and since Q does not overlap M, then $Q \subseteq M$ or $M \subseteq Q$. In both cases, Q' does not overlap M', which is the final contradiction. □

In the following, the node of $MD(G')$ corresponding to the strong module Q' of G' is denoted by q'.

Lemma 5 *Let $G' = G + x$ and let q be the least common ancestor of non-proper nodes of T_G. $T_{G'}$ is obtained from T_G by replacing the subtree T_q of T_G with $T_{G'[Q']} = T_{G[Q]+x}$.*

Proof Since Q is a strong module of G and since, from Lemma 4, $Q' = Q \cup \{x\}$ is a strong module of $G' = G + x$, then $G/\{Q\}$ and $G'/\{Q'\}$ are well defined. These two quotients are equal since choosing the representative vertex of Q' in Q leads to the same set of representative vertices for the two quotients. Note that $T_{G/\{Q\}}$ is obtained from T_G by replacing node q with a leaf corresponding to its representative vertex. That is $T_{G/\{Q\}}$ is exactly the complement part of T_q in T_G. Similarly, $T_{G'/\{Q'\}}$ is the complement part of $T_{q'}$ in $T_{G'}$. Since $G/\{Q\} = G'/\{Q'\}$, it follows that $T_{G'}$ is obtained from T_G by replacing the subtree T_q with $T_{G'[Q']}$. □

Definition 3 The *insertion node* q is the least common ancestor of non-proper nodes of T.

In the following, q will always denote the insertion node. From Lemma 5, we conclude that updating T_G under the insertion of x reduces to insert x in $T_{G[Q]}$.

Modular Decomposition Tree of $G'[Q']$ Let us distinguish the different situations for the insertion node q.

Definition 4 The insertion node q of T_G is *cut* iff q is either

1. a degenerate node with no mixed child but with uniform children of both types (i.e. *linked* and *notlinked*), or
2. q is a prime node with no mixed child but one being a twin of x in G_q.

Otherwise, q is *uncut*. Namely, q is a degenerate node with at least one mixed child, or q is a prime node with no child being a twin of x in G_q.

The case where the insertion node is a cut degenerate node is similar to the case, considered by [4], of maintaining the modular decomposition tree of a cograph under

vertex insertion. If q is a series (resp. parallel) node, the root q' of $T_{q'}$ is a series (resp. parallel) node. The children of q' are those children of q typed *linked* (resp. *notlinked*) and a new parallel (resp. series) node q'_1. The children of q'_1 are $\{x\}$ and the remaining children of q, i.e. those typed *notlinked* (resp. *linked*).

The case where the insertion node is a cut prime node is quite easy to deal with. In the children of q, the twin f of x is replaced by a new degenerate node q_1 (i.e. q_1 takes the place of f in the realiser of q). The label of q_1 is series if f is typed linked, and parallel if f is typed *notlinked*. $\{x\}$ and f are made children of q_1.

Let us now consider the case where the insertion node q is uncut.

Notation 1 Let us define the vertex set Q_s as the set Q if q is a prime node and as the set $F_m(q) \cup F_{nl}(q)$ (resp. $F_m(q) \cup F_l(q)$) if q a series node (resp. parallel node). As usual, Q'_s will denote $Q_s \cup \{x\}$.

Determining the modular decomposition of $G'[Q'_s]$ (see Theorem 2 below) is crucial to obtain the modular decomposition tree of $G'[Q']$. From now on, we denote $\mathcal{MUM}(G)$ the set of maximal uniform (with respect to x) modules of a graph G. It should be noticed that $\mathcal{MUM}(G)$ is by definition a congruence partition of G. The two following lemmas are useful for the proof of Theorem 2.

Lemma 6 *Let M be a module of G and M' a module of $G' = G + x$, such that $x \in M'$ and $M \cap M' \neq \emptyset$. Then $M \cup M'$ is a module of G'.*

Proof Let $G = (V, E)$ and $G' = G + x = (V', E')$. Let $y \in V' \setminus (M \cup M')$ and $z \in M \cap M'$. Since M is a module of G, $\{y, z\} \in E'$ iff $\forall u \in M, \{y, u\} \in E'$. And since M' is a module of G', $\{y, z\} \in E'$ iff $\forall v \in M, \{y, v\} \in E'$. □

Lemma 7 *If $G' = G + x$ is connected and co-connected and if $\{x\}$ is a maximal strong module of G', then the set of maximal strong modules of G' different from $\{x\}$ is exactly the set of maximal uniform (with respect to x) module of G. $\mathcal{MSM}(G') \setminus \{x\} = \mathcal{MUM}(G)$.*

Proof Since G' is connected and co-connected, its maximal strong modules are its maximal modules. Since x is one of them, the others do not contain x. Thus, the maximal modules of G' different from $\{x\}$ are the maximal elements (for inclusion) of the set of modules of G' which do not contain x. Let us show that these maximal elements are exactly $\mathcal{MUM}(G)$. If M is a uniform module of G, M is a module of G'. Conversely, if M' is a module of G' which does not contain x, M' is a uniform module of G. The set of uniform modules of G is exactly the set of modules of G' which do not contain x, and then the maximal elements of these two sets are the same. □

Theorem 2 *Let x be a vertex to be inserted in a graph G. If the insertion node q of the modular decomposition tree T of G is uncut, then $G'[Q'_s]$ is connected and co-connected and $\mathcal{MSM}(G'[Q'_s]) = \mathcal{MUM}(G[Q_s]) \cup \{\{x\}\}$.*

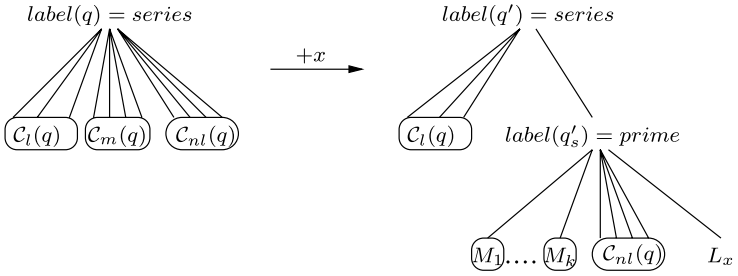

Fig. 3 Updating the modular decomposition tree when the insertion node is a series node. The modules $M_1 \dots M_k$ are the maximal uniform modules of $G[Q_s]$

Proof Thanks to Lemma 7, we just have to show that $G'[Q'_s]$ is connected and co-connected, and $\{x\}$ is a maximal strong module of $G'[Q'_s]$.

If q is a prime node, by definition, $Q'_s = Q'$. $G[Q]$ is connected and co-connected, and since Q is mixed, then $G'[Q']$ is connected and co-connected. If q is a series node, by definition of $Q_s = \bigcup_{f \in \mathcal{C}_{nl}(q) \cup \mathcal{C}_m(q)} V(f)$, $G[Q_s]$ is connected. In addition, since q is uncut, then $\mathcal{C}_m(q) \neq \varnothing$. It follows that there exists $y \in Q_s$ such that $\{x, y\}$ is an edge. Thus $G'[Q'_s]$ is connected. Since q is a series node, any child f of q is parallel or prime. Therefore, $G[F]$ is co-connected. Moreover, for each $f \in \mathcal{C}_{nl}(q) \cup \mathcal{C}_m(q)$, by definition, there exists $y \in F$ such that $\{x, y\}$ is not an edge. It follows that $G'[Q'_s]$ is co-connected. If q is a parallel node, considering the complement of G leads to the same conclusion than in the series case. Thus $G'[Q'_s]$ is connected and co-connected. We now show that $\{x\}$ is a maximal strong module of $G'[Q'_s]$.

Claim *If the insertion node q is an uncut prime node having no mixed child, then $\{x\}$ is a maximal strong module of $G'[Q'_s]$.*

Proof Let M' be the maximal strong module of $G'[Q']$ containing x. Since $G'[Q']$ is connected and co-connected, M' is actually the maximal module of $G'[Q']$ containing x. We denote $M = M' \setminus \{x\}$. Assume $M \neq \varnothing$, then, from Property 1, M is a module of $G[Q]$. Let Q_M be the maximal module of $G[Q]$ containing M. Q_M and M' fulfils assumptions of Lemma 6 in $G[Q]$, and thus $Q_M \cup M'$ is a module of $G'[Q']$. As M' is maximal, $Q_M \cup M' = M'$. Since, by definition of M and Q_M, $Q_M \cup M' = Q_M \cup \{x\}$, then $Q_M \cup \{x\} = M'$ and $M = Q_M$. Since $Q_M \cup \{x\}$ is a module of $G'[Q']$, q_M is a twin of x in the representative graph of q. This is impossible since q is uncut. Thus, $M = \varnothing$ and $\{x\}$ is a maximal strong module of $G'[Q']$. □

Claim *If the insertion node q is an uncut prime node having at least one mixed child, then $\{x\}$ is a maximal strong module of $G'[Q'_s]$.*

Proof Let M' be the maximal strong module of $G'[Q']$ containing x. We denote $M = M' \setminus \{x\}$. Assume $M \neq \varnothing$. As in the case where q has no mixed child, we denote Q_M the maximal module of $G[Q]$ containing M and we obtain that $M = Q_M$, since the proof made in that previous case is still valid. If q has a unique mixed child q_t,

since q is a non proper node and $Q_M \cup \{x\}$ is a module of $G'[Q']$, then q_M is not this unique mixed child of q, by definition of a non-proper node, $q_M \neq q_t$. If q has at least two mixed children, at least one of them q_t is different from q_M. In both cases, there exists a mixed child q_t of q such that $q_t \neq q_M$. Let $u \in Q_t$ such that $\{x, u\} \in E'$ and $v \in Q_t$ such that $\{x, v\} \notin E'$. Let $z \in Q_M$, since $Q_M \cup \{x\}$ is a module of $G'[Q']$ and $\{x, u\} \in E'$, then $\{z, u\} \in E'$. Since $Q_M \cup \{x\}$ is a module of $G'[Q']$ and $\{x, v\} \notin E'$, then $\{z, v\} \notin E'$. It is a contradiction with the fact that Q_t is a module of $G[Q]$. Thus, $M = \varnothing$ and $\{x\}$ is a maximal strong module of $G'[Q']$. □

Claim *If the insertion node q is an uncut series node, then $\{x\}$ is a maximal strong module of $G'[Q'_s]$.*

Proof Let M' be the maximal strong module of $G'[Q'_s]$ containing x. Since $G'[Q'_s]$ is connected and co-connected, M' is also the maximal module of $G'[Q'_s]$ containing x. We denote $M = M' \setminus \{x\}$. Note that, by definition, $M' \neq Q'_s$ and $M \neq Q_s$. Assume $M \neq \varnothing$, then, from Property 1, M is a module of $G[Q]$. Since $M \neq Q_s$, it is possible to find $f_j \in \mathcal{C}_{nl}(q) \cup \mathcal{C}_m(q)$ such that $F_j \cap M = \varnothing$. Since F_j is mixed or notlinked, there exists $y \in F_j$ such that $\{x, y\}$ is not an edge. Let $z \in M$. Since f_j is a child of q which is a series node and M is a module of $G[Q]$ such that $F_j \cap M = \varnothing$, $\{y, z\}$ is an edge. On the other hand, since M' is a module of $G'[Q']$, $y \notin M'$, $\{x, z\} \subseteq M'$ and $\{x, y\}$ is not an edge, then $\{y, z\}$ is not an edge. And we get a contradiction. Thus $M = \varnothing$ and $\{x\}$ is a maximal strong module of $G'[Q'_s]$. □

Claim *If the insertion node q is an uncut parallel node, then $\{x\}$ is a maximal strong module of $G'[Q'_s]$.*

The proof is quite similar to the proof in the case where q is an uncut series node (consider the complement of $G'[Q']$). □

The modular decomposition tree $T_{q'}$ of $G'[Q']$ is organised as follows. If q is a prime node, then, from Theorem 2, $G'[Q']$ is connected and co-connected. Consequently, q' is a prime node and its children correspond to the maximal uniform modules of $G[Q]$ (Theorem 2). If q is degenerate, then q' is degenerate and has the same label than q. If q is a series (resp. parallel) node, the set of children of q' is $\{q'_s\} \cup \mathcal{C}_l(q)$ (resp. $\{q'_s\} \cup \mathcal{C}_{nl}(q)$) where q'_s is a new node representing vertices of Q'_s. From Theorem 2, q'_s is a prime node and its children correspond to the maximal uniform modules of $G[Q]$ (Theorem 2). Note that when q is series (resp. parallel), it may happen that $\mathcal{C}_l(q)$ (resp. $\mathcal{C}_{nl}(q)$) is empty. In that case, $Q' = Q'_s$ and we do not need to create a new node q'_s. Given a maximal uniform module M of $G[Q_s]$, the modular decomposition tree of $G'[M]$ is the part of T_G restricted to M. We determined the whole modular decomposition tree $T_{G'}$. But we still do not know if G' is a permutation graph, and, in the positive, how to build the realisers of $\mathcal{R}_{G'}$. That is the purpose of next section.

4.2 Dynamic Characterisation of Permutation Graphs

In the case where x is an isolated or universal vertex, G' is a permutation graph and the full modular representation is easy to update in constant time. We do not consider

this case and focus on the case where T_G contains at least one non-proper node, see Lemma 3. As usual, we denote q the insertion node. In the previous section, we showed that the insertion of x in T_G reduces to the insertion of x in $T_{G[Q]}$. Lemma 8 shows that, similarly, inserting x in $MD(G)$ reduces to insert x in $MD(G[Q])$.

Lemma 8 *Let $G' = G + x$ and let q be the insertion node. G' is a permutation graph iff $G'[Q']$ is a permutation graph. Moreover, if G' is a permutation graph, $MD(G')$ is obtained from $MD(G)$ by replacing the subtree T_q of T_G with $T_{G'[Q']=G[Q]+x}$, and replacing, in $\mathcal{R}_G$, the realisers of the representative graphs of the prime nodes of T_q with the ones of the prime nodes of $T_{G'[Q']}$.*

Proof Since the permutation graphs family is hereditary, if G' is a permutation graph then $G'[Q']$ is. Conversely, since the permutation graphs family is closed under substitution composition (the invert operation of quotient) and since $G/\{Q\} = G'/\{Q'\}$ (see the proof of Lemma 5), if $G'[Q']$ is a permutation graph then G' is. In the proof of Lemma 5, we already showed that $T_{G'}$ is obtained from T_G by replacing the subtree T_q of T_G with $T_{G'[Q']=G[Q]+x}$. For every node of the complement part of $T_{G'}$ which is not in $T_{q'}$, we can find a set of representative vertices of the quotient that avoid x. Thus, the representative graphs of these nodes are the same as the ones of the corresponding nodes in T_G. Thus, we obtain $MD(G')$ by replacing, in $\mathcal{R}_G$, the realisers of the representative graphs of the prime nodes of T_q with the ones of the prime nodes of $T_{G'[Q']}$. □

We now propose a characterisation, based on $MD(G'[Q'])$, of the cases where G' is a permutation graph (Theorem 3). As we ask G' to be a permutation graph, the mixed nodes of T_q cannot be spread anywhere in the tree. Lemma 9 claims that there are at most two branches of mixed nodes in T_q rooted at q. These two branches correspond to the two families of Lemma 2.

Lemma 9 *If G' is a permutation graph then the insertion node q has at most two mixed children and any node $p \neq q$ of T_q has at most one mixed child.*

Proof If q is cut, then q has no mixed descendants and the statement holds.

So assume q is uncut. Let T_{q_s} be the modular decomposition tree of $G[Q_s]$. By definition of Q_s, T_{q_s} is obtained from T_q by removing some uniform children of q. Then, we can equivalently show the statement on T_{q_s} rather than on T_q. From Theorem 2, $H' = G'[Q'_s]/\mathcal{MSM}(G'[Q'_s])$ is a prime permutation graph. Still from Theorem 2, we have $H = G[Q_s]/\mathcal{MUM}(G[Q_s]) = H' - x$. From Theorem 2, the non-trivial strong modules of H can be divided in at most two families totally ordered by inclusion. Therefore the root of T_H has at most two children which are not leaves and any node of T_H distinct from its root has at most one child which is not a leaf.

By definition, any mixed strong module of $G[Q_s]$ is not a singleton and is the union of some modules of $\mathcal{MUM}(G[Q_s])$. Thereby, Lemma 1 applies and $\mathcal{M} \subseteq \mathcal{MUM}(G[Q_s])$ is a non-trivial strong module of H iff $P = \bigcup_{M \in \mathcal{M}} M$ is a non-trivial strong module of $G[Q_s]$. It follows that there is a bijection between the set of mixed strong modules of $G[Q_s]$ and non-singleton strong modules of H. To achieve

the proof, we need to prove that this bijection respects the parent/child relationship between the nodes of $G[Q_s]$. Formally, we show that if p_1 and p_2 are two mixed nodes of T_{q_s} such that p_1 is the parent of p_2, then their corresponding non-leaf nodes $\tilde{p}_1$ and $\tilde{p}_2$ in T_H are such that $\tilde{p}_1$ is the parent of $\tilde{p}_2$. Let $\mathcal{M}_1 \subseteq \mathcal{MUM}(G[Q_s])$ such that $P_1 = \bigcup_{M \in \mathcal{M}_1} M$. $\mathcal{M}_2$ is defined similarly. Note that $\mathcal{M}_1 = \widetilde{P_1}$ and $\mathcal{M}_2 = \widetilde{P_2}$. Since $P_2 \subseteq P_1$, necessarily, $\mathcal{M}_2 \subseteq \mathcal{M}_1$. Assume for contradiction that there exists some $\mathcal{M}_3 \subseteq \mathcal{MUM}(G[Q_s])$ such that $\mathcal{M}_3$ is a strong module of H and $\mathcal{M}_2 \subsetneq \mathcal{M}_3 \subsetneq \mathcal{M}_1$. Then, from Lemma 1, $P_3 = \bigcup_{M \in \mathcal{M}_3} M$ is a strong module of $G[Q_s]$. Moreover, $P_2 \subsetneq P_3 \subsetneq P_1$, which is a contradiction with the fact that p_1 is the parent of p_2. Thus, $\tilde{p}_1$ is the parent of $\tilde{p}_2$. □

Unfortunately, Lemma 9 is not a sufficient condition for G' being a permutation graph. Theorem 3 gives necessary and sufficient conditions. Given a graph $G = (V, E)$, $S \subsetneq V$ and $y \in V \setminus S$, we denote $G - yS = (V\ ,\ E \setminus \{\{y, z\} \mid z \in S\})$. If p is a node of T_q, then set $P' = P \cup \{x\}$. Since the maximal strong modules of $G[P]$ are uniform with respect to x in $G'[P'] - xF_m(p)$, they are modules of $G'[P'] - xF_m(p)$. We denote

$$H'_p = (G'[P'] - xF_m(p))/(\mathcal{MSM}(G[P]) \cup \{\{x\}\}).$$

Conditions of Theorem 3 precisely apply on the graphs H'_p where p is a node of T_q.

Theorem 3 *Let x be a vertex to be inserted in a permutation graph G. Then $G' = G + x$ is a permutation graph iff either the insertion node q of the modular decomposition tree T of G is cut; or if q is uncut then the nodes of T_q satisfy conditions 1 and 2 below.*

1. *q satisfies one of the two following conditions*:
   (a) *q has two mixed children f_1 and f_2, and H'_q is a permutation graph admitting a realiser $R_{H'_q} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1, and x and f_2 are consecutive in π_2.*
   (b) *q has a unique mixed child f_1, and H'_q is a permutation graph admitting a realiser $R_{H'_q} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1.*
   (c) *q has no mixed child and $H'_q = G'[Q']/(\mathcal{MSM}(G[Q]) \cup \{\{x\}\})$ is a permutation graph.*
2. *and any node $p \neq q$ of T_q satisfies one of the two following conditions*:
   (a) *p has a unique mixed child f_1, and H'_p is a permutation graph admitting a realiser $R_{H'_p} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1, and x is the first element of π_2.*
   (b) *p has no mixed child, and H'_p is a permutation graph admitting a realiser $R_{H'_p} = (\pi_1, \pi_2)$ such that x is the first element of π_2.*

Proof ⇒ If q is cut, no new prime nodes are introduced in T' (see Sect. 4.1), then G' is a permutation graph.

So let us assume that q is uncut. We show that the nodes of T_q satisfy the conditions of Theorem 3. The three claims below deal with the cases where the node of T_q

considered has at least one mixed child. The cases where it has no mixed child follow as particular cases of the cases treated in the claims.

Let f be a mixed child of q and F the corresponding strong module of G. Let $R' = (\pi'_1, \pi'_2)$ be a realiser of the representative graph $G'_{q'_s}$. As $\mathcal{MSM}(G'[Q'_s]) = \mathcal{MUM}(G[Q_s]) \cup \{\{x\}\}$ (see Theorem 2), removing x from R' results in a realiser R of $G[Q_s]/\mathcal{MUM}(G[Q_s])$. As F is a mixed strong module of $G[Q_s]$, F is not a singleton and Lemma 1 applies. That is the set $A = \{f' \in \mathcal{C}(q'_s) \mid F' \in \mathcal{MUM}(G[F])\}$ is a strong module of $G[Q_s]/\mathcal{MUM}(G[Q_s])$ and by Proposition 1, A is a common interval of the realiser R. Moreover as F is mixed, the set A surrounds x in π'_1 or π'_2.

Claim *If G' is a permutation graph and if the insertion node q of T is uncut and has two mixed children f_1 and f_2, then H'_q is a permutation graph admitting a realiser $R_{H'_q} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1, and x and f_2 are consecutive in π_2.*

Proof From the discussion above, the sets $A_1 = \{f \in \mathcal{C}(q'_s) \mid F \in \mathcal{MUM}(G[F_1])\}$ and $A_2 = \{f \in \mathcal{C}(q'_s) \mid F \in \mathcal{MUM}(G[F_2])\}$ are common intervals of the realiser R. As F_1 and F_2 are both mixed, A_1 and A_2 surround x in necessarily different linear order of R'. Without loss of generality, assume that A_1 surrounds x in π'_1 and A_2 in π'_2 and that elements of A_1 are smaller than elements of A_2 in π'_1 (otherwise, interchange and/or reverse the two orders of the realiser). Let us assume that F_1 does not neighbour F_2 (the other case is similar). Then the elements of A_1 are smaller than the elements of A_2 in π'_2. It also follows that q is either a *parallel* node, or a *prime* node and f_1 does not neighbour f_2 in G_q.

- Assume q is a prime node. Then $Q = Q_s$ and $\mathcal{MUM}(G[Q_s]) = \mathcal{MUM}(G[F_1]) \cup \mathcal{MUM}(G[F_2]) \cup (\mathcal{MSM}(G[Q]) \setminus \{F_1, F_2\})$. The realiser $R_{H'_q} = (\pi_1, \pi_2)$ of H'_q is obtained as follows. Remove x from π'_1 and π'_2. Merge the elements of A_1 into a unique element f_1 and those of A_2 into f_2. Finally, reinsert x right after f_1 in π_1 and right before f_2 in π_2.
- Assume q is a parallel node. Then by definition $Q \neq Q_s$ and $\mathcal{MUM}(G[Q_s]) = \mathcal{MUM}(G[F_1]) \cup \mathcal{MUM}(G[F_2]) \cup \{F_l\}$. To obtain the realiser $R_{H'_q} = (\pi_1, \pi_2)$ of H'_q, first proceed as in the prime case. Then replace the element f_l representing F_l by the set $\mathcal{C}_l(q)$ in the same relative order in π_1 and π_2. Finally, add in π_1 and π_2, the elements of $\mathcal{C}_{nl}(q)$ at the end of π_1 and π_2 so that they appear in the same relative order. □

Claim *If G' is a permutation graph and if the insertion node q of T is uncut and has a unique mixed child f_1, then H'_q is a permutation graph admitting a realiser $R_{H'_q} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1.*

Proof From the discussion above, the set $A_1 = \{f \in \mathcal{C}(q'_s) \mid F \in \mathcal{MUM}(G[F_1])\}$ is a common interval of the realiser R and A_1 surrounds x in some linear order of R', say π'_1. Without loss of generality, assume that x is smaller than the elements of A_2 in π'_2 (otherwise, reverse the two orders). Let us consider the different cases depending on the label of q.

- Assume q is a prime node. Then $Q = Q_s$ and $\mathcal{MUM}(G[Q_s]) = \mathcal{MUM}(G[F_1]) \cup (\mathcal{MSM}(G[Q]) \setminus \{F_1\})$. The realiser $R_{H'_q} = (\pi_1, \pi_2)$ of H'_q is obtained as follows. Remove x from π'_1 and π'_2. Merge the elements of A_1 into a unique element f_1. Finally, reinsert x right before f_1 in π_1 and at its original place in π_2.
- Assume q is parallel (the case q is a series node is similar). Then by definition $Q \neq Q_s$ and $\mathcal{MUM}(G[Q_s]) = \mathcal{MUM}(G[F_1]) \cup \{F_l\}$. To obtain the realiser $R_{H'_q} = (\pi_1, \pi_2)$ of H'_q, first proceed as in the prime case. Then replace the element f_l representing F_l by the set $\mathcal{C}_l(q)$ in the same relative order in π_1 and π_2. Finally, add in π_1 and π_2, the elements of $\mathcal{C}_{nl}(q)$ at the end of π_1 and π_2 so that they appear in the same relative order. □

Claim *If G' is a permutation graph, if the insertion node q of T is uncut and if $p \neq q$ is a node of T_q having a unique mixed child f_1, then H'_p is a permutation graph admitting a realiser $R_{H'_p} = (\pi_1, \pi_2)$ such that x and f_1 are consecutive in π_1, and x is the first element of π_2.*

Proof Let $R' = (\pi'_1, \pi'_2)$ be a realiser of $G'[Q']$. Removing x from R' results in a realiser R of $G[Q]$. Let q_1 be the child of q which is an ancestor of p. As Q_1, P and F_1 are strong modules of $G[Q]$, Q_1, P and F_1 are common intervals of R (see Proposition 1). As F_1 is mixed, F_1 (and therefore P and Q_1) surrounds x in some linear order of R', say π'_1 without loss of generality. Assume for contradiction that $P \cup \{x\}$ is an interval of π'_2. Then $Q_1 \cup \{x\}$ is an interval of π'_2 and is therefore a module of $G'[Q']$. Thereby q cannot have any mixed child different from q_1 and q cannot be non-proper, which is a contradiction. $P \cup \{x\}$ is not an interval of π'_2.

Let p_u be a child of p different from f_1. As P does not surround x in π'_2, P_u does not surround x in π'_2. Therefore, since P_u is uniform with respect to x, P_u cannot surround x in π'_1. Moreover, since P_u is a strong module of $G[Q]$, P_u is a common interval of $R = R'[V \setminus x]$. It follows, as P_u does not surround x in both π'_1 and π'_2, that P_u is a common interval of R'.

Let us now describe how to obtain $R_{H'_p} = (\pi_1, \pi_2)$. First, restricting R' to the vertices of $P \cup \{x\}$ results in a realiser R'_p of $G'[P \cup \{x\}]$ such that x is the first or the last element of the restriction of π'_2, without loss of generality say the first; and such that $F_1 \cup \{x\}$ is an interval of the restriction of π'_1. Move x in the restriction of π'_1 and place it right before F_1 (F_1 becomes a common interval). Then merging the common interval F_1 into a unique element f_1, and merging, for any child p_u of p different from f_1, the common interval P_u into a unique element p_u, yields the realiser $R_{H'_p} = (\pi_1, \pi_2)$. □

$\Leftarrow$ By Lemma 5, to conclude that G' is a permutation graph, we need to prove that $G'[Q']$ is a permutation graph. If q is cut, then the representative graph of any prime node is unchanged (see discussion following Definition 4) and thus G' is a permutation graph. Therefore let us assume that q is uncut.

We first show by induction that for any node $p \neq q$ of T_q, $G'[P']$ is a permutation graph admitting a realiser $R_{G'[P']} = (\pi'_1, \pi'_2)$ such that x is the first element of π'_2, where $P' = P \cup \{x\}$. If P is uniform with respect to x, the property holds. Thereby

leaves trivially satisfy the inductive hypothesis. Let p be a mixed node whose children satisfy the inductive hypothesis. To obtain the realiser $R_{p'}$, proceed as follows. In $R_{H'_p} = (\pi_1, \pi_2)$, substitute the realiser $R_{G[F]}$ for any $f \in \mathcal{C}(p)$ such that f is not mixed. If p has no mixed child, then Theorem 3 (2) guarantees that the resulting realiser has the wished property. So assume p has a unique mixed child f_1. By the inductive hypothesis, $G'[F'_1]$ has a realiser $R_{f'_1} = (\tau_1, \tau_2)$ such that x is the first element of τ_2. Therefore substituting, in π_1, τ_1 for the interval $\{x, f_1\}$ and substituting, in π_2, $\tau_2[F_1]$ for the element f_1, results in $R_{p'}$. Indeed by Theorem 3 (2), x is the first element of π_2 and remains by the above operations the first element of π'_2.

Let us now consider the insertion node q. If q has zero or one mixed child, then the discussion above on descendant nodes p of q applies and shows that a realiser of $G'[Q']$ can be built. $G'[Q']$ is therefore a permutation graph. Assume q has two mixed children f_1 and f_2. We proved that $G'[F'_1]$ and $G'[F'_2]$ respectively have realisers $R_1 = (\tau_1, \tau_2)$ and $R_2 = (\sigma_1, \sigma_2)$ such that x is the first element of τ_2 and σ_2. By assumption, q satisfies Theorem 3 (1). Assume without loss of generality that in $R_{H'_q} = (\pi_1, \pi_2)$, f_1 is smaller than f_2 in π_1. If f_1 does not neighbour f_2 in H'_q, then f_1 is before f_2 in π_2, and after f_2 otherwise. We consider only the case where f_1 and f_2 do not neighbour, the other one is similar. A realiser $R_{G'[Q']}$ is obtained as follows. In $R_{H'_q}$, substitute the realiser $R_{G[F]}$ for any $f \in \mathcal{C}(p)$ such that f is not mixed. Then substitute, in π_1, $\overline{\tau_1}$ for the interval $\{x, f_1\}$, and $\sigma_2[F_2]$ for f_2; and, in π_2, σ_1 for the interval $\{x, f_2\}$, and $\overline{\tau_2}[F_1]$ for f_1. □

4.3 Algorithm and Complexity

4.3.1 Data-Structure

To encode the full modular representation $MD(G) = (T_G, \mathcal{R}_G)$ of a permutation graph G, we shall first represent the modular decomposition tree. Each node maintains a pointer toward its parent and a list of pointers toward its children. Each node is given a label (prime, parallel or series) and each prime node p is associated with the realiser R_p of its representative graph G_p. The realiser $R_p = (\pi_1, \pi_2)$ will be stored in two doubly linked lists representing the two linear orders π_1 and π_2. Each cell of a list represents a child c of p and is doubly linked to c. Moreover each cell contains its rank in the list (namely $\pi_1(c)$ or $\pi_2(c)$), which allows to answer adjacency queries in constant time. Finally $\mathcal{R}$ also contains a realiser R_G of the whole graph G such that the cells of its two lists are doubly linked to the leaves of T_G.

4.3.2 Routine InsPrime

As a prime permutation graph G has a unique realiser $R = (\pi_1, \pi_2)$, $G + x$ is a permutation graph iff x can be inserted in R. Routine *InsPrime* performs, if possible, that insertion. The insertion positions in a linear order on n vertices are denoted $i + 0.5$, where $i \in [\![0, n]\!]$.

Lemma 10 *Let $R = (\pi_1, \pi_2)$ be the realiser of a prime permutation graph $G = (V, E)$, with $|V| = n$, and $x \notin V$ a vertex to be inserted. $G + x$ is a permutation*

graph iff there exists a couple $(i, j) \in [\![0, n]\!]^2$ *such that*

$$\begin{aligned} &\forall u \in N(x), \quad \pi_1(u) \leq i \quad \textit{iff} \quad \pi_2(u) > j \quad \textit{and} \\ &\forall v \in \overline{N}(x), \quad \pi_1(v) \leq i \quad \textit{iff} \quad \pi_2(v) \leq j. \end{aligned} \tag{1}$$

A realiser of $G + x$ *is obtained by inserting* x *at position* $i + 0.5$ *in* π_1 *and at position* $j + 0.5$ *in* π_2.

Proof Assume $G' = G + x$ is a permutation graph and let $R' = (\pi'_1, \pi'_2)$ be a realiser of G'. The restriction $R'[V]$ is a realiser of G which, by definition of a realiser, satisfies (1). Conversely, if for a realiser R of G, there exists i and j satisfying (1), then it can clearly be extended to a realiser of G' as described above. □

Formally, in the following, we call *insertion position* a couple $(i + 0.5, j + 0.5)$ such that (i, j) satisfies the conditions of Lemma 10.

Definition 5 An *initial common interval* of a realiser $R = (\pi_1, \pi_2)$ is a common interval of R containing both $\pi_1^{-1}(1)$ and $\pi_2^{-1}(1)$. For convenience, the empty set is considered as an initial common interval.

Note that the number of initial common intervals of a realiser is $O(n)$. They will play an important role in the detection of the insertion positions for x.

Remark Let $(i, j) \in [\![0, n]\!]^2$. The sets $\overline{N}_1(x) = \{v \in \overline{N}(x) \mid \pi_1(v) \leq i\}$ and $N_1(x) = \{u \in N(x) \mid \pi_1(u) \leq i\}$ are initial common intervals of, respectively, $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$ iff i and j satisfies (1) of Lemma 10.

Notation 2 Let $R = (\pi_1, \pi_2)$ be a realiser of a prime permutation graph. Let I be an initial common interval of $R[\overline{N}(x)] = (\pi_1[\overline{N}(x)], \pi_2[\overline{N}(x)])$. We denote $\bar{a}_1$ the right bound of I in $\pi_1[\overline{N}(x)]$ and $\bar{b}_1$ the successor of $\bar{a}_1$ in $\pi_1[\overline{N}(x)]$. $\bar{a}_2$ denotes the right bound of I in $\pi_2[\overline{N}(x)]$ and $\bar{b}_2$ the successor of $\bar{a}_2$ in $\pi_2[\overline{N}(x)]$. Let J be an initial common interval of $\overline{R}[N(x)] = (\pi_1[N(x)], \overline{\pi}_2[N(x)])$. We denote a_1 the right bound of J in $\pi_1[N(x)]$ and b_1 the successor of a_1 in $\pi_1[N(x)]$. b_2 denotes the right bound of J in $\overline{\pi}_2[N(x)]$ and a_2 the successor of b_2 in $\overline{\pi}_2[N(x)]$.

If $I = \varnothing$, $\bar{a}_1$ is undefined, and if $I = \overline{N}(x)$, $\bar{b}_1$ is undefined. Similarly, $\bar{a}_2$ and $\bar{b}_2$ may be undefined. These particular cases can be avoided by adding fictive vertices y_0 and y_{n+1} respectively at positions 0 and $n + 1$ in R, which are considered as being both in $N(x)$ and $\overline{N}(x)$. An initial common interval I of $R[\overline{N}(x)]$ becomes $I \cup \{y_0\}$. When I is empty, $\bar{a}_1 = y_0$, and when $I = \overline{N}(x)$, $\bar{b}_1 = y_{n+1}$. And similarly for the common intervals J of $\overline{R}[N(x)]$.

In the following, we do not consider anymore the cases where $I = \varnothing$ or $I = \overline{N}(x)$, as well as the cases where $J = \varnothing$ or $J = N(x)$.

Remark By definition, different intervals $[\bar{a}_1^i, \bar{b}_1^i]$ corresponding to different initial common intervals I_i of $R[\overline{N}(x)]$ can intersect only on their bounds. The same is also

true for different intervals $[\bar{a}_2^i, \bar{b}_2^i]$, as well as different intervals $[a_1^j, b_1^j]$ and different $[a_2^j, b_2^j]$ corresponding to different initial common intervals J_j of $\overline{R}[N(x)]$.

Lemma 11 *Let $R = (\pi_1, \pi_2)$ be a realiser of a prime permutation graph. Let I and J be initial common intervals of $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$ respectively. If, in π_1, $[\bar{a}_1, \bar{b}_1]$ and $[a_1, b_1]$ intersect, then $|[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]| = 2$. If, in π_2, $[\bar{a}_2, \bar{b}_2]$ and $[a_2, b_2]$ intersect, then $|[\bar{a}_2, \bar{b}_2] \cap [a_2, b_2]| = 2$.*

Proof We give the proof for π_1, the proof for π_2 is similar. Since $\{\bar{a}_1, \bar{b}_1\} \subseteq \overline{N}(x)$ and $\{a_1, b_1\} \subseteq N(x)$, $[\bar{a}_1, \bar{b}_1]$ and $[a_1, b_1]$ cannot share a bound. It follows that $|[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]| \geq 2$. Without loss of generality, assume that $a_1 <_{\pi_1} \bar{a}_1$. Since $[\bar{a}_1, \bar{b}_1]$ and $[a_1, b_1]$ intersect, then $\bar{a}_1 <_{\pi_1} b_1$. If the successor of $\bar{a}_1$ in π_1 is a non-neighbour of x, then, by definition, it is $\bar{b}_1$. Thus $|[\bar{a}_1, \bar{b}_1]| = 2$ and the proof is over. Otherwise, if the successor s of $\bar{a}_1$ in π_1 is a neighbour of x, then, since $\bar{a}_1 <_{\pi_1} b_1$ it follows that $s \leq_{\pi_1} b_1$. Since b_1 is the successor of a_1 in π_1 restricted to the neighbours of x, then $s = b_1$. Thus, $|[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]| = 2$. □

Corollary 1 rephrases Lemma 10 according to the discussion above.

Corollary 1 *Let G be a prime permutation graph and $R = (\pi_1, \pi_2)$ its realiser. $G + x$ is a permutation graph iff there exist I and J initial common intervals of respectively $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$, such that $[\bar{a}_1, \bar{b}_1]$ and $[a_1, b_1]$ intersect in π_1 and $[\bar{a}_2, \bar{b}_2]$ and $[a_2, b_2]$ intersect in π_2. And if such I, J exist, we obtain a realiser of $G + x$ by inserting x between the two elements of $[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]$ in π_1, and between the two elements of $[\bar{a}_2, \bar{b}_2] \cap [a_2, b_2]$ in π_2.*

Proof If $G + x$ is a permutation graph, there exist i, j satisfying (1) of Lemma 10. Then, $I = \overline{N}_1(x) = \{v \in \overline{N}(x) \mid \pi_1(v) \leq i\}$ and $J = N_1(x) = \{u \in N(x) \mid \pi_1(u) \leq i\}$ are initial common intervals of, respectively, $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$. From Notation 2 and definition of $\overline{N}_1(x)$ and $N_1(x)$, it follows that both $[\bar{a}_1, \bar{b}_1]$ and $[a_1, b_1]$ contain $\{\pi_1^{-1}(i), \pi_1^{-1}(i+1)\}$, and therefore intersect. Similarly, $[\bar{a}_2, \bar{b}_2]$ and $[a_2, b_2]$ intersect.

Conversely, if there exist I, J satisfying condition of the corollary, then from Lemma 11, $|[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]| = 2$ and $|[\bar{a}_2, \bar{b}_2] \cap [a_2, b_2]| = 2$. It is not difficult to see that inserting x between the two vertices of $[\bar{a}_1, \bar{b}_1] \cap [a_1, b_1]$ in π_1, and between the two vertices of $[\bar{a}_2, \bar{b}_2] \cap [a_2, b_2]$ in π_2, we obtain the correct adjacencies both between x and its neighbourhood and between x and its non-neighbourhood. □

Remark It is worth to notice that there is a bijection between the set of couples (I, J) of initial common intervals of respectively $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$ and the set of insertion positions for x in R.

Lemma 10 states that, for x a new vertex to be inserted in a prime permutation graph G, $G + x$ is a permutation graph iff there exists at least one insertion position for x in the realiser of G. The following theorem proves that there cannot be many of them.

Theorem 4 *Let $G = (V, E)$ be a prime permutation graph and $R = (\pi_1, \pi_2)$ its realiser. Let x be a vertex to be inserted in G such that V is not uniform with respect to x. There are at most two insertion positions for x in R (possibly none). Moreover, there are two different insertion positions iff x has a twin in $G + x$.*

Proof Consider two different insertion positions for x in R. Perform simultaneously the two corresponding insertions of x_1 and x_2 in R (x_1 and x_2 represent x in the two different possible insertions). Let $R_{ins} = (\pi_1^{ins}, \pi_2^{ins})$ be the resulting realiser. Let $a \in V$ be a vertex which is between x_1 and x_2 in one of the two orders of R_{ins}. Then, a is between x_1 and x_2 in the other order of R_{ins}. Otherwise, a would not be linked in the same way to x_1 and x_2 which is a contradiction since x_1 and x_2 both represent possible insertion positions for x in R. Consequently, the set B of vertices between x_1 and x_2 in π_1^{ins} are the same than the ones between x_1 and x_2 in π_2^{ins}. B is a common interval of R, then B is a module of G which is prime. The fact that V is mixed implies that $B \neq V$. It follows that $|B| = 1$. This shows that two insertion positions cannot be separated by more than one vertex in any of the two orders of the realiser. It follows that there are at most two insertion positions for x in R.

We denote by a the unique element of B. For any of the two possible insertion positions for x, $\{a, x\}$ is an interval of the resulting realiser R' of $G' = G + x$. Thus, $\{a, x\}$ is a module of $G + x$ and a is a twin of x in $G + x$. Conversely, if x has a twin a in G', then $\{a, x\}$ is a strong module of G'. In any realiser of G', $\{a, x\}$ is a common interval. Actually, since G is prime, $\{a, x\}$ is the unique module of G'. It follows that G' admits exactly two realisers, any one being obtained from the other by interchanging a and x in the two orders. The restriction of any of the two realisers of G' results in the unique realiser of G. Thereby x has two different insertion positions in R. □

It is easy to design an algorithm providing all common initial intervals of a realiser in $O(l)$ time, where l is the number of elements of the realiser. Let us now sketch Routine *InsPrime*.

1. First extract from $R = (\pi_1, \pi_2)$, the realisers $R[\overline{N}(x)]$ and $\overline{R}[N(x)]$ and compute their initial common intervals.
2. For each initial common interval I of $R[\overline{N}(x)]$, mark on π_1 the corresponding insertion interval among the non-neighbours of x, i.e. associate with $\bar{a}_1$ (resp. $\bar{b}_1$), an identifying label for I, the pointers to $\bar{a}_2$ and $\bar{b}_2$ and a pointer to $\bar{b}_1$ (resp. $\bar{a}_1$). Proceed similarly for the initial common intervals J of $\overline{R}[N(x)]$.
3. Scan π_1 looking for the intersections of some $[\bar{a}_1^i, \bar{b}_1^i]$ with some $[a_1^j, b_1^j]$. When such an intersection is found, check whether the corresponding $[\bar{a}_2^i, \bar{b}_2^i]$ and $[a_2^j, b_2^j]$ intersect in π_2. If so, by Corollary 1, output the insertion position for x in R and continue scanning π_1. Otherwise, continue scanning π_1.

Note that marking the insertion intervals in step 2 can be done in constant time for each initial common interval. Each intersection test in step 3 costs $O(1)$. Since the insertion intervals among the non-neighbours of x do not intersect except on their bounds, as well as the insertion intervals among the neighbours of x, the scan of π_1 described above takes $O(n)$ time.

From Theorem 4, Routine *InsPrime* finds at most two possible insertion positions for x in R. From Corollary 1, if no insertion position is found, then $G + x$ is not a permutation graph. If exactly one insertion position is found, then $G + x$ is a permutation graph and there is a unique way of inserting x in R. If two insertion positions are found, then, $G + x$ is a permutation graph, and from Theorem 4 x has a twin a in $G + x$, which is the only vertex between the two possible insertion positions in any of the two orders of the realiser R. Therefore, a can be found in constant time.

To summarise, if $G + x$ is a permutation graph, then in $O(n)$ time, Routine *InsPrime* returns a pair of doubly linked lists, the realiser of $G + x$, and outputs the twin of x if it exists.

4.3.3 The Typing Routine

This routine is exactly the marking process of [19], where the marks 1, −1 and 0 have been replaced by the types *linked*, *notlinked* and *mixed*. Each node receives its type in a bottom-up process. A leaf L_y of T_G is typed *linked* if $y \in N(x)$ and *notlinked* otherwise. Each node forwards its type to its parent node. If a node p receives the same type from all its children, p is given this type. Otherwise, p is given the type *mixed*. The process ends when the root is given a type. It is lightfull that the nodes typed *mixed* are exactly the mixed nodes, the nodes typed *linked* are the uniform nodes linked to x and the nodes typed *notlinked* are the uniform nodes not linked to x. Since the number of nodes in T_G is $O(n)$ and since each edge of T_G is crossed once, the typing routine runs in $O(n)$ time.

4.3.4 Finding the Insertion Node q

The purpose of this step is to find the insertion node q, in the case where the root r of T_G is typed *mixed*. By definition, q is the least common ancestor of the non-proper nodes of T_G. Any node p of the unique path between r and q is *mixed* and proper if $p \neq q$. Since, by Definition 2, any proper mixed node has a unique mixed child, finding the insertion node can be done by a top-down search of T_G following the path from r to q. The search stops when the current node p is non-proper, which can be tested as follows. If p is a series node (resp. parallel node), then p is proper iff all its children but one are typed *linked* (resp. *notlinked*) and the remaining child is *mixed*. If p is a prime node, p is proper iff x has a twin in the representative graph of p, which can be checked by Routine *InsPrime*. In both cases, testing whether p is a proper node can be done in $O(|\mathcal{C}(p)|)$. As T_G contains $O(n)$ nodes, the search finds the insertion node q in $O(n)$ time.

4.3.5 Maintaining the Full Modular Representation

Let us finally explain how it can be checked whether $G'[Q']$ is a permutation graph or not, and in the positive how the full modular representation can be updated. We first consider two simple cases.

- If the insertion node q has more than two mixed children, from Lemma 9, $G'[Q']$ is not a permutation graph, then the algorithm stops. The test can clearly be done in $O(|\mathcal{C}(q)|)$ time.

- If q is cut, then $G + x$ is a permutation graph (see Theorem 3) and the realisers of the representative graphs of prime nodes remain unchanged. From Definition 4, q is either a degenerate node with no mixed child but uniform children of both types, or a prime node with no mixed child but a child being a twin of x in G_q. Both cases can be tested in $O(|\mathcal{C}(q)|)$ time. This is clear for the former one. The latter one can be checked by a call to routine *InsPrime*. In both cases, updating T_G into $T_{G'}$ can be done, as described in Sect. 4.1, in $O(|\mathcal{C}(q)|)$ time.

Let us assume now that the insertion node q is uncut. First of all, note that from Lemma 4, the changes in T resulting from the insertion of x are located in the subtree T_q. Discussion below focuses on the computation of $MD(G'[Q']) = (T_{G'[Q']}, \mathcal{R}_{G'[Q']})$. To simplify the notations, let us set $T' = T_{G'[Q']}$ and $\mathcal{R}' = \mathcal{R}_{G'[Q']}$.

For seek of simplicity, we present the algorithm as a three-step process (note that, in practise, these three steps can be merged into a single one): 1) first, compute the modular decomposition tree T'; 2) determine if $G'[Q']$ is a permutation graph; and 3) in the positive, compute the realisers of $\mathcal{R}'$.

Computation of T' Discussion following the proof of Theorem 2 explains how T' can be obtained once the modular decomposition tree has been marked by the typing routine. The main task is to determine the new strong modules that appears with the insertion of x. From Theorem 2 these strong modules are the maximal uniform modules. They can be found in $O(n)$ time by a search in T_{q_s} since M is a maximal uniform module iff there exists a mixed node p descendant of q_s such that either p is degenerate and $M = F_l(p)$ or $M = F_{nl}(p)$; or p is prime and M is the vertex set of some uniform child of p.

Permutation Graph Test Let us consider the set Q'_s defined in Notation 1. Since for any maximal uniform module M of $G'[Q'_s]$, the modular decomposition tree of $G'[M]$ is inherited from T without adding any prime node, then $G'[Q'_s]$ is a permutation graph iff the representative graph $G'_{q'_s}$ is. Notice that $G'_{q'_s} = G[Q_s]/\mathcal{MUM}(Q_s) + x$ and that the full modular representation $MD(G[Q_s]/\mathcal{MUM}(Q_s)) = (\widetilde{T}, \widetilde{R})$ can be extracted from $MD(G)$ in $O(n)$ time. We need to check whether each node p of $\widetilde{T}$ fulfils the condition of Theorem 3.

Since q is uncut, the root of $\widetilde{T}$ is also uncut. If p is a degenerate node having the right number of mixed children (0, 1 or 2 depending on p being the root), then H'_p always enjoys a realiser satisfying Theorem 3 (see Fig. 4). If p is prime, using Routine *InsPrime* (which runs in $O(|\mathcal{C}(p)|)$ time), we insert x in $\widetilde{R}_p$ by making x adjacent to $\mathcal{C}_l(p)$ and non-adjacent to $\mathcal{C}_m(p) \cup \mathcal{C}_{nl}(p)$. There may be two different positions to insert x (only if x has a twin vertex). We then test if at least one of the possible positions fulfils the conditions of Theorem 3 which simply consists in testing the position of x in the realiser returned by *InsPrime* (extremity in an order and/or consecutiveness with the mixed child). That can be done in $O(1)$. Since we only handle the representative graphs of prime nodes of $\widetilde{T}$, each of which being processed in $O(|\mathcal{C}(p)|)$ time, the time complexity of this second step is $O(n)$.

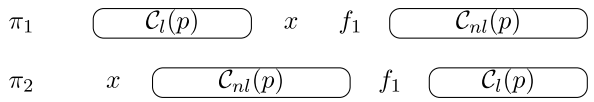

Fig. 4 The unique realiser of H'_p (if p is a series node) that fulfils condition 2a of Theorem 3. For a parallel node p, $\mathcal{C}_{nl}(p)$ and $\mathcal{C}_l(p)$ has to be exchanged in π_2

Computation of $\mathcal{R}' = \mathcal{R}_{G'[Q']}$ We now assume that $G' = G + x$ is a permutation graph. Remind that the representative graph $G'_{q'_s}$ is the graph $G[Q_s]/\mathcal{MUM}(Q_s) + x$ (see Theorem 2). Then, as for the testing step, the algorithm makes use of $MD(G[Q_s]/\mathcal{MUM}(Q_s)) = (\widetilde{T}, \widetilde{R})$. Notice that, for complexity issues, the ranks of the cells in the lists of the intermediate realisers computed along the process are not maintained.

To compute $\mathcal{R}' = \mathcal{R}_{G'[Q']}$, we apply the bottom-up process, described in the proof of Theorem 3, on $MD(G[Q_s]/\mathcal{MUM}(Q_s))$. For a prime mixed node p of $\widetilde{T}$, the realiser of H'_p is given by Routine *InsPrime*. For a degenerate node p of $\widetilde{T}$, the realiser of H'_p is the one depicted in Fig. 4. As the realisers are encoded by pairs of doubly linked lists, the substitution operation used in the proof of Theorem 3 can be done in $O(1)$ time. Thus, during the bottom-up process, each node p is handled in $O(|\mathcal{C}(p)|)$ time. It follows that the realiser $\mathcal{R}_s$ is computed in $O(n)$ time. Finally to maintain the whole data-structure, a scan of the lists of $\mathcal{R}_s$ allows to get the ranks of the cells.

Theorem 5 *Updating the full modular representation of a permutation graph under vertex insertion costs* $O(n)$ *time.*

References

1. Bergeron, A., Chauve, C., de Montgolfier, F., Raffinot, M.: Computing common intervals of K permutations, with applications to modular decomposition of graphs. In: 13th European Symposium on Algorithm (ESA'05). Lecture Notes in Computer Science, vol. 3669, pp. 779–790. Springer, New York (2005)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia (1999)
3. Bui-Xuan, B.M., Habib, M., Paul, C.: Revisiting uno and yagiura's algorithm. In: 16th International Symposium on Algorithms and Computation (ISAAC'05). Lecture Notes in Computer Science, vol. 3827, pp. 146–155. Springer, New York (2005)
4. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear time recognition algorithm for cographs. SIAM J. Comput. **14**(4), 926–934 (1985)
5. Crespelle, C., Paul, C.: Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. In: 31th International Workshop on Graph Theoretical Concepts in Computer Science (WG'05). Lecture Notes in Computer Science, vol. 3787. Springer, New York (2005)
6. Crespelle, C., Paul, C.: Fully dynamic recognition algorithm and certificate for directed cographs. Discrete Appl. Math. **154**(12), 1722–1741 (2006)
7. de Montgolfier, F.: Décomposition modulaire des graphes—Théorie, extensions et algorithmes. Ph.D. thesis, Université de Montpellier 2, France (2003)
8. Di Battista, G., Tamassia, R.: On-line planarity testing. SIAM J. Comput. **25**(5), 956–997 (1996)
9. Ehrenfeucht, A., Gabow, H.N., McConnell, R.M., Sullivan, S.J.: An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs. J. Algorithms **16**, 283–294 (1994)

10. Eppstein, D., Galil, Z., Italiano, G.F., Spencer, T.H.: Separator-based sparsification II: Edge and vertex connectivity. SIAM J. Comput. **28**(1), 341–381 (1998)
11. Gallai, T.: Transitiv orientierbare graphen. Acta Math. Acad. Sci. Hung. **18**, 25–66 (1967)
12. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic, New York (1980)
13. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM J. Comput. **31**(1), 289–305 (2001)
14. Ibarra, L.: Fully dynamic algorithms for chordal graphs. In: 10th ACM-SIAM Annual Symposium on Discrete Algorithm (SODA'99), pp. 923–924 (1999)
15. McConnell, R.M., Spinrad, J.: Linear-time transitive orientation. In: 8th ACM-SIAM Annual Symposium on Discrete Algorithm (SODA'97), pp. 19–25 (1997)
16. McConnell, R.M., Spinrad, J.: Modular decomposition and transitive orientation. Discrete Math. **201**(1–3), 189–241 (1999)
17. Möhring, R.H.: Algorithmic aspect of the substitution decomposition in optimization over relations, set systems and boolean functions. Ann. Oper. Res. **4**, 195–225 (1985)
18. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with combinatorial optimization. Ann. Discrete Math. **19**, 257–356 (1984)
19. Muller, J.H., Spinrad, J.P.: Incremental modular decomposition algorithm. J. Assoc. Comput. Mach. **36**(1), 1–19 (1989)
20. Nikolopoulos, S.D., Palios, L., Papadopoulos, C.: A fully dynamic algorithm for the recognition of P_4-sparse graphs. In: 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06). Lecture Notes in Computer Science, vol. 4271, pp. 256–268. Springer, New York (2006)
21. Shamir, R., Sharan, R.: A fully dynamic algorithm for modular decomposition and recognition of cographs. Discrete Appl. Math. **136**(2–3), 329–340 (2004)
22. Spinrad, J.: Efficient Graph Representations. Fields Institute Monographs, vol. 19. American Mathematical Society, Providence (2003)
23. Uno, T., Yagiura, M.: Fast algorithms to enumerate all common intervals of two permutations. Algorithmica **26**(2), 290–309 (2000)