HAL open science

Fully Dynamic Representations of Interval Graphs

Christophe Crespelle

To cite this version:

Christophe Crespelle. Fully Dynamic Representations of Interval Graphs. 35th International Workshop on Graph Theoretical Concepts in Computer Science - WG'09, Jun 2009, Montpellier, France. hal-00644222

HAL Id: hal-00644222

https://inria.hal.science/hal-00644222v1

Submitted on 26 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial 4.0 International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S0304397519300209
Manuscript_ffbe3655f6f6c0d126bb4e8c187b67f6

Fully Dynamic Representations of Interval Graphs[1]

Christophe Crespelle
Univ Lyon, UCB Lyon 1, ENS de Lyon, CNRS, Inria, LIP UMR 5668,
15 parvis René Descartes, F-69342, Lyon, FRANCE
christophe.crespelle@inria.fr

Abstract

We present a fully dynamic algorithm that maintains three different representations of an interval graph: a minimal interval model of the graph, the PQ-tree of its maximal cliques, and its modular decomposition. After each vertex or edge modification (insertion or deletion), the algorithm determines whether the new graph is an interval graph in $O(n)$ time, and, in the positive, updates the three representations within the same complexity. Furthermore, we state that the modular decomposition tree of an interval graph and the PQ-tree of its maximal cliques are algorithmically equivalent and we completely explicit the relationship between the two structures.

Keywords: interval graphs, dynamic algorithms, PQ-tree, modular decomposition

1. Introduction

Since Cook [2] introduced the notion of NP-completeness, recognition problems have played a central role in algorithmic graph theory (see e.g.[3]). Indeed, one possible way to cope with the difficulty of NP-complete problems is to restrict the set of instances taken as input by the algorithm, i.e., in case of graph algorithms, to restrict to a particular graph class. Doing so allows to design polynomial-time algorithms for problems that are NP-complete in general, by taking advantage of the specific structure of the considered graph class. This leans on the fact that, in practice, an algorithm will not have to consider all possible instances of graphs but only some of them, often coming from a specific context that confers them more properties than arbitrary graphs have. This approach therefore emphasizes on the problem of recognising whether a given graph belongs to a particular graph class $\mathcal{F}$, which is called the recognition problem associated to $\mathcal{F}$.

In this paper, we are interested in the recognition problem for the class of interval graphs. Interval graphs were introduced in [4] as the intersection graphs of intervals of the real line. They naturally appear in many applications coming from various contexts such as scheduling, genomic, archaeology and chemistry (see [5]). This is the reason why a lot of efforts have been made to better understand their structural properties and compute classical graph problems efficiently for the class. In particular, the recognition problem received a lot of attention [6, 7, 8, 9, 10, 11, 12, 13, 14] as well as the recognition of the subclass of proper (or unit) interval graphs[2] [15, 16, 17, 18, 19].

Here, we will not consider the recognition problem in its classic form, but instead in its dynamic version. In other words, we do not consider a graph given once and for all but a graph

[1]This work is a complete version of [1].
[2]Those interval graphs that admit a model in which all the intervals have the same length.

© 2019 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

subject to a series of modifications which are not known in advance. In practice, there is a strong need for such algorithms, as the graphs considered are often evolving during time, like for instance, a data-structure in some application, the data-base of a company, an intermediate result in an algorithm, and so on. In these situations where the graph considered is modified, the results of the computations made on the previous state of the graph are no longer valid. Then, if one wants to obtain these results for the new graph, the question arises to know whether the computation has to be done entirely again or whether it is possible to obtain the results faster, taking into account the fact that the structure of the graph has been only very partially modified. This is actually not only a practical necessity but also a fundamental theoretic question: what is the sensitivity of an algorithm to a slight modification of its input? Of course, if any modification is allowed, the problem cannot be solved faster than the static corresponding problem, as the input may entirely change. But on the opposite, if the modifications considered are elementary, like adding or removing an edge or a vertex of the graph, then the question to know whether it is possible to update the result significantly faster than the computation from scratch on the new graph is of key interest.

The properties shared by a class of graphs often give rise to some specific representations of the graphs in the class. These representations are fundamental: they give a deeper insight on the structure of the graphs and often allow to solve algorithmic problems very efficiently for the class [20]. Therefore, recognition algorithms, either static or dynamic, often aim not only at determining whether the input graph belongs to the specified class but also at providing such characteristic representations when it does so. In this paper, we aim at dynamically maintaining three characteristic representations of interval graphs. The first one is an *interval model*, which constitutes the primary definition of the class (see Section 2.1 below) and which allows to solve some difficult problems, such as colouring for example, very efficiently for the class. The second, and most fundamental representation of interval graphs, which encompasses and goes beyond interval models, was introduced in [6] and completed in [21]. It is known as PQ-trees, or MPQ-trees. Instead of giving only one particular interval model of the graph, PQ-trees provide a very efficient representation of all its interval models. This is of key interest as some problems are sensitive to the choice of the intersection model used to solve them, see e.g. [22]. In other words, these problems may be easily solvable with some models but not with some others. Therefore, the use of PQ-trees allows to choose the most appropriate model to solve a given problem and then results in efficient solutions for many problems. Finally, the third representation we use here for interval graphs is general to all arbitrary graphs: this is the modular decomposition of graphs (see section 2.2 below for a definition). It is known that modular decomposition performs very well for interval graphs and that this representation of the class presents strong connections with PQ-trees. In particular, it has been noticed that when a problem admits an efficient solution based on one of these two representations, it is in general possible to design an equivalently efficient solution based on the other representation. In this paper we go beyond this empirical remark by formally establishing an algorithmic equivalence between PQ-trees and modular decomposition of interval graphs.

More formally, the problem we consider in this paper lies in the general framework of *dynamic recognition and representation problems* [23, 24, 25, 26, 27, 28, 29, 30, 31], which aims for a family $\mathcal{F}$ of graphs at determining whether a dynamically changing graph belongs to $\mathcal{F}$ and at maintaining a characteristic representation of the modified graph as long as it belongs to $\mathcal{F}$. The input of the problem is a graph $G \in \mathcal{F}$ with its representation and a modification which is one of the following: inserting or deleting a vertex (along with the edges incident to it), inserting or deleting an edge. After any modification, the algorithm determines whether the new graph belongs to $\mathcal{F}$ and, in the positive, updates the chosen representation. In this paper, we consider the dynamic recognition and representation problem for the class of interval graphs,

and design an $O(n)$ time algorithm that maintains three key representations of the class under the four basic modification operations, namely an interval model, the PQ-tree and the modular decomposition.

Related works.

The seminal paper for the recognition of interval graphs [6] solved the problem in linear time by introducing a data structure called PQ-tree. The algorithm of [6] is not dynamic: though the consecutiveness constraints of each vertex are added one by one, the graph needs to be known completely in advance since the algorithm first makes use of a Lex-BFS sweep in order to compute all the maximal cliques of the graph.

The algorithm of [11] also considers the vertices arriving one by one and updates the PQ-tree. But in order to achieve a linear complexity, the ordering on the vertices is not arbitrary. It is actually a Lex-BFS ordering of the whole graph, which must be precomputed statically.

On the opposite, the algorithm of [32] is truly incremental on vertices and processes the whole graph in $O(m + n\,log\,n)$ time. In the worst case, the cost of a vertex insertion may be up to $\Omega(n)$. This amortised complexity leans on a data structure that, as mentioned by the author, does not allow to treat vertex deletion, while our algorithm is able to do so, within the same worst case time complexity.

Recently, [33] designed a fully dynamic algorithm, expressed in terms of 0-1 matrices, which maintains the PQ-tree of an interval graph and whose running time matches the one of our algorithm. Moreover, the algorithm of [33] is able to provide, in $O(n \log n)$ time, a certificate that the modified graph is not an interval graph, when this occurs. Interestingly, the description of [33]'s algorithm is somehow simple. Nevertheless, it should be noted that this simplicity mainly relies on the fact that [33]'s algorithm uses external primitives, such as the incremental step of [6], whose description is in itself quite intricate. On the opposite, the algorithm we present here is self-contained and is closer to implementation.

For edge modifications, [34] designed a fully dynamic algorithm that runs in $O(n\,log\,n)$ time per operation. Here, we lower this complexity to $O(n)$.

For the sake of completeness, let us mention that [27] designed a fully dynamic algorithm for recognising proper interval graphs that handles each update (edge or vertex) in $O(d + \log n)$ time, where d is the number of edges involved in the operation. The representation they use is very specific to proper interval graphs and is not applicable to arbitrary interval graphs. Note that a similar result was obtained in [35] when only edge modifications are allowed, using a different data structure.

Our results.

Our algorithm is the first one treating the insertion of a vertex in an interval graph in a truly dynamic manner and dealing with the deletion of a vertex as well. We also lower the complexity of the best dynamic algorithm for edges [34] from $O(n \log n)$ to $O(n)$ per operation, insertion or deletion. In addition, we do not only deal with the recognition problem but also maintain three fundamental representations of the graph. It should be noted that this result has been used by [22] to design the fastest known algorithm for interval completion of an arbitrary graph. Theorem 1 states our main result.

Theorem 1. *There is a fully dynamic algorithm that maintains a minimal interval model, the PQ-tree and the modular decomposition of an interval graph under vertex and edge modifications, insertion and deletion, in $O(n)$ time per operation.*

Compared to [6], we deal with the full difficulty of the vertex insertion problem by dynamically computing the maximal cliques of the modified graph and inserting them at their

right place in the PQ-tree; while [6] statically pre-computes the maximal cliques and considers incrementally only the consecutiveness constraints. Though [11] dynamically computes the maximal cliques of the modified graph, it does not tackle the vertex insertion problem in its whole difficulty, since the statically precomputed order of the vertices used in [11] gives rise to very restricted vertex insertions. Their algorithm cannot treat arbitrary vertex insertions and the graph must be completely known in advance.

Beside our algorithmic results, we give new insight into the structure of interval graphs by making completely explicit the non-trivial connection between the PQ-tree and the modular decomposition of an interval graph. As a consequence, the work of [36, 24] for inserting a vertex in the modular decomposition also applies to the PQ-tree of interval graphs. In addition, we show that the modular decomposition tree of an interval graph and the PQ-tree of its maximal cliques are *linear-time equivalent*. That is, any one of these two representations can be computed from the other one in linear time with regard to the size of the input, that is $O(n)$ time. Thus, together with the result of [37], which showed the linear-time equivalence of a minimal interval model and the PQ-tree of an interval graph, we obtain that the modular decomposition of an interval graph can be retrieved from any of its minimal models in $O(n)$ time. This extends to the class of interval graphs a result of [38, 39, 40] for permutation graphs.

Outline of the paper.

Section 2 gives some necessary background on interval graphs, PQ-trees and modular decomposition. Section 3 describes the implementations of the three representations we use and states the linear-time equivalence of the PQ-tree and the modular decomposition of an interval graph. Section 4 presents our fully dynamic algorithm for edge operations and vertex deletion, and Section 5 presents vertex insertion, which is the core of our algorithm.

2. Preliminaries

Every graph $G = (V, E)$ considered here will be finite, undirected, loopless and simple. We denote $V(G)$ the set of vertices of G and $n = |V(G)|$. The edge between vertices x and y will arbitrarily be denoted either xy or yx. The neighbourhood of a vertex $x \in V$ is denoted $N(x)$ and its non-neighbourhood $\overline{N}(x)$. For a subset $S \subseteq V$ of vertices, the neighbourhood of S, denoted $N(S)$, is defined as the set $\bigcup_{x \in S} N(x) \setminus S$. $\mathcal{K}(G)$ is the set of maximal cliques of G. A vertex x is simplicial in G iff its neighbourhood is a clique. For a rooted tree T and a node u of T, we denote $\text{parent}_T(u)$ for the parent of u in T, $\text{Anc}_T(u)$ (resp. $\text{Desc}_T(u)$) for the ancestors (resp. descendants) of u in T ($u \in \text{Anc}_T(u) \cap \text{Desc}_T(u)$), and T_u for the subtree of T rooted at u. We may sometimes identify the tree and its set of nodes by abusively denoting $u \in T$. The set of children of a node $u \in T$ is denoted $\mathcal{C}_T(u)$. When the tree referred to is clear from the context, we omit to precise it by the subscript. A linear ordering σ on a ground set X is a total order on X, i.e. an order relation such that any pair of elements of X are comparable. We use the notation $x \leq_\sigma y$ when x is less than or equal y in σ, and $x <_\sigma y$ when $x \leq_\sigma y$ and $x \neq y$. We also denote $\min(\sigma)$ and $\max(\sigma)$ for respectively the minimum and maximum element of σ. A linear ordering σ can equivalently be seen as a sequence containing each element of X exactly once, with the meaning that an element x appears before another one y in the sequence iff $x <_\sigma y$. We also consider that the sequence is written from left to right and we say that x is on the left of y, or y on the right of x. Then, $\min(\sigma)$ is also referred to as the first or leftmost element of σ and $\max(\sigma)$ as the last or rightmost element.

2.1. Interval graphs

Interval graphs are the intersection graphs of intervals of the real line (see [5]). An interval model of a graph $G = (V, E)$ is a set $\mathcal{I}$ of intervals of the real line along with a mapping from

V to $\mathcal{I}$ such that two vertices of G are adjacent iff their corresponding intervals intersect. The class remains the same if intervals are required to have integer bounds and to be closed. All interval models considered in the following satisfy this restriction. Associating with each vertex of a graph the two integer bounds of its corresponding interval in some model of G yields an efficient data structure providing adjacency in constant time. A minimal interval model is an interval model whose number of distinct interval bounds is minimum among all interval models.

Interval graphs are well known to be *chordal*, that is they do not contain any induced cycle of length ≥ 4. They admit a bunch of nice characterisations, such as the following.

Theorem 2. [41] *A graph G is an interval graph iff its maximal cliques can be linearly ordered such that, for every vertex x of G, the maximal cliques containing x occur consecutively.*

Such an ordering of the maximal cliques is called a *consecutive ordering* of G. Numbering the maximal cliques with their rank in a consecutive ordering σ and assigning to each vertex x of G the interval of the cliques containing x in σ results in a model of G. Furthermore, such a model is minimal.

[6] shows that all the consecutive orderings of the maximal cliques of an interval graph G can be represented by a $O(|\mathcal{K}(G)|)$-space structure called PQ-tree.

Definition 1 (PQ-tree). *A PQ-tree is a rooted tree whose leaves are the maximal cliques of an interval graph G and whose internal nodes are labelled P (*degenerate nodes*) or Q (*prime nodes*), such that any Q-node q has at least three children and is assigned two linear orderings, denoted σ_q and $\bar{\sigma}_q$, on the set of its children, $\bar{\sigma}_q$ being the reverse order of σ_q.*

A *solidification* of a PQ-tree T, is the tree T itself where each node p of T is assigned a linear ordering on its children : any linear ordering if p is a P-node, σ_p or $\bar{\sigma}_p$ if p is a Q-node. The *frontier of a solidification* s is the prefix order of the leaves of T resulting from a depth first search where the children of a given node p of T are explored in the order defined by s. The founding result of PQ-trees is the following.

Theorem 3 ([6]). *Let G be an interval graph. Then, there exists a unique PQ-tree T such that the frontier is a one to one mapping from the set of solidifications of T onto the set of consecutive orderings of G.*

In other words, Theorem 3 states that for any interval graph G, there exists a PQ-tree, which is moreover unique, such that by rearranging the orders of the children of the nodes of this PQ-tree in the regular way described above we obtain all consecutive orderings of G exactly once (that is, different choices of orders lead to different consecutive orderings). This gives rise to the following definition.

Definition 2 ($T^c(G)$). The PQ-tree *of an interval graph G is the unique PQ-tree whose existence is stated by Theorem 3. It is denoted $T^c(G)$, or simply T^c when the graph G referred to is clear from the context.*

2.2. Modular decomposition

Here, we give some known definitions and results that we use in the following (see [42, 43] for surveys). Let $G = (V, E)$ be a graph. A subset $S \subsetneq V$ of vertices is *uniform* with respect to vertex $x \in V \setminus S$ if $S \subseteq N(x)$ (S is *full*) or $S \subseteq \overline{N}(x)$ (S is *hollow*). If S is not hollow, S is *linked*, and *mixed* if S is neither hollow nor full. If vertex y is adjacent to x, we say that y is linked to x. When there is no confusion, we omit to mention the vertex x referred to.

A *module* of a graph $G = (V, E)$ is a subset of vertices $M \subseteq V$ which is uniform with respect to any vertex $x \in V \setminus M$ (by convention, when $V \setminus M = \varnothing$, i.e. when $M = V$, M is a module). It follows from definition that V and the singletons $\{x\}, x \in V$, are modules of G, namely the *trivial modules*. A graph is *prime* if it contains only trivial modules.

For a module M of G, we define the quotient graph $G/M = G[(V \setminus M) \cup \{a\}]$, where $a \in M$ is called the representative vertex of M. As M is a module, G/M does not depend on the representative vertex chosen. Similarly, for a family $\mathcal{P}$ of pairwise disjoint modules, we define the quotient graph by choosing a representative vertex for each module in $\mathcal{P}$. Namely, if $\mathcal{P} = \{M_1, \ldots, M_k\}$, then $G/\mathcal{P} = G[(V \setminus \bigcup_{1 \leq i \leq k} M_i) \cup \{a_1, \ldots, a_k\}]$, where $a_i \in M_i$ for all $i \in [\![1, k]\!]$. The *substitution composition* is the converse operation of quotient. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs and $x \in V_G$. The composed graph $G_{x \leftarrow H}$ is defined by $G_{x \leftarrow H} = (V, E)$ with $V = (V_E \setminus \{x\}) \cup V_H$ and $E = (E_G \setminus \{e \mid e \in E_G, x \in e\}) \cup \{yz \mid y \in V_G \setminus \{x\}, z \in V_H, xy \in E_G\} \cup E_H$. Note that, by definition, V_H is a module of $G_{x \leftarrow H}$.

Two sets S_1 and S_2 *overlap* iff $S_1 \cap S_2 \neq \varnothing$ and $S_1 \setminus S_2 \neq \varnothing$ and $S_2 \setminus S_1 \neq \varnothing$. A module M is *strong* if it does not overlap any module M', that is, for all modules $M' \neq M$, $M \cap M' = \varnothing$ or $M \subseteq M'$ or $M' \subseteq M$. We call *maximal strong modules* of a graph $G = (V, E)$, denoted $\mathcal{MSM}(G)$, the strong modules of G maximal with regard to inclusion and distinct from V. The modular decomposition tree of G, denoted $T^{\mathrm{m}}(G)$ (or simply T^{m}), is the transitive reduction of the inclusion order of the strong modules of G. The leaves of T^{m} are the vertices of G and a node $p \in T^{\mathrm{m}}$ represents the strong module P of G which is the set of leaves of T^{m}_p. The children of a node p of T^{m} are the maximal strong modules of $G[P]$. In order to store the edge set of the graph, to each node p of T^{m}, we associate its quotient graph $G_p = G[P]/\mathcal{MSM}(G[P])$. Then G can be retrieved by applying the substitution composition along all the edges of T^{m}. From the well-known modular decomposition theorem, the quotient G_p is either a stable (iff $G[P]$ is not connected), then p is labelled *parallel*, or a clique (iff $G[P]$ is not co-connected), then p is labelled *series*, or a prime graph (iff $G[P]$ is connected and co-connected), then p is labelled *prime*. The parallel and series nodes are also called degenerate nodes. In the following, for any node $p \in T^{\mathrm{m}}(G)$, we denote $V(p)$ for the set of leaves of T^{m}_p. We say that node p is uniform, full, hollow or mixed referring to the subset $V(p)$ of $V(G)$ it represents.

Remark 1. *Note that $MD(G)$ allows to answer adjacency queries between any pair of vertices x, y of G. To that purpose, one has to find the least common ancestor u of x and y in T^m and check if the children of u containing respectively x and y are adjacent in the quotient graph G_u.*

Two vertices x, y of a graph G are *twins* when $\{x, y\}$ is a module of G, or in other words when $N(x) \setminus \{y\} = N(y) \setminus \{x\}$. Traditionally, when x and y are twins and are adjacent, they are called *true twins* and when they are twins but not adjacent, they are called *false twins*. In this article, we do not use this vocabulary and simply say that x and y are twins when they are either true twins or false twins.

Throughout the paper, we will extensively use the remarkable properties of interval graphs with regard to substitution decomposition. However, the situation is not as simple as one could expect since the composed graph of two interval graphs may not be an interval graph. The following lemma characterises the cases where such a composition results in an interval graph.

Lemma 1. *Let G and H be interval graphs, and x a vertex of G. $G_{x \leftarrow H}$ is an interval graph iff:* ***(i)*** *x is simplicial; or* ***(ii)*** *H is a clique.*

Proof. In [9], it is shown that for any module M of a chordal graph, M is a clique or its neighbourhood is a clique.

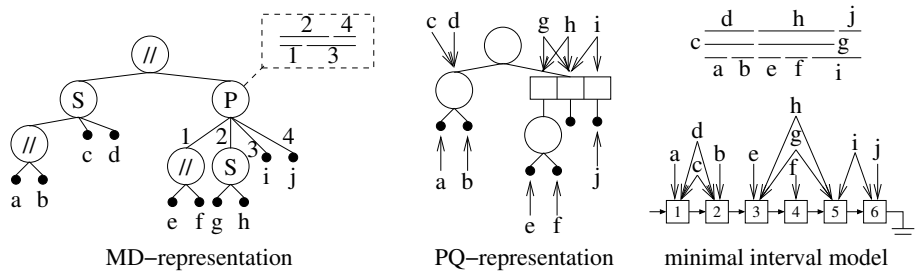

Figure 1: Three representations of an interval graph. In the PQ-representation, the degenerate nodes are represented by circles and the prime nodes by rectangles. Note that in the minimal interval model above, for the vertices having two pointers toward the same cell of the list, only one of them has been represented.

If $G_{x\leftarrow H}$ is an interval graph, then it is chordal. Since $V(H)$ is a module of $G_{x\leftarrow H}$, it follows that H is a clique or the neighbourhood of $V(H)$ in $G_{x\leftarrow H}$ is a clique. When the latter is true, since the neighbourhood of x in G is precisely the neighbourhood of $V(H)$ in $G_{x\leftarrow H}$, x is simplicial in G. Thus, the conditions of Lemma 1 are necessary.

Conversely, if these conditions are satisfied, let σ be a consecutive ordering of G. It is clear that if H is a clique, substituting $V(H)$ to x in each clique of σ results in a consecutive ordering of $G_{x\leftarrow H}$. On the other hand, if x is simplicial, there is a unique clique K_x of G containing x. Then, substituting K_x, in σ, with a consecutive ordering of the cliques of H augmented with the vertices of $K_x \setminus \{x\}$ results in a consecutive ordering of $G_{x\leftarrow H}$. Consequently, $G_{x\leftarrow H}$ is an interval graph. □

3. Three representations of interval graphs

In Section 3.1, we describe the three classic representations maintained by our algorithm. We introduce some notations to refer to them and we give implementation details of these structures, as they will matter in the algorithmic part of the paper (Section 3.2 and 5.3). In Section 3.2, we show the linear-time equivalence between the PQ-representation and the MD-representation. And in Section 3.3, we show the linear-time equivalence between the PQ-representation and a minimal interval model.

3.1. Implementation

Here, we introduce the notations and the implementation details we use to manipulate the three classic representations maintained by our algorithm.

3.1.1. Minimal interval models

A minimal interval model of an interval graph G is a consecutive ordering σ of G stored as a list. Each cell of the list contains only its rank in the list and each vertex of G is assigned two pointers (possibly the same) toward the cells representing the first and the last (with respect to σ) maximal clique of G containing x. In particular, note that the maximal cliques of G are not stored in extension in the cells of the list. One can determine the adjacency relationship of any two vertices of the graph by comparing the rank of the cells to which these vertices point in the list (see Figure 1). This can be done in constant time. The size of such a structure is clearly $O(n + |\mathcal{K}(G)|) = O(n)$, as $|\mathcal{K}(G)| \leq n - 1$ for any interval (actually chordal) graph.

The reason why we choose lists instead of array comes from the fact that in our dynamic algorithm we need to concatenate different models in order to get a new one, which can be done in constant time using lists, if we do not re-number the cells of the concatenated model. And

indeed, for complexity reasons, our algorithm computes the new numbering of the cells only once at the end of all the concatenations made by the algorithm, and not in the intermediate results (see Section 5.3.4).

3.1.2. The PQ-representation

The PQ-trees introduced by Booth and Lueker [6] are a fundamental combinatorial object that encodes all the linear orderings σ of a finite set X of elements that respect a set of linear constraints (i.e. subsets of X required to be intervals of σ). In the case of interval graphs, X is the set of maximal cliques of the graph and the linear constraints are defined by the vertices of the graph: the subset of cliques containing a given vertex must form an interval in any consecutive ordering of the maximal cliques. See Section 2.1 for a more detailed introduction to PQ-trees. The complete view of this object was given later by Korte and Möhring [21]: instead of storing the cliques of the graph in extension in the leaves of the tree, they propose to store the vertices of the graph in the internal nodes of the tree. This has two key advantages: first, this makes the consecutiveness constraints appear explicitly in the structure and second, this lowers the total encoding size of the structure from $O(n+m)$ to $O(n)$.

The PQ-representation we use here is essentially the same structure as the MPQ-tree introduced in [21]. However, we formalise it in a different way that fits better our purposes. In the following, $G = (V, E)$ is an interval graph and T^{c} is its PQ-tree. We use the following notations.

Notation 1. *For any vertex $x \in V$, we denote by e_x the least common ancestor of the leaves of T^c corresponding to the maximal cliques of G containing x.*

The key property used by [21, 11] to design the PQ-representation is the following.

Lemma 2. *[11] For any vertex x of an interval graph G, exactly one of the two following conditions holds:*
(i) *the maximal cliques of G containing x are exactly the leaves of $T^c_{e_x}$, or*
(ii) *e_x is a prime node and there exist $(u_1, u_2) \in (\mathcal{C}(e_x))^2 \setminus \{(\min(\sigma_{e_x}), \max(\sigma_{e_x}))\}$ such that $u_1 \neq u_2$ and the maximal cliques of G containing x are exactly the union of leaves of all the subtrees T^c_v for v a child of e_x between u_1 and u_2.*

Note that Conditions **(i)** and **(ii)** of Lemma 2 are mutually exclusive.

Notation 2. *When Condition **(ii)** of Lemma 2 is satisfied, we denote e^1_x and e^2_x for the children u_1 and u_2 of e_x.*

The PQ-representation is defined as follows.

Definition 3. *The PQ-representation of an interval graph G, denoted $PQ(G)$, is made of its PQ-tree $T^c(G)$ plus the set of vertices of G where each vertex x stores a* primary pointer *toward node e_x of $T^c(G)$, and two* secondary pointers *toward resp. e^1_x and e^2_x when x satisfies Condition **(ii)** of Lemma 2.*

We will extensively use the following notations to refer to the subsets of vertices of G pointing to some nodes of the PQ-representation. For each of them, an example is given on Figure 2.

Notation 3. *(cf. Figure 2) Let r be the root of T^c. For any node u of T^c, we define the following sets:*
$X_u = \{x \in V(G) \mid e_x = u \text{ and } x \text{ has no secondary pointers}\}$

$$v^* = \{a, b, c, d\}$$
$$\Delta_s = \{i\}$$
$$B_{t_1} = \{f, m, g, h, k, l, o\}$$
$$B_{t_2} = \{e, m, g, h, k, l, o\}$$
$$B_{u_1} = \{m, g, h, k, l, o\}$$
$$u_1^* = \{e, f, m\}$$

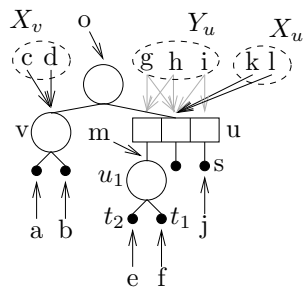

Figure 2: Some examples of sets X_u, Y_u, u^*, Δ_u and B_u. The primary pointers are black, while the secondary pointers are grey. The primary pointers from the vertices having secondary pointers are not represented. One can see that B_{u_1} is the subset of vertices involved in all the maximal cliques being leaves of $T^c_{u_1}$, i.e. $B_{u_1} = B_{t_1} \cap B_{t_2} = \{m, g, h, k, l, o\}$, and u_1^* is the subset of vertices involved only in the leaves of $T^c_{u_1}$ and no others, i.e. $u_1^* = \{e, f, m\}$, and set $X_{u_1} = \{m\}$ is the subset of vertices that are involved in all the leaves of $T^c_{u_1}$ and only in those cliques. Finally, $B_{u_1} \cup u_1^* = \{e, f, m, g, h, k, l, o\}$ is the subset of vertices involved in some leaf of $T^c_{u_1}$.

$$Y_u = \{x \in V(G) \mid e_x = u \textit{ and } x \textit{ has secondary pointers}\}$$
$$u^* = \{x \in V(G) \mid e_x \in T^c_u\}$$
$$\Delta_u = \begin{cases} \{x \in Y_v \mid e^1_x \leq_{\sigma_v} u \leq_{\sigma_v} e^2_x\} & \textit{if } u \neq r \quad \textit{where } v = \textit{parent}(u) \\ \varnothing & \textit{if } u = r \end{cases}$$
$$B_u = \bigcup_{v \in Anc(u)} (X_v \cup \Delta_v)$$

In the following, set B_u is referred to as the *branch* of node u. If vertex $x \in \Delta_u$ for some node u of T^c, we say that x *covers* u. Note that, by definition, if u is degenerate then $Y_u = \varnothing$, and consequently, if parent(u) is degenerate then $\Delta_u = \varnothing$.

Notation 4 ($C_\Delta(x)$). *For u a prime node and $x \in Y_u$, we denote $C_\Delta(x)$ the interval of children of u covered by x, i.e. $C_\Delta(x) = \{v \in \mathcal{C}(u) \mid e^1_x \leq v \leq e^2_x\}$.*

Remark 2. *For computational reasons, in the PQ-representation, not only the vertices of the graph store pointers to the node of T^c, but conversely, each node u of T^c also stores the sets of vertices that point toward u. Namely, each node u stores the sets X_u, Y_u, and the sets $\{x \in V(G) \mid e^1_x = u\}$ and $\{x \in V(G) \mid e^2_x = u\}$.*

One of the key advantages of the PQ-representation is that one do not need to store in extension, for each leaf f of T^c, the list of vertices belonging to the maximal clique K_f corresponding to f: K_f is the union, over all ancestors u of f in T^c, of the vertices of X_u and of the set of vertices x such that u is between e^1_x and e^2_x (i.e. x covers u), when the parent of u is a prime node. In other words, with the notations we have just adopted, we have the following.

Remark 3. *The maximal clique K_f of G corresponding to a leaf $f \in T^c$ is precisely B_f.*

If one wishes to build K_f in extension, it can be done by parsing all the sets X_u and Y_u of the nodes u encountered on the path between f and the root of T^c. This takes $O(\sum_{u \in \text{Anc}(f)}(|X_u| + |Y_u|)) = O(n)$ time. Note that actually, in our algorithm, we never need to do so. We only need to make some computation, for each node u of T^c, which is linear in the sizes of sets X_u, Y_u and $\mathcal{C}(u)$. This is a major interest of the PQ-representation, and this is what allows us to achieve an $O(n)$ total computation time for our dynamic algorithm.

As we mentioned previously, the fact that the vertices of G are stored in the internal nodes of the tree rather than in the leaves result in an $O(n)$ space representation, instead of $O(n+m)$

for the classical PQ-tree. Indeed, since any internal node of T^{c} has at least two children, it follows that the total number of nodes in T^{c} is $O(|\mathcal{K}(G)|) = O(n)$. And since each vertex of G has at most three pointers toward the nodes of T^{c}, and is stored in at most four subsets of vertices associated to the nodes of T^{c}, it follows that the total size of the PQ-representation is $O(n)$.

Remark 3 on the equality between the set of vertices in B_f and the maximal clique K_f corresponding to a leaf f of T^{c} can be actually generalised to any node u of the tree (i.e. not necessarily a leaf). This is what is stated by Remark 4 below, and this constitutes a key structural property of the PQ-representation that we use in all the rest of the article. See examples given in the caption of Figure 2 for node u_1.

Remark 4. *For any node $u \in T^c$, we have the following identities (see Figure 2):*

- *B_u is the subset of vertices that belong to all the maximal cliques of G that are leaves of T^c_u,*
- *u^* is the subset of vertices that are involved only in the maximal cliques of G that are leaves of T^c_u,*
- *$B_u \cup u^*$ is the subset of vertices belonging to some maximal clique of G that is a leaf of T^c_u, and*
- *$X_u = B_u \cap u^*$ is the subset of vertices belonging to all the maximal cliques of G that are leaves of T^c_u and not belonging to any other maximal clique of G.*

We extend to nodes of T^{c} the vocabulary introduced for subsets in Section 2.2, to refer to the way a vertex x is adjacent to a node u of T^{c}.

Definition 4. *A node $u \in T^c$ is said to be* uniform, full, hollow, mixed or linked *with respect to a vertex $x \notin u^*$ by referring to the corresponding notion for its set u^*. In addition, if u^* is the empty set ($u^* = \varnothing$), we say that node u is* empty *and if $B_u \cup u^*$ is full, we say that u is* saturated.

In the rest of the article, the notions above are used with respect to the newly introduced vertex $x \notin G$ in the vertex incremental algorithm.

Remark 5. *Note the difference between hollow and empty: node u is hollow when $u^* \cap N(x) = \varnothing$ and empty when $u^* = \varnothing$. In particular, when u is empty, then u is both full and hollow.*

Finally, note that, as the modular decomposition, the PQ-representation offers a convenient way to determine whether two vertices x and y of G are adjacent or not.

Remark 6. *As well as the two other representations we use, $PQ(G)$ allows to determine whether two vertices x and y are adjacent. Indeed, x and y are adjacent iff they appear in a same maximal clique. That is, one of e_x, e_y is a strict ancestor of the other, or $e_x = e_y$ and either e_x is a degenerate node or e_x is prime and the intervals $[\![e^1_x, e^2_x]\!]$ and $[\![e^1_y, e^2_y]\!]$ intersect.*

3.1.3. The MD-representation

The MD-representation of an arbitrary graph G, denoted $MD(G)$, is its modular decomposition tree T^{m} along with the quotient graphs G_p of its prime nodes p (also called *representative graphs*) and a one-to-one mapping from the vertices of G_p onto $\mathcal{C}(p)$. Clearly, since any node of T^{m} has at least two children and since T^{m} has n leaves, the number of nodes in T^{m} is $O(n)$. In the case of an interval graph G, since any quotient graph G_p is an induced subgraph of G and since the family of interval graphs is hereditary, the quotients of the prime nodes are interval graphs. Therefore, we store them by minimal interval models (see Figure 1), and as the model of a prime node p takes $O(|\mathcal{C}(p)|)$ space, the MD-representation of an interval graph G is an $O(n)$ space representation of G.

3.2. Linear time equivalence of PQ-representation and MD-representation

It has often been noticed that the PQ-tree and the modular decomposition tree of interval graphs are closely related (see e.g. [21]). Nevertheless, the complete links between the two structures have never been made explicit. In this section, we make these links explicit, and we show that in addition to their mathematical equivalence, the PQ-representation and the MD-representation of interval graphs are also algorithmically equivalent: given one of the two structures, it is possible to compute the other in linear time in the size of the input structure, i.e. $O(n)$ time in this case.

Then, in the rest of the article, we will use the equivalence of the PQ-representation and the MD-representation in order to work with the most convenient representation at each step of the algorithm, and save much effort by using known results on the update of the modular decomposition under vertex insertion [36, 24].

3.2.1. Modules in the PQ-representation

The key of the equivalence between the two structures is the fact that, as we show below, the PQ-representation of an interval graph is quite well structured regarding to its strong modules. Note that [44] studied the relations between PQ-trees and modules in context of $(0,1)$-matrices, but their results are not equivalent to ours. The aim of this section is to establish Theorem 4 and 5, which localise the modules and the strong modules of a graph in its PQ-representation. We first state some basic properties of the PQ-representation that we need in the following.

Lemmas 3 and 4 give some properties of the pointers of the vertices in Y_u, for a prime node u of T^{c}. These properties capture some fundamental constraints imposed by the structure of a prime node: no other order than σ_u and $\bar{\sigma}_u$ on the children of u may result in a consecutive ordering of the maximal cliques of the graph.

Lemma 3. *Let u be a prime node of T^c. For any $v \in \mathcal{C}(u)$, we have $\Delta_v \neq \varnothing$.*

Proof. Otherwise, we could move v both at the beginning and at the end of σ_u, without changing the relative order of the other children of u. This would results in two different consecutive orderings of G, at least one of which is not possible to obtain from a regular solidification of the PQ-tree, which contradicts the definition of the PQ-tree. □

Lemma 4. *Let u be a prime node of T^c and let I be an interval of σ_u such that $1 < |I| < |\mathcal{C}(u)|$. There exists $y \in Y_u$ such that $C_\Delta(y)$ overlaps I.*

Proof. Otherwise, we could reverse the order of the elements of I in $\mathcal{C}(u)$ without failing to satisfy any consecutiveness constraint. Which gives a contradiction with the definition of the

PQ-tree, since $|I| > 1$ and $I \neq \mathcal{C}(u)$. $\square$

Lemmas 5 and 6 give some other constraints on the pointers on the nodes of T^{c}, involving both degenerate and prime nodes and coming from the fact that no maximal clique of G is included in any other.

Lemma 5. *If u is a degenerate node or a leaf of T^c, different from the root, and such that $v = parent(u)$ is a degenerate node, then $X_u \neq \varnothing$.*

Proof. Suppose for contradiction that $X_u = \varnothing$. Then we have $B_u = B_v$. Consider a sibling u_s of u in T^{c}, and distinguish two cases.

- If u is a leaf, its corresponding maximal clique K_u satisfies $K_u = B_u = B_v$. Then, $K_u = B_v$ is included in all the maximal cliques of G that are leaves of $T^{\mathrm{c}}_{u_s}$, which is clearly a contradiction.

- If u is a degenerate node, let σ be a consecutive ordering of G and let u_1, u_2 be respectively the first and second child of u in the solidification of T^{c} defining σ. From Remark 4, the vertices involved both in some clique that is a leaf of $T^{\mathrm{c}}_{u_1}$ and in some clique that is a leaf of $T^{\mathrm{c}}_{u_2}$ belong to $B_{u_1} \cap B_{u_2} = B_u$, since u is degenerate. And since $B_u = B_v$, these vertices also appear in all the cliques that are leaves of $T^{\mathrm{c}}_{u_s}$. On the other hand, the vertices involved in some clique that is a leaf of $T^{\mathrm{c}}_{u_s}$ either appear only in some of these cliques, that is belong to u_s^*, or belong to $B_{u_s} \setminus X_{u_s} = B_v$ (see Remark 4) and then appear in all the cliques that are leaves of $T^{\mathrm{c}}_{u_1}$ and $T^{\mathrm{c}}_{u_2}$. Consequently, we can move all the cliques that are leaves of $T^{\mathrm{c}}_{u_s}$ between the cliques that are leaves of $T^{\mathrm{c}}_{u_1}$ and those that are leaves of $T^{\mathrm{c}}_{u_2}$ without failing to satisfy any consecutiveness constraint. This is a contradiction with the definition of T^{c}.

$\square$

Lemma 6. *Let u be a prime node of T^c. For any $v \in \mathcal{C}(u)$, $v^* \neq \varnothing$ or we have both there exists $y \in Y_u$ such that $e^2_y = v$ and there exists $y \in Y_u$ such that $e^1_y = v$.*

Proof. If $\{y \in Y_u \mid v = \mathrm{e}^2_y\} = \varnothing$ or $\{y \in Y_u \mid v = \mathrm{e}^1_y\} = \varnothing$, up to reversing σ_u, we can assume without loss of generality that $\{y \in Y_u \mid v = \mathrm{e}^2_y\} = \varnothing$. Let us first notice that since $\Delta_{\max(\sigma_u)} \neq \varnothing$, then there exists $y \in Y_u$ such that $\mathrm{e}^2_y = \max(\sigma_u)$. It follows that $v \neq \max(\sigma_u)$. Then, let us denote v_+ for the child of u immediately following v in σ_u. Since $\{y \in Y_u \mid v = \mathrm{e}^2_y\} = \varnothing$, then we have $\Delta_v \subseteq \Delta_{v_+}$, and therefore $B_v \setminus X_v \subseteq B_{v_+} \setminus X_{v_+}$.

Suppose for contradiction that $v^* = \varnothing$. Then, T^{c}_v does not contain any prime node v_1, since from Lemma 3, $v_1^* \neq \varnothing$ for such nodes. Furthermore, v is necessarily a leaf. Otherwise, if v was not a leaf, then, since T^{c}_v does not contain any prime node, there would exist some leaf f in T^{c}_v whose parent is a degenerate node. And from Lemma 5, f^*, as well as v^*, would not be empty. Consequently, since v is a leaf and since we have both $B_v \setminus X_v \subseteq B_{v_+} \setminus X_{v_+}$ and $v^* = \varnothing$, it follows that the maximal clique B_v corresponding to the leaf v is included in all the maximal cliques of G that are leaves of $T^{\mathrm{c}}_{v_+}$, which is a contradiction. Thus $v^* \neq \varnothing$. $\square$

As mentioned in Section 2.2, the connectivity and co-connectivity of a graph determine the type of the root of its modular decomposition tree. As a first step toward characterising the modules of a graph G on its PQ-representation, Lemma 7 gives some properties of connectivity and co-connectivity of the graphs induced by the set of vertices associated to the nodes of $T^{\mathrm{c}}(G)$.

Lemma 7. *[21]Let G be an interval graph and let u be a node of $T^c(G)$.*

1. *If u is degenerate, then $G[u^* \setminus X_u]$ is not connected; and*
2. *if u is prime, then $G[u^* \setminus X_u]$ is connected and co-connected.*

Proof.

1. Let u_1, u_2 be two distinct children of u. If u_1 is degenerate, then, from Lemma 5, $X_{u_1} \neq \varnothing$; and if u_1 is prime, Lemma 3 implies that $Y_{u_1} \neq \varnothing$. In both cases, $u_1^* \neq \varnothing$. Similarly, we have $u_2^* \neq \varnothing$. Since there is no vertex of u_1^* involved in any maximal clique containing some vertex of u_2^*, the vertices of u_1^* are not adjacent to any vertex of u_2^*. Consequently, $G[u^* \setminus X_u]$ is not connected.
2. From Lemma 3, any vertex of $u^* \setminus (X_u \cup Y_u)$ is adjacent to some vertex of Y_u. As we will show that $G[Y_u]$ is connected, then $G[u^* \setminus X_u]$ is connected. Let $y \in Y_u$ and let S_y be its connected component in $G[Y_u]$. It is well known, and easy to prove, that $I = \bigcup_{z \in S_y} C_\Delta(z)$ is an interval of σ_u. Suppose that $I \neq \mathcal{C}(u)$, then, from Lemma 4, there exists $y_1 \in Y_u$ such that $C_\Delta(y_1)$ overlaps I. It follows that $y_1 \in S_y$, which is a contradiction. Then, $I = \mathcal{C}(u)$. This implies that every $z \in Y_u$ is adjacent to some vertex of S_y. Thus, $Y_u = S_y$ is connected in G.
   In order to show that $G[u^* \setminus X_u]$ is co-connected, let us first notice that if u_1 and u_2 are two children of u such that $u_1^* \neq \varnothing$ and $u_2^* \neq \varnothing$, then all the vertices of u_1^* are not adjacent to any vertex of u_2^*. Therefore, $G[u^* \setminus (X_u \cup Y_u)]$ is co-connected. Furthermore, since, from Lemma 2, we have that for all $y \in Y_u, \min(\sigma_u) \notin C_\Delta(y)$ or $\max(\sigma_u) \notin C_\Delta(y)$ and since $\min(\sigma_u)^* \neq \varnothing$ and $\max(\sigma_u)^* \neq \varnothing$ (see Lemma 6), it follows that any $y \in Y_u$ is not adjacent to some vertex of $u^* \setminus (X_u \cup Y_u)$. Thus, $G[u^* \setminus X_u]$ is co-connected.

□

We are now ready to establish Theorems 4 and 5, which show how to determine respectively modules and strong modules on the PQ-representation.

Theorem 4. *Let G be an interval graph and let T^c be its PQ-tree. M is a module of G iff there exists a node $u \in T^c$ satisfying one of the three following conditions:*

1. *$M \subseteq X_u$ or $u^* \setminus X_u \subseteq M \subseteq u^*$; or*
2. *u is prime and there exists $u_1, u_2 \in \mathcal{C}(u)$ such that $M \subseteq \{y \in Y_u \mid e_y^1 = u_1$ and $e_y^2 = u_2\}$; or*
3. *u is degenerate and there exist $k \in [\![2, |\mathcal{C}(u)| - 1]\!]$ and $u_1, \ldots, u_k \in \mathcal{C}(u)$ such that $M = \bigcup_{1 \leq i \leq k} u_i^*$.*

Proof. $\Longleftarrow$. If all the vertices of M have the same primary and secondary pointers toward T^c, then their closed neighbourhoods are equal and it follows that M is a module. Therefore, if M satisfies Condition 2 or if $M \subseteq X_u$ (first part of Condition 1), then M is a module of G.

If $u^* \setminus X_u \subseteq M \subseteq u^*$ (second part of Condition 1), consider a vertex $y \in V(G) \setminus M$. Since $u^* \setminus X_u \subseteq M$, $e_y \notin \mathrm{Desc}(u)$. Then, either $e_y \in \mathrm{Anc}(u)$ and y is adjacent to all the vertices of M, or $e_y \notin \mathrm{Anc}(u) \cup \mathrm{Desc}(u)$ and y is adjacent to none of the vertices of M. It follows that M is a module.

If M satisfies Condition 3, consider a vertex $y \in V(G) \setminus M$. Since $M = \bigcup_{1 \leq i \leq k} u_i^*$, $e_y \notin \bigcup_{1 \leq i \leq k} \mathrm{Desc}(u_i)$. Then, either $e_y \in \mathrm{Anc}(u)$ and y is adjacent to all the vertices of M, or $e_y \notin \mathrm{Anc}(u) \cup \bigcup_{1 \leq i \leq k} \mathrm{Desc}(u_i)$ and then, since u is degenerate, y is adjacent to none of the vertices of M. Thus, M is a module.

$\Longrightarrow$. Let M be a module of G, and let u be the least common ancestor of the subset of nodes $\{e_x \mid x \in M\}$. We have the inclusion $M \subseteq u^*$. If $M \subseteq X_u$ or $u^* \setminus X_u \subseteq M$, then M satisfies condition 1. Therefore, from now and until the end of the proof, we consider the case **(*)** where $M \not\subseteq X_u$ and $u^* \setminus X_u \not\subseteq M$. In this case, there exist $x, y \in u^* \setminus X_u$ such that $x \in M$ and $y \notin M$.

From Lemma 7, $G[u^* \setminus X_u]$ is co-connected. Then, there exists $z_1 \in (u^* \setminus X_u) \cap M$ and $z_2 \in (u^* \setminus X_u) \setminus M$ such that z_1 and z_2 are not adjacent. Consequently, since M is a module, z_2 is not adjacent to any vertex of M. And since z_2 is adjacent to all the vertices of X_u, necessarily $M \cap X_u = \varnothing$.

Let us distinguish two main cases.

- u is degenerate. By definition of the least common ancestor, there are at least two children u_1, u_2 of u such that $u_1^* \cap M \neq \varnothing$ and $u_2^* \cap M \neq \varnothing$. Let v be a child of u such that $v^* \cap M \neq \varnothing$. Suppose for contradiction that $v^* \setminus M \neq \varnothing$. If v is prime, $G[v^*]$ is connected (cf. Lemma 7); and if v is degenerate then, from Lemma 5, we have $X_v \neq \varnothing$, which implies that $G[v^*]$ is connected. Then, there exist $x_v \in v^* \cap M$ and $y_v \in v^* \setminus M$ such that x_v and y_v are adjacent. Since there exists $v_2 \in \mathcal{C}(u) \setminus \{v\}$ such that $v_2^* \cap M \neq \varnothing$, and since $y_v \in v^*$ is not adjacent to any vertex of v_2^*, y_v is not adjacent to some vertex of M. This is a contradiction with the fact that y_v is adjacent to $x_v \in M$ and M is a module. Thus $v^* \subseteq M$, for all $v \in \mathcal{C}(u)$ such that $v^* \cap M \neq \varnothing$. Moreover, since $u^* \setminus X_u \not\subseteq M$ (remember that we are in Case **(*)**), M satisfies Condition 3.

- u is prime. Let us first show that the set S of children of u covered by at least one vertex of M is an interval of σ_u and is such that $1 < |S| < |\mathcal{C}(u)|$.
  First, suppose for contradiction that S is not an interval of σ_u. Then, there exist two children $u_a, u_b \in S$ of u such that $u_a <_{\sigma_u} u_b$ and u_a and u_b are not consecutive in σ_u and the subset I of children of u that are between u_a and u_b in σ_u do not contain any node of S. By definition, I is an interval of σ_u different from $\mathcal{C}(u)$. Then, from Lemma 4, there exists $y \in Y_u$ such that $C_\Delta(y)$ overlaps I. Then, $y \notin M$ is adjacent to the vertices of M that cover one of u_a, u_b but is not adjacent to the vertices of M covering the other one: this is a contradiction with the fact that M is a module. Thus, S is an interval of σ_u.

  Suppose now, for contradiction again, that $Y_u \cap M = \varnothing$, then by definition of the least common ancestor, there are at least two children u_1, u_2 of u such that $u_1^* \cap M \neq \varnothing$ and $u_2^* \cap M \neq \varnothing$. If $\{u_1, u_2\} \neq \{\min(\sigma_u), \max(\sigma_u)\}$ then, from Lemma 4, there exists $y \in Y_u$ such that $C_\Delta(y)$ overlaps $[\![u_1, u_2]\!]$. Otherwise, if $\{u_1, u_2\} = \{\min(\sigma_u), \max(\sigma_u)\}$, then let $y \in Y_u$ such that $\mathrm{e}_y^1 = \min(\sigma_u)$; by definition, $\mathrm{e}_y^2 \neq \max(\sigma_u)$. In both cases, exactly one of u_1, u_2 is covered by y, while the other is not. Thus, $y \notin M$ is adjacent to some vertex of M and is not adjacent to some other vertex of M, which is a contradiction. Thus, $Y_u \cap M \neq \varnothing$ and it follows that $|S| > 1$.

  Suppose now that every child of u is covered by at least one vertex of M. Then, any vertex of $(u^* \setminus X_u) \setminus M$ is adjacent to some vertex of $M \cap Y_u$. And since M is a module, all the vertices of $(u^* \setminus X_u) \setminus M$ (which is not empty since $u^* \setminus X_u \not\subseteq M$, see Case **(*)**) are adjacent to all the vertices of $M \cap (u^* \setminus X_u)$ (which is not empty since $Y_u \cap M \neq \varnothing$, see the conclusion of the previous paragraph, and since $Y_u \subseteq u^* \setminus X_u$ by definition). This contradicts the co-connectivity of $G[u^* \setminus X_u]$, stated by Lemma 7. Then, we also have $|S| < |\mathcal{C}(u)|$.

  Let us now show that $M \cap (u^* \setminus (X_u \cup Y_u)) = \varnothing$. We denote u_f, u_l for respectively the first and last element of S in σ_u.

- If $u_f^* = \varnothing$ and $u_l^* = \varnothing$. Then, from Lemma 6, there exist $y_f, y_l \in Y_u$ such that $\mathrm{e}^2_{y_f} = u_f$ and $\mathrm{e}^1_{y_l} = u_l$. It follows that $y_f, y_l \notin M$ and both y_f and y_l are adjacent to some vertex of M. Then, since M is a module, y_f and y_l are adjacent to all the vertices of M. Since there is no vertex of $u^* \setminus (X_u \cup Y_u)$ which is adjacent to both y_f and y_l, it follows that $M \cap (u^* \setminus (X_u \cup Y_u)) = \varnothing$.
- We now examine the case where $u_f^* \neq \varnothing$ or $u_l^* \neq \varnothing$.
  * If $u_f^* = \varnothing$ or $u_l^* = \varnothing$, then, up to reversing σ_u, we can assume without loss of generality that $u_f^* \neq \varnothing$ and $u_l^* = \varnothing$. Then, we denote y for a vertex in Y_u such that $\mathrm{e}^1_y = u_l$ (cf. Lemma 6). Vertex y is such that $u_f \notin C_\Delta(y)$.
  * If $u_f^* \neq \varnothing$ and $u_l^* \neq \varnothing$, then, since S is an interval of σ_u and $|S| < |\mathcal{C}(u)|$, we denote y for a vertex of Y_u such that $\mathcal{C}_\Delta(y)$ overlaps $[\![u_f, u_l]\!]$ (cf. Lemma 4), and up to reversing σ_u, we can assume without loss of generality that $u_f \notin C_\Delta(y)$.

  In the two sub-cases above, $y \notin M$ is adjacent to some vertex of M and to none of the vertices of $u_f^* \neq \varnothing$, since $u_f \notin C_\Delta(y)$. It follows that $u_f^* \cap M = \varnothing$. Let $a \in u_f^*$ and u_{ne} a child of u such that $u_{ne}^* \neq \varnothing$. Since $a \notin M$ is adjacent to some vertex of M (by definition of S) and since M is a module, a is adjacent to all the vertices of M. Since a is adjacent to none of the vertices of u_{ne}^*, it follows that $u_{ne}^* \cap M = \varnothing$. As it holds for any $u_{ne} \in \mathcal{C}(u) \setminus \{u_f\}$, we have $M \cap (u^* \setminus (X_u \cup Y_u)) = \varnothing$.

In order to achieve the proof, we have to show that the vertices of M have the same secondary pointers. If $u_l^* = \varnothing$, then we denote y for a vertex of Y_u such that $\mathrm{e}^1_y = u_l$ (cf. Lemma 6); otherwise, we denote y for some vertex of u_l^*. In both cases, $y \notin M$ and y is adjacent to some vertex of M. Furthermore, $N(y) \cap M = \{z \in M \mid \mathrm{e}^2_z = u_l\}$. Consequently, as M is a module, every $z \in M$ is such that $\mathrm{e}^2_z = u_l$. Similarly, by reversing σ_u, we show that every $z \in M$ satisfies $\mathrm{e}^1_z = u_f$. Thus, M satisfies Condition 2.

□

Once modules are identified, it is easy to do so for strong modules, which are the nodes of the modular decomposition tree.

Theorem 5. *Let G be an interval graph. M is a non trivial strong module of G iff $1 < |M| < n$ and there exists some node $u \in T^c(G)$ satisfying one of the two following conditions:*

1. *$M = u^*$ or $M = u^* \setminus X_u$; or*
2. *u is a prime node and there exist $u_1, u_2 \in \mathcal{C}(u)$ such that $M = \{y \in Y_u \mid e^1_y = u_1$ and $e^2_y = u_2\}$.*

Proof.

We just have to determine which modules given by Theorem 4 overlap with others, and which do not. If M is a non trivial module ($|M| \geq 2$) such that $M \subseteq X_u$ for some node $u \in T^c$, then $(u^* \setminus X_u) \cup \{a\}$, where $a \in M$, overlaps M (note that $(u^* \setminus X_u) \neq \varnothing$ from Lemmas 3 and 5). Conversely, if $u^* \setminus X_u \subsetneq M \subsetneq u^*$, then X_u overlaps M. On the other hand, if $M = u^* \setminus X_u$ or $M = u^*$, no other module overlaps M.

If M satisfies Condition 3 of Theorem 4, then $\bigcup_{v \in \mathcal{C}(u) \setminus \{u_2, \ldots, u_k\}} v^*$ overlaps M.

Finally, if $M \subsetneq \{y \in Y_u \mid \mathrm{e}^1_y = u_1$ and $\mathrm{e}^2_y = u_2\}$ for some prime node $u \in T^c$ and two children u_1, u_2 of u, then $(\{y \in Y_u \mid \mathrm{e}^1_y = u_1$ and $\mathrm{e}^2_y = u_2\} \setminus M) \cup \{a\}$, where $a \in M$, overlaps M. On the opposite, if $M = \{y \in Y_u \mid \mathrm{e}^1_y = u_1$ and $\mathrm{e}^2_y = u_2\}$, no other module overlaps M.

□

As a first step toward the equivalence between the MD-representation and the PQ-representation, we are now able to introduce a formal correspondence between some nodes of the MD-representation and the nodes of the PQ-representation, based on their associated sets of vertices. This correspondence will be of great help in the vertex insertion algorithm where we use simultaneously the two structures.

From Theorem 5, for all nodes $u \in T^{\mathrm{c}}$, there exists $u' \in T^{\mathrm{m}}$ such that $V(u') = u^* \setminus X_u$. Then, the following application ξ is properly defined.

$$\begin{array}{rccl} \xi\colon & T^{\mathrm{c}} & \to & T^{\mathrm{m}} \\ & u & \mapsto & \xi(u) = u' \text{ such that } V(u') = u^* \setminus X_u \end{array}$$

Lemma 8. *Application ξ is a one-to-one mapping from the internal nodes of T^c to the set of parallel nodes and prime nodes of T^m.*

Proof. Since, from Lemmas 3 and 5, $|u^* \setminus X_u| \geq 2$ for all internal nodes u of T^{c}, then $\xi(u)$ is an internal node of T^{m}. In addition, from Lemma 7, either $G[u^* \setminus X_u]$ is not connected (when u is degenerate), and then $\xi(u)$ is a parallel node of T^{m}, or $G[u^* \setminus X_u]$ is connected and co-connected (when u is prime), and then $\xi(u)$ is a prime node of T^{m}. We also have that ξ is injective since all the internal nodes u of T^{c} have pairwise distinct sets $u^* \setminus X_u$ (from Lemmas 3 and 5). And finally, ξ is surjective since Theorem 5 gives all the non trivial strong modules M of G and the only case where $G[M]$ is co-connected (which is precisely the case for parallel nodes and prime nodes of T^{m}) is the case where $M = u^* \setminus X_u$ for some $u \in T^{\mathrm{c}}$. □

This allows us to use the following vocabulary in the rest of the paper.

Definition 5. *For any node $u \in T^c$, u and $\xi(u)$ are called* associate nodes.

As mentioned above, we use associate nodes in the first part of our dynamic algorithm in order to simplify its presentation: we will be able to get for free some information about nodes of T^{c} thanks to the information computed for their associate nodes in T^{m}. But beyond this technical point, the fact that ξ is a one-to-one mapping implies that the internal part of the PQ-tree of a graph is the internal part of its modular decomposition tree where the series nodes have been removed. As we will see in the next two sections, some of the series nodes of T^{m} simply disappear from the tree, while some others become leaves of the PQ-tree.

3.2.2. From the PQ-representation to the MD-representation

We now exhibit an application ϕ from the set of PQ-representations of interval graphs to the set of MD-representations of interval graphs, which we call the PQ-MD-transformation. We define ϕ as a bottom-up process that builds the tree $T^{\mathrm{m}}(G[u^*])$ for each node u of $T^{\mathrm{c}}(G)$. The process ends with the computation of $\phi(T^{\mathrm{c}}(G)) = T^{\mathrm{m}}(G[r^*]) = T^{\mathrm{m}}(G)$, where r is the root of $T^{\mathrm{c}}(G)$. We first give the description of the process defining ϕ, then we prove, thanks to Theorem 5, that $\phi(T^{\mathrm{c}}(G)) = T^{\mathrm{m}}(G)$, and finally we show that ϕ can be computed in $O(n)$ time.

The process starts with the computation of $T^{\mathrm{m}}(G[l^*])$ for each leaf l of $T^{\mathrm{c}}(G)$ st. $l^* \neq \varnothing$. Since l^* is a clique, $T^{\mathrm{m}}(G[l^*])$ has one series node (or none if $|l^*| = 1$) having a leaf child for each vertex of l^*. Then, we inductively build $T^{\mathrm{m}}(G[u^*])$ for any internal node u of $T^{\mathrm{c}}(G)$, denoting $\{u_1, \ldots, u_k\}$ the set of children of u in T^{c}. We have to consider three different cases.

1. *Creation of a series node.* If $X_u \neq \varnothing$ (Figure 3), we first ignore the vertices of X_u and build $T^{\mathrm{m}}(G[u^* \setminus X_u])$ like in the cases where $X_u = \varnothing$, described below. Then, we introduce a new series node whose children are the leaves representing vertices of X_u and the root of $T^{\mathrm{m}}(G[u^* \setminus X_u])$.

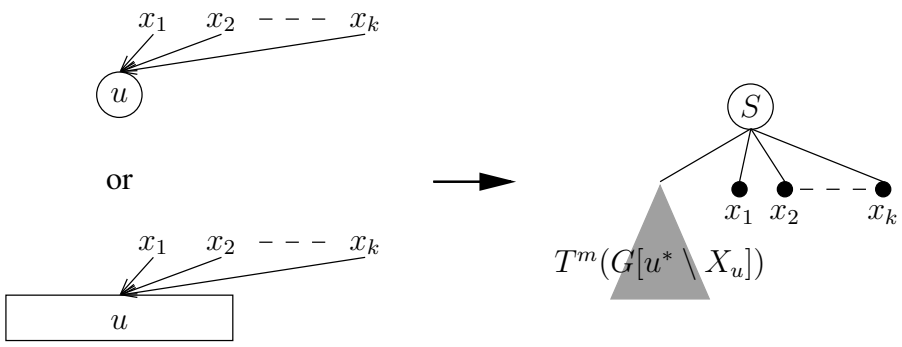

Figure 3: From $PQ(G)$ to $MD(G)$: dealing with a node u such that $X_u \neq \varnothing$.

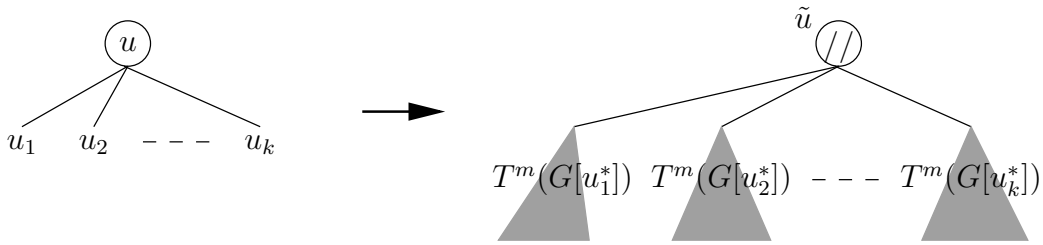

Figure 4: From $PQ(G)$ to $MD(G)$: dealing with a degenerate node u such that $X_u = \varnothing$.

2. *Creation of a parallel node.* If $X_u = \varnothing$ and u is a degenerate node (Figure 4), the root $\tilde{u}$ of $T^{\mathrm{m}}(G[u^*])$ is a parallel node whose children are the roots of the trees $\{T^{\mathrm{m}}(G[u_1^*]), \ldots, T^{\mathrm{m}}(G[u_k^*)]\}$.
3. *Creation of a prime node.* If $X_u = \varnothing$ and u is a prime node (Figure 5). The root $\tilde{u}$ of $T^{\mathrm{m}}(G[u^*])$ is a prime node. The interval model of $G_{\tilde{u}}$ is made with the list Z of children of u ordered by σ_u. The set of simplicial vertices of $G_{\tilde{u}}$ is the set $\mathcal{S} = \{v \in \mathcal{C}(u) \mid v^* \neq \varnothing\}$. For any $v \in \mathcal{S}$, the root of $T^{\mathrm{m}}(G[v^*])$ is made a child of $\tilde{u}$ and its corresponding vertex in $G_{\tilde{u}}$ has two pointers toward the cell of v in Z. The non simplicial vertices of $G_{\tilde{u}}$ are the classes $\bar{y}$ of vertices $y \in Y_u$ having the same secondary pointers. The child $\tilde{v}$ of $\tilde{u}$ corresponding to $\bar{y}$ is a series node (or a leaf if $|\bar{y}| = 1$) whose children are the vertices of $\bar{y}$. The pointers of $\tilde{v}$ toward Z are the same as the pointers of any $y \in \bar{y}$.

Let us check that the process described above correctly computes $T^{\mathrm{m}}(G[u^*])$ for each node u of T^{c}. To that purpose, we need to identify the maximal strong modules of $G[u^*]$, which can be done on $T^{\mathrm{c}}(G[u^*])$ thanks to Theorem 5. This is a well known fact that $T^{\mathrm{c}}(G[u^*]) = T^{\mathrm{c}}_u(G)$. Thus, we have to check that in each of the three cases above, the children assigned to the new node u created in the MD-representation are the maximal strong modules given by Theorem 5 applied on $T^{\mathrm{c}}_u(G)$.

1. If $X_u \neq \varnothing$, then from Theorem 5 there is only one non-trivial maximal strong module, which is $u^* \setminus X_u$. This implies that the other maximal strong modules, those containing the vertices of X_u, are trivial : they are singletons. Since all the vertices of X_u are adjacent to all the vertices of $u^* \setminus X_u$, it follows that the MD-representation built in this case is correct.
2. If $X_u = \varnothing$ and u is a degenerate node, then from Theorem 5 the maximal strong modules of $G[u^*]$ are the sets $u_1^*, \ldots, u_k^*$. Since these sets of nodes are all pairwise disconnected, the MD-representation built is correct.

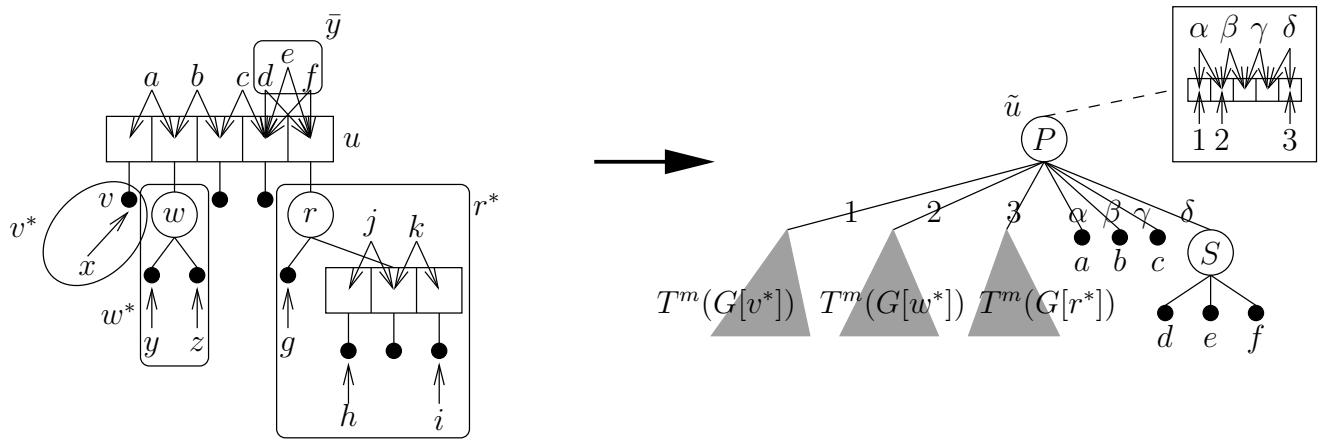

Figure 5: From $PQ(G)$ to $MD(G)$: dealing with a prime node u such that $X_u = \varnothing$.

3. If $X_u = \varnothing$ and u is a prime node, then Theorem 5 gives that the maximal strong modules are the classes $\bar{y}$ of vertices $y \in Y_u$ having the same secondary pointers and the sets $v^* \neq \varnothing$, where v is a child of u. Replacing each of these sets by a single vertex indeed result in the quotient graph whose interval model is described in Case 3 above. Since this quotient graph is obviously not a clique nor a stable, the well known modular decomposition theorem (see Section 2.2) implies that it is prime. Finally, since all the vertices of a class $\bar{y}$ form a clique, the node corresponding to $\bar{y}$ in the MD-representation is a series node having only leaf children (or simply one leaf if $|\bar{y}| = 1$).

This proves that the process defining ϕ correctly builds $T^{\mathrm{m}}(G[u^*])$ at each step and that, consequently, it ends with the computation of $T^{\mathrm{m}}(G[r^*]) = T^{\mathrm{m}}(G)$, where r is the root of $T^{\mathrm{c}}(G)$. Thus, for each PQ-tree $T^{\mathrm{c}}(G)$ of an interval graph G, the image of $T^{\mathrm{c}}(G)$ by ϕ is the modular decomposition tree $T^{\mathrm{m}}(G)$ of G: $\phi(T^{\mathrm{c}}(G)) = T^{\mathrm{m}}(G)$. And since the PQ-tree and the modular decomposition tree of an interval graph are both unique, it follows that ϕ is a bijection from the set of PQ-representations of interval graphs to the set of MD-representations of interval graphs.

From the validity of the process defining ϕ, we immediately derive the following theorem, which we need in the next section to define the converse transformation process : from the MD-representation to the PQ-representation. We do not prove formally Theorem 6 since it is clear that the rules of the process defining ϕ imply the following structure on the MD-representation produced.

Theorem 6. *The MD-representation of an interval graph $G = (V, E)$ satisfies the three following properties.*

1. *every series node of $T^m(G)$ has at most one non-leaf child; and*
2. *every prime node u of $T^m(G)$ is such that the children of u corresponding to non-simplicial vertices of G_u are leaves or series node whose children are leaves; and*
3. *every prime node u of $T^m(G)$ is such that G_u is an interval graph.*

Let us now prove that the process defining ϕ can be implemented in $O(n)$ time. Clearly, the initial processing of the leaves of $T^{\mathrm{c}}(G)$ takes $O(n)$ time. For a node u st. $X_u \neq \varnothing$, Treatment 1 takes $O(|X_u|)$ time. For a degenerate node u, Treatment 2 takes $O(|\mathcal{C}(u)|)$ time. In the processing of a prime node, the difficult operation is to find the equivalence classes $\bar{y}$ in Y_u. To that purpose, we can bucket sort the vertices $y \in Y_u$ with the rank of e_y^1 in Z as primary

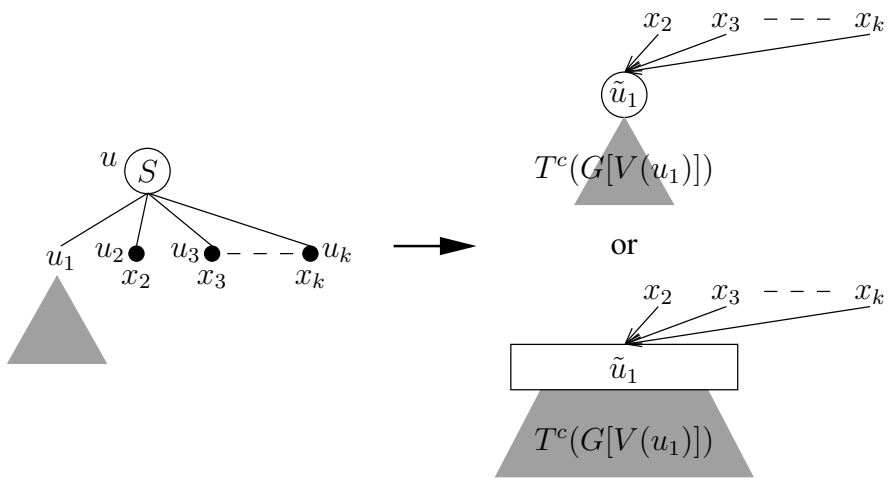

Figure 6: From $MD(G)$ to $PQ(G)$: dealing with a series node u.

key, and the rank of e_y^2 as secondary key. As we sort $|Y_u|$ elements having values between 1 and $|\mathcal{C}(u)|$, this takes $O(|Y_u| + |\mathcal{C}(u)|)$ time. It follows that the processing time of a prime node is $O(|Y_u| + |\mathcal{C}(u)|)$. Thus, the total computation time of $T^{\mathrm{m}}(G)$ is $O(\sum_{u \in T^{\mathrm{c}}} |X_u| + \sum_{u \in T^{\mathrm{c}}} |Y_u| + \sum_{u \in T^{\mathrm{c}}} |\mathcal{C}(u)|) = O(n)$.

3.2.3. From the MD-representation to the PQ-representation

We now exhibit the converse application ψ of ϕ from the set of MD-representations of interval graphs to the set of PQ-representations of interval graphs, which we call the MD-PQ-transformation. As previously, we describe the image of ψ as a result of a bottom-up process that computes the PQ-representation of an interval graph G, whose MD-representation is given, by inductively computing $T^{\mathrm{c}}(G[V(u)])$ for all nodes u of $T^{\mathrm{m}}(G)$ as follows.

For any leaf l_x of $T^{\mathrm{m}}(G)$ corresponding to vertex $x \in V(G)$, $T^{\mathrm{c}}(G[V(l_x)])$ has only one leaf and no internal node. The main pointer of x points to this leaf. For an internal node u of $T^{\mathrm{m}}(G)$, whose set of children is $\{u_1, \ldots, u_k\}$, we consider three different cases regarding the label of u.

1. *u is a series node* (Figure 6). From Theorem 6, u has at most one non-leaf child. If it has no non-leaf child, then $T^{\mathrm{c}}(G[V(u)])$ has only one leaf and no internal node, and the vertices of $V(u)$ point to that leaf. If u has one non-leaf child u_1, then $T^{\mathrm{c}}(G[V(u)])$ is obtained from $T^{\mathrm{c}}(G[V(u_1)])$, whose root is denoted $\tilde{u}_1$ and may be either a degenerate or a prime node, by making all the vertices $x_2, \ldots, x_k$ corresponding to the leaf children $u_2, \ldots, u_k$ of u point toward $\tilde{u}_1$.
2. *u is a parallel node* (Figure 7). In this case, the root $\tilde{u}$ of $T^{\mathrm{c}}(G[V(u)])$ is a degenerate node whose children are the roots of the trees $T^{\mathrm{c}}(G[V(u_1)]), \ldots, T^{\mathrm{c}}(G[V(u_k))]$. The pointers of vertices of $V(u)$ are unchanged.
3. *u is a prime node* (Figure 8). The root $\tilde{u}$ of $T^{\mathrm{c}}(G[V(u)])$ is a prime node. The children of $\tilde{u}$ are in bijection with the cells of the list Z constituting the interval model of the representative graph G_u, and $\sigma_{\tilde{u}}$ is precisely the order of the cells in Z.
   For a cell c of Z whose corresponding maximal clique of G_u contains no simplicial vertex, the child of $\tilde{u}$ corresponding to c is a leaf. For a cell c of Z whose corresponding maximal

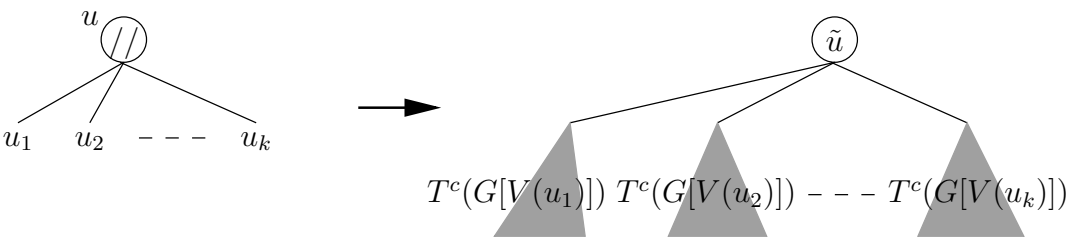

Figure 7: From $MD(G)$ to $PQ(G)$: dealing with a parallel node u.

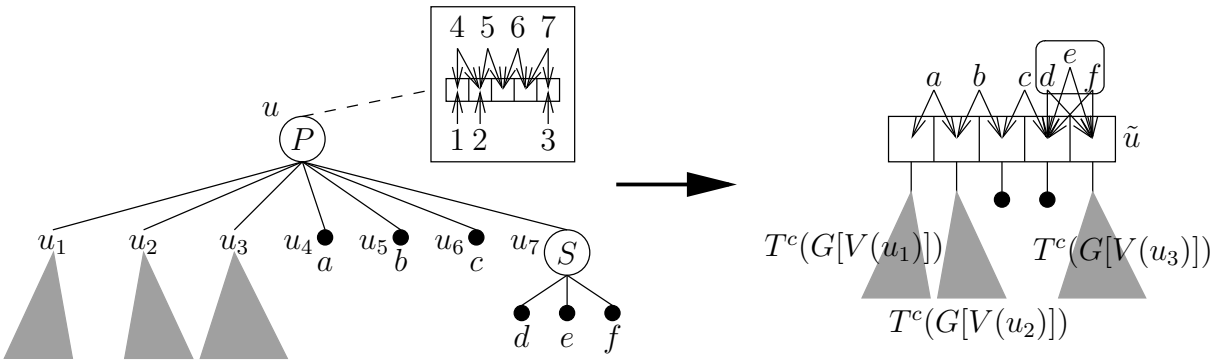

Figure 8: From $MD(G)$ to $PQ(G)$: dealing with a prime node u.

clique of G_u contains one[3] simplicial vertex x, the child of $\tilde{u}$ corresponding to c is the root of $T^{\mathrm{c}}(G[V(u_x)])$, where u_x is the child of u corresponding to x in $T^{\mathrm{m}}(G)$. The vertices of $V(u_x)$ keep their pointers toward $T^{\mathrm{c}}(G[V(u_x)])$.
Let $\mathcal{NS}$ be the set of children of u whose corresponding vertex in G_u is not simplicial. We have $Y_{\tilde{u}} = \bigcup_{u_{ns} \in \mathcal{NS}} V(u_{ns})$. For any $u_{ns} \in \mathcal{NS}$ corresponding to vertex y of G_u, all the vertices of $V(u_{ns})$ are given a main pointer toward $\tilde{u}$ and two secondary pointers toward the children of $\tilde{u}$ corresponding to the cells of Z pointed by y. Remind that, from Theorem 6, the vertices of $V(u_{ns})$ form a clique.

It is easy to check that ψ is the reciprocal bijection of ϕ since the three transformations described above are the converse transformations of those used in the definition of ϕ. Thus, for each MD-representation $T^{\mathrm{m}}(G)$ of an interval graph G, the image of $T^{\mathrm{m}}(G)$ by ψ is the PQ-representation $T^{\mathrm{c}}(G)$ of G: $\psi(T^{\mathrm{m}}(G)) = T^{\mathrm{c}}(G)$.

Let us analyse the running time of the process defining ψ. A leaf of $T^{\mathrm{m}}(G)$ needs constant time. A parallel or series node u is treated in $O(|\mathcal{C}(u)|)$ time. For a prime node u, we first build the list of its children ordered by σ_u. Initially, all the children of u are made leaves. Their number and order are given by the list Z. This operation takes $O(|Z|)$ time, where $|Z|$ is the length of Z. In order to find the cells c of Z whose corresponding maximal clique of G_u contains one simplicial vertex x, we can parse the vertices of G_u and check whether their two pointers toward Z are the same. When a vertex $y \in G_u$, whose corresponding child of u is denoted u_y, indeed points twice to the same cell c of Z, the leaf child of $\tilde{u}$ corresponding to c is replaced by the root of $T^{\mathrm{c}}(G[V(u_y)])$. Otherwise, when y has two distinct pointers toward cells c_1, c_2 of Z, we assign to the vertices of $V(u_y)$ secondary pointers toward the children of $\tilde{u}$ corresponding to

[3]Since G_u is prime, there cannot be more than one simplicial vertex in a given clique of G_u.

c_1, c_2. Again, remind that $V(u_y)$ is necessarily a clique from Theorem 6. Thus, the computing time of a prime node $u \in T^{\mathrm{m}}(G)$ is $O(|Z|+|\mathcal{C}(u)|+\sum_{y\in\mathcal{NS}(G_u)}|V(u_y)|)$, where $\mathcal{NS}(G_u)$ is the set of non-simplicial vertices of G_u. Since G_u is an interval graph on $|\mathcal{C}(u)|$ vertices, $|Z| = O(|\mathcal{C}(u)|)$, and the computation of a prime node u costs $O(|\mathcal{C}(u)|+\sum_{y\in\mathcal{NS}(G_u)}|V(u_y)|)$. Furthermore, since for any $y \in \mathcal{NS}(G_u)$, u_y is a leaf or a series node having only leaf children (Theorem 6), it follows that for any vertex $x \in G$ there exists at most one prime node u and one vertex $y \in \mathcal{NS}(G_u)$ such that x belongs to u_y. Consequently, $\sum_{u\in T^{\mathrm{m}}(G),u \text{ prime}}\sum_{y\in\mathcal{NS}(G_u)}|V(u_y)| = O(n)$. Thus, the total computation time of $T^{\mathrm{c}}(G)$ is $O(n) + O(\sum_{u\in T^{\mathrm{m}}(G)}|\mathcal{C}(u)|) = O(n)$, since the number of nodes in $T^{\mathrm{m}}(G)$ is $O(n)$.

The following remark will be useful in the first step of our algorithm : thanks to the MD-PQ-transformation, we will be able to get some information about the nodes of T^{c} that we already computed for the nodes of T^{m}.

Remark 7. *The node $\tilde{u}$ in cases 2 and 3 of the MD-PQ-transformation is precisely the associate node of u in T^c (see Definition 5). Thus, the correspondences between associate nodes of T^m and T^c can be computed without extra cost during the MD-PQ-transformation.*

As a general conclusion of Section 3.2, we obtained that the MD-representation and the PQ-representation of an interval graph are linearly equivalent structures. In other words, any of them can be retrieved from the other in linear time, that is $O(n)$ time. In particular, this implies that any algorithm taking as input one of the two structures and whose complexity is at least $O(n)$ can equivalently take as input the other structure and still achieve the same complexity. And the same remark holds when the MD-representation or the PQ-representation is the output of the algorithm rather than its input. In particular, this is the case of our dynamic algorithm which aims at producing both structures as output in $O(n)$ time. Therefore, we can focus on producing only one of them and get the other one in the same complexity thanks to the transformations ϕ and ψ described in Sections 3.2.2 and 3.2.3.

3.3. Equivalence of the PQ-representation and an interval model

In order to achieve the proof of the linear-time equivalence of the three structures we use in our algorithm, we will now establish the equivalence between the PQ-representation and an interval model. The difficult part in doing so is to show that we can compute the PQ-representation from any interval model in $O(n)$ time. This challenging task has been treated in [37] (Theorem 14). However, their algorithm is not quite sufficient to achieve our goal since it computes the PQ-tree in the form introduced by [6], and we need it in the form introduced by [21], which we call the PQ-representation. The only difference between the two structures is the set of pointers from the vertices of the graph to the nodes of the PQ-tree. Fortunately, this task is not difficult, and once the tree has been built from the interval model, the pointers from all vertices of the graphs can be computed efficiently, namely in $O(n)$ time. We note that the algorithm of [37] could be modified to do so, but for sake of clarity, we prefer to describe an independent algorithm to get these pointers afterwards. Moreover, this extends a result of [11] which shows that you can obtain the PQ-representation form the PQ-tree and the graph in $O(n + m)$ time (Theorem 7). In this section, we note that if the graph is given by one of its interval models instead of its adjacency lists, this time bound can be lowered to $O(n)$.

The input of the algorithm described in the rest of this section is an interval model σ of the interval graph G and the PQ-tree T of G, where each maximal clique has a unique ID which is an integer whose value is $O(n)$ and which is stored both in the corresponding cell of the list constituting the interval model (see Section 3.1.1) and in the corresponding leaf of the PQ-tree. In particular, this is the case in the PQ-tree being the output of the algorithm of [37]. Since the values of the IDs are $O(n)$, it is possible to compute the correspondence between the leaves

of T and the cells of σ in $O(n)$ time, using an array of size $O(n)$. Consequently, from now, we consider that each cell of σ has a pointer to its corresponding leaf in T, and that conversely, each leaf of T has a pointer to its corresponding cell in σ.

In order to compute the PQ-representation, we proceed in three steps : i) we compute the solidification π of the tree which corresponds to the interval model σ given as input, ii) we assign to each node of the tree the pointers to the first and last clique of its subtree in σ and iii) for each vertex y of G we compute the pointers from y to e_y, e_y^1 and e_y^2.

First of all, for convenience of the description of the algorithm, we start by selecting and keeping a single representative vertex for each class of equivalence of vertices having the same two pointers to the cells of σ. This can be done in $O(n)$ time by simply bucket-sorting all the vertices with their left pointer as primary key and their right pointer as secondary key. Once we will have assigned the correct pointers $e_y, \mathrm{e}_y^1, \mathrm{e}_y^2$ to the representative vertices y of all the classes of equivalence, it will be very easy to assign the same pointers to all the vertices of the considered class within $O(n)$ time overall complexity. So from now, we assume that all the vertices of G have distinct couples of pointers toward the cells of σ.

Let us now show how to achieve task i) in $O(n)$ time. For each node u, we want to compute the order $\pi(u)$ of its children corresponding to the desired solidification, namely the solidification resulting in the consecutive ordering σ given as input. We start by initialising all lists $\pi(u)$ to the empty list. Then, for each maximal clique K taken in the order defined by σ, we parse the branch of T starting from the leaf l_K corresponding to K toward the root of T and we stop when we find a node s of T that we have already encountered before. For each node u on the path from parent(l_K) to s, both included, we place the child of u we arrived from at the end of the list $\pi(u)$ of the children of u that have been discovered so far. The process ends when we have parsed all the cliques K in σ. Then, since the cliques K are considered in the order defined by σ, it is clear that the solidification of T defined by the orders $\pi(u)$ computed for each node u of T is precisely the solidification resulting in σ. Moreover, since for each clique K, we stop climbing the branch starting at l_K when we encounter a node that has already been discovered before, the complexity of task i) is dominated by the size of the tree, that is $O(n)$.

We achieve task ii) by a simple bottom-up process along the tree where each node inductively forwards to its parent the two pointers to the first and last clique of its subtree in σ. The process is initialised on the leaves of the tree, for which we already know their two (identical) pointers to σ. When a node u has received the pointers from all its children, its left pointer to σ is assigned the leftmost left pointer of its children and its right pointer is assigned the rightmost right pointer of its children. Then, node u forwards its two pointers to its parent, and so on, until the root of the tree is assigned its two pointers. This takes $O(n)$ time.

For task iii), we start by placing in each cell c of σ two sorted lists : the list of vertices whose left bound is c and the list of vertices whose right bound is c, both sorted according to inclusion order (i.e. smaller intervals appear first in the list). In order to create the lists of left bounds, one can bucket-sort all the vertices by increasing right bounds, then parse the obtained list and place each vertex in the list of the cell being its left bound. One can proceed similarly to get the lists of right bounds sorted appropriately. Thanks to these lists stored in the cells of σ, it becomes easy to assign the pointers from the vertices of G to the nodes of T. We parse σ starting from its first cell. For each cell K having a non empty list L_{left} of left bounds, we build the list L_{anc} of nodes we encounter when going up in T from the leaf l_K corresponding to K until we reach a node s which is not the first child of its parent in solidification π (i.e. s is not the first element of $\pi_{\text{parent}(s)}$). List L_{anc} includes s but not parent(s). Note that the nodes u in L_{anc} are sorted according to inclusion of the interval of cliques corresponding to u (i.e. the leaves of T_u). Also note that all the vertices whose left bound is in L_{left} are included in the interval of parent(s). We merge the two sorted lists L_{anc} and L_{left} in order to obtain the

list L_{merge} sorted according to interval inclusion. When there is equality between the intervals of a node in L_{anc} and a vertex in L_{left}, we break the tie by saying that the element of L_{anc} is less than the one in L_{left}, i.e. it appears before in L_{merge}. Moreover, we had the node parent(s) at the end of the list L_{merge}, which respects the inclusion order as noted before.

For each vertex y in L_{left}, we determine the node u_1 which is the first node of L_{anc} below y in L_{merge} and node u_2 which is the first node of L_{anc} above y in L_{merge}. If the interval of y is equal to the interval of u_1, then we set $e_y = u_1$ and y has no secondary pointers. Otherwise, we set $e_y = u_2$ and $\mathrm{e}_y^1 = u_1$. We can proceed similarly to treat the lists of right bounds and to assign e_y^2. The complexity of merging the two sorted lists L_{left} and L_{anc} is $O(|L_{\text{left}}| + |L_{\text{anc}}|)$. The computation of u_1 and u_2 can be done for all the vertices $y \in L_{\text{left}}$ of L_{merge} by two scans of L_{merge}, which takes $O(|L_{\text{merge}}) = O(|L_{\text{left}}| + |L_{\text{anc}}|)$ time. Thus, the total time needed to complete task iii) is $O(\sum_{K\in\sigma} |L_{\text{left}}| + |L_{\text{anc}}|)$. Clearly, $\sum_{K\in\sigma} |L_{\text{left}}| = n$ since each vertex appears exactly once in the lists of left bounds. Moreover, two distinct lists L_{anc} may intersect only on their last element. Thus, the set of edges of the tree T covered by two distinct lists are disjoints, and it follows that $\sum_{K\in\sigma} |L_{\text{anc}}| = O(|T|) = O(n)$. Therefore, task iii) can be accomplished in $O(n)$ time.

Finally, computing the PQ-representation from the PQ-tree and an interval model can be done in $O(n)$ time. And thus, together with the algorithm of [37], we obtain that the PQ-representation can be computed from an interval model in $O(n)$ time.

It is well known that the converse operation, that is computing an interval model from the PQ-representation, can be done in $O(n)$ time. Indeed, a consecutive ordering σ can be obtained by a simple Depth First Search on T. Then, one can execute task ii) above to get the left and right bounds of the interval formed by the cliques of T_u, for each node u in T. At last, one can assign to each vertex y of G its interval bounds in σ by taking the same bounds as e_y if y has no secondary pointers, or by taking the left bound of e_y^1 and the right bound of e_y^2 otherwise.

As a general conclusion of Section 3, we showed that the three representations we will use throughout the paper, namely interval models, PQ-representation and MD-representation, are linear-time equivalent, that is any of them can be obtained from any of the two others in $O(n)$ time. We will extensively use these results in the following. First, in our dynamic algorithm, this will allow us to maintain only one of the three representations in $O(n)$ time, and get the two others for free within the same time complexity. Second, we will use the correspondence we highlighted between the PQ-representation and the MD-representation in order to work with the more convenient representation at each step of the algorithm, see Section 5.3.

4. Edge modification and vertex deletion

Our algorithm is fully dynamic in the sense that it can handle both deletions and additions, either of an edge or of a vertex (along with the edges defining its neighbourhood). In general, in dynamic algorithms, the modification operations considered may not all have the same difficulty to be treated (see e.g. [27]). In our case, it turns out that the difficulties of treating the four operations we consider are about the same. Nevertheless, the rest of the paper concentrates on the sole addition of a vertex. The reason is that vertex modifications can simulate edge modifications, and we will rely on the results we presented in Section 3 and on the result of [37] in order to treat vertex deletions.

One should keep in mind that we maintain three representations at each step of our algorithm: the MD-representation, the PQ-representation and a minimal interval model. Then, we can use all these three representations in order to treat a given modification, but we only need to show how to maintain one of them in $O(n)$ time, since we already showed in Section 3, that we can obtain the two other representations in $O(n)$ time from any of the three representations.

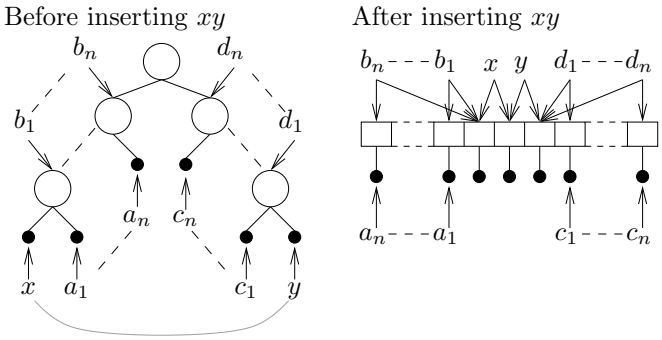

Figure 9: An edge modification implying $\Omega(n)$ changes in the PQ-tree. Before the insertion of xy, there are $4n+3$ nodes in the tree, while after the insertion, there are only $2n+4$ nodes.

Edge modifications. As we mentioned above, we will not specifically consider edge modifications but get them as a consequence of the vertex modification algorithm. Indeed, since we treat vertex insertion and deletion in $O(n)$ time, we can handle the modification of an edge incident to vertex x by first removing x and then inserting x again with the updated neighbourhood. This gives an $O(n)$ time complexity algorithm for edge modifications. We did not try to improve this complexity because there are examples of edge modifications resulting in $O(n)$ changes in each of the three representations we want to maintain[4] (see Figure 9).

Vertex deletions. We now present how to perform the deletion of a vertex x. This operation can be handled very easily thanks to a minimal interval model. First, remove vertex x from the model: you obtain an interval model of $G - x$. But, this model may not be minimal, since some of the cliques of the consecutive ordering may no longer be maximal[5] after the removal of x. Then, we just have to remove the cells of the list implementing the model that no longer correspond to maximal cliques: it is straightforward to see that these cells are precisely those that receive no left pointer from any vertex of the graph, or receive no right pointer from any vertex of the graph. A simple parse of the vertices and of the list allows to remove those cells in $O(n)$ time.

The case of removing a vertex may give a false feeling of simplicity because we use the results of Section 3 that give us the PQ-representation (using the result of [37]) and the MD-representation from an interval model in $O(n)$ time. This is precisely where the difficulty of vertex removal lies, since in order to get a fully dynamic algorithm, we cannot avoid to maintain these representations during a vertex deletion, as we need them in the case of a vertex insertion.

Theorem 7. *Updating the MD-representation, the PQ-representation and a minimal interval model of an interval graph on n vertices under vertex deletion costs $O(n)$ time.*

5. Vertex insertion

We now concentrate, until the end of the paper, on the insertion of a vertex x given together with its neighbourhood in G. In all the following, the graph resulting from the insertion of x in G is denoted by $G' = G + x$.

As already discussed, since the three representations we aim at maintaining are linear-time equivalent (see Section 3), we can choose to maintain only one of them in $O(n)$ time, and get

[4]Note that even renumbering the cells of an interval model may take $\Omega(n)$ time.

[5]Note that, actually this may happen only for the two cliques being the bounds of the interval of x.

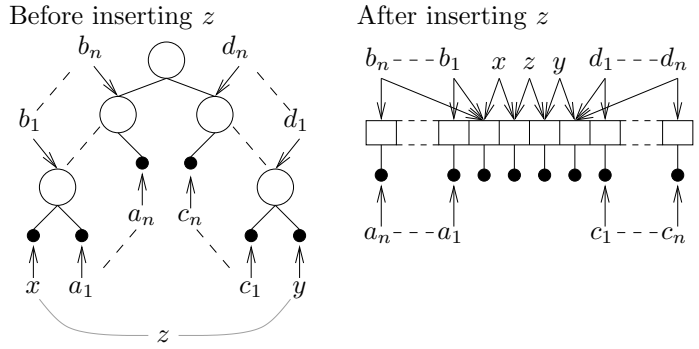

Figure 10: The insertion of vertex z having two neighbours and implying $\Omega(n)$ changes in the PQ-tree. Before the insertion of z, there are $4n+3$ nodes in the tree, while after the insertion, there are only $2n+5$ nodes.

the two others for free within the same time complexity. In the case of the insertion of a vertex, we choose to maintain the MD-representation. The reason of this choice is that this will allow us to distinguish three parts of the algorithm that can be treated separately:

1. Update the shape of the modular decomposition tree, without regard of the fact that G' is an interval graph or not. This is possible as every graph (not necessarily interval) admits a modular decomposition tree. Note that at this stage of the algorithm, we determine the entire shape of the tree as well as the interval models of the representative graphs associated to all the prime nodes of the tree, except for one new prime node w'_s that may be created after the insertion of x.
2. Determine whether G' is an interval graph.
3. In the positive, complete the MD-representation by computing a minimal interval model of the representative graph of node w'_s.

In turns out that part 1 has already been achieved in [36, 24] for arbitrary graphs. This is the reason why we choose to maintain the MD-representation: we save much work to determine the updated shape of the tree. Nevertheless, to achieve part 2 and 3, we will use the PQ-representation which is more directly related to interval models than the MD-representation is.

Concerning the complexity of our algorithm, namely $O(n)$ time, we would like to emphasize the fact that it is very challenging to achieve a better complexity in the worst case. Indeed, as shown on Figure 10, there exist insertions of one vertex having only two neighbours that result in $\Omega(n)$ changes in both the PQ-representation and the MD-representation: that is the numbers of nodes in T^{m} before and after the modification differ from $\Omega(n)$. Therefore, any dynamic algorithm willing to maintain the MD-representation of the graph and storing this structure alone (free from any additional information) in a separate part of the memory cannot avoid the $O(n)$ worst case time complexity. This does not mean that there does not exist a fully dynamic algorithm maintaining the MD-representation in a lower complexity. But any algorithm willing to do so must store the MD-representation as part of a larger structure which is possible to maintain with a complexity lower than $O(n)$ time. Finding such a data-structure and such a dynamic algorithm, if possible, is a highly challenging problem and would certainly open a new field of investigation in algorithmic theory.

Section 5.1 below describes the modifications of the modular decomposition tree under vertex insertion, Section 5.2 characterises the cases where the insertion results in an interval graph; and Section 5.3 gives the whole algorithm and analyses its complexity.

5.1. Modification of T^m

In this section, we refer to the previous works of [36, 24] in order to determine the modifications of the modular decomposition tree T^{m} under the insertion of the vertex x in G. To this purpose, we don't need to worry whether G' is an interval graph or not, since all graphs admit a modular decomposition tree. In addition, we also obtain an interval model for all the prime nodes of $T^{\mathrm{m}}(G')$ except one: a new prime node, denoted w'_s in the following, which may be created after the insertion of x. All the other prime nodes of $T^{\mathrm{m}}(G')$ are also nodes of $T^{\mathrm{m}}(G)$ and have the same representative graph in both trees. Therefore, in this section, not only we determine the shape of $T^{\mathrm{m}}(G')$ but we also determine an interval model of all its prime nodes except one.

Let us now describe the modifications of T^{m}. They occur only in a restricted part of the tree. In order to determine it, [24] introduced the following definition.

Definition 6. *[24] A node u of T^m is a* proper *node iff either u is uniform with respect to x, or u is a mixed node with a (unique) mixed child γ such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(u) \cup \{x\}]$. Otherwise u is a* non-proper *node.*

Remark 8. *The reason why the word unique is between parentheses in the definition above is that if there exists a mixed child γ such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(u) \cup \{x\}]$, then it is necessarily unique (the other children must be uniform).*

The part of T^{m} subject to modifications after the insertion of x is the part rooted at the node w_m called the *insertion node*, formally defined below.

Definition 7. *[24] The* insertion node*, denoted w_m, is defined as being the least common ancestor of the non-proper nodes of T^m.*

It is worth to note that, in the following, we do not consider the trivial case where the vertex set of G is uniform with respect to x. In this case, x is either linked to all the vertices of G or to none of them and the MD-representation is easy to update. As we do not consider this case, the root of T^{m} (as well as the root of T^{c}) is mixed and the insertion node always exists. In [24], it is shown that it satisfies the following property.

Lemma 9. *[24] Let $G' = G + x$ and let w_m be the least common ancestor of non-proper nodes of $T^m(G)$. Node w_m is mixed, non-proper and is such that $V(w_m) \cup \{x\}$ is a strong module of G'.*

This result is of key interest as it implies that only the part of T^{m} rooted at w_m is affected by the insertion of x. Indeed, since $V(w_m) \cup \{x\}$ is a strong module of G', there is a node w'_m in $T^{\mathrm{m}}(G')$ corresponding to the set of vertices $V(w_m) \cup \{x\}$. Moreover, from the properties of modular decomposition, we know that replacing w'_m by a leaf in $T^{\mathrm{m}}(G')$ gives the modular decomposition tree of the quotient graph $G'/\{V(w_m) \cup \{x\}\}$. On the other hand, $G'/\{V(w_m) \cup \{x\}\}$ is equal to $G/\{V(w_m)\}$, for which we know the modular decomposition tree: this the tree obtained from $T^{\mathrm{m}}(G)$ by replacing node w_m by a leaf. Then, we conclude that the part of $T^{\mathrm{m}}(G')$ which is outside of the subtree of $T^{\mathrm{m}}(G')$ rooted at w'_m is exactly the same as the part of $T^{\mathrm{m}}(G)$ which is outside of the subtree of $T^{\mathrm{m}}(G)$ rooted at w_m. As a consequence, in all the rest of the paper, we aim at determining $T^{\mathrm{m}}(G[V(w_m)] + x)$, that is we focus on the insertion of x in $G[V(w_m)]$ and we do not consider any more the part of T^{m} which is out of $T^{\mathrm{m}}_{w_m}$. This is what is stated by the following corollary.

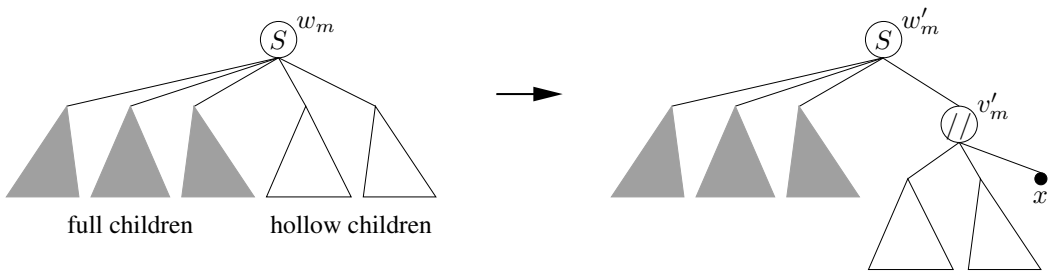

Figure 11: Modification of T^{m} under the insertion of x when w_m is a cut series node.

Corollary 1. *Under the insertion of a vertex x, the only part of T^m which is affected by changes is the subtree $T^m_{w_m}$ rooted at the insertion node w_m. The rest of the tree remains unchanged, as well as the representative graphs of the prime nodes contained in it.*

In order to describe precisely $T^{\mathrm{m}}(G[V(w_m)]+x)$, which will be our goal from now and until the end of Section 5.1, we have to distinguish two main cases, depending on the way x is linked to the children of w_m. We refer to these two cases saying that w_m is *cut* or *uncut*, as defined below.

Definition 8. *The insertion node w_m of $T^m(G)$ is* cut *iff w_m has no mixed child and is either*

1. *a prime node with a (unique) child γ such that $V(\gamma)\cup\{x\}$ is a module of $G'[V(w_m)\cup\{x\}]$, or*
2. *a degenerate node.*

Otherwise w_m is uncut.

Remark 9. *Note that when w_m is a cut prime node, the child γ of w_m such that $V(\gamma)\cup\{x\}$ is a module of $G'[V(w_m)\cup\{x\}]$ is necessarily unique.*

We now discuss depending on whether w_m is cut or uncut, starting by the easier case where w_m is cut.

5.1.1. $T^m(G')$ when the insertion node is cut

Let us describe $T^{\mathrm{m}}(G[V(w_m)]+x)$ in the case where the insertion node is a cut degenerate node (see Figure 11). If w_m is a series (resp. parallel) node, then the root w'_m of $T^{\mathrm{m}}(G[V(w_m)]+x)$ is a series (resp. parallel) node. The children of w'_m are the full (resp. hollow) children of w_m and a new parallel (resp. series) node v'_m. The children of v'_m are $\{x\}$ and the hollow (resp. full) children of w_m.

The case where the insertion node is a cut prime node is also easy to deal with (see Figure 12). Recall that, as stated by Remark 9, the child γ of w_m satisfying the condition in the definition of a cut node is unique. Then, in the children of w_m, this child γ is replaced by a new degenerate node v'_m (i.e. v'_m takes the place of γ in the children of w_m). The label of v'_m is series if γ is full, and parallel if γ is hollow. If the label of γ is different from the one of v'_m, then $\{x\}$ and γ are made children of v'_m. Otherwise, i.e. if γ as the same label as v'_m (which is precisely the case depicted on Figure 12), γ is removed from the tree and the children of v'_m are $\{x\}$ and the children of γ.

Note that when the insertion node is cut, the MD-representation is entirely determined by the description above, that is the prime nodes of the updated MD-representation are the same as those of $MD(G)$ and have the same representative graphs.

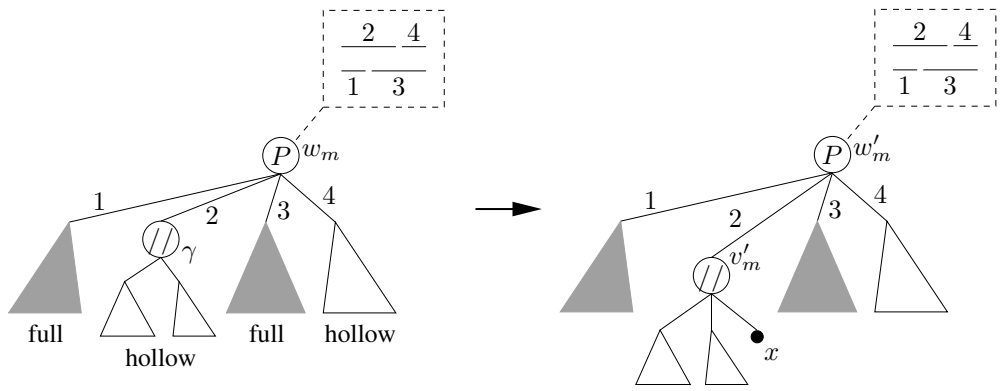

Figure 12: Modification of T^{m} under the insertion of x when w_m is a cut prime node.

5.1.2. $T^m(G')$ when the insertion node is uncut

Let us now consider the much more difficult case where the insertion node w_m is uncut. In the following, $\mathcal{C}_f(u)$, $\mathcal{C}_h(u)$ and $\mathcal{C}_m(u)$ stand for the set of full, hollow and mixed children of u, respectively. For $t \in \{m, f, h\}$, we denote $F_t(u) = \bigcup_{v \in \mathcal{C}_t(u)} V(v)$.

Roughly speaking, in the case where w_m is uncut, the insertion of x will leave unchanged the uniform parts of $T^{\mathrm{m}}_{w_m}$ and the non uniform parts will collapse to form a new prime node (Theorem 8), denoted w'_s in the following. We now define the subset W_s of vertices of G which is such that the set of vertices $V(w'_s)$ associated to this new prime node w'_s in $T^{\mathrm{m}}(G')$ is precisely $W_s \cup \{x\}$.

Notation 5. *Let us define the vertex set W_s as the set $V(w_m)$ if w_m is a prime node and as the set $F_m(w_m) \cup F_h(w_m)$ (resp. $F_m(w_m) \cup F_f(w_m)$) if w_m is a series node (resp. parallel node).*

Determining the modular decomposition of $G[W_s] + x$ (see Theorem 8 below) is the key to obtain the modular decomposition tree of $G[V(w_m)] + x$. From now on, we denote $\mathcal{MUM}(G)$ the set of maximal uniform (with respect to x) modules of a graph G. Note that $\mathcal{MUM}(G)$ is, by definition, a congruence partition of G, that is a partition of the vertex set of G into (disjoint) modules.

Theorem 8. *[24] Let x be a vertex to be inserted in a graph G. If the insertion node w_m of the modular decomposition tree T^m of G is uncut, then $G[W_s] + x$ is connected and co-connected and $\mathcal{MSM}(G[W_s] + x) = \mathcal{MUM}(G[W_s]) \cup \{\{x\}\}$.*

Thanks to Theorem 8, we are able to entirely determine $T^{\mathrm{m}}(G[W_s] + x)$, from which we very easily deduce $T^{\mathrm{m}}(G[V(w_m)] + x)$. Indeed, from Theorem 8, it follows that the root w'_s of $T^{\mathrm{m}}(G[W_s] + x)$ is a prime node (see Section 2.2) whose children are the maximal uniform modules of $G[W_s]$ and $\{x\}$ (see Figure 13). The maximal uniform modules of $G[W_s]$ are nothing else but the sets[6] $F_f(u)$ and $F_h(u)$ that are not empty, with u being any mixed degenerate node of $T^{\mathrm{m}}_{w_m}$, and the sets $V(u)$ with u being any uniform child of any prime node of $T^{\mathrm{m}}_{w_m}$. It turns out that obtaining the tree $T^{\mathrm{m}}(G[M])$ for each maximal uniform module M of $G[W_s]$ is very easy:

[6]Excepted set $F_f(w_m)$ when w_m is series and set $F_h(w_m)$ when w_m is parallel, which are not included in W_s.

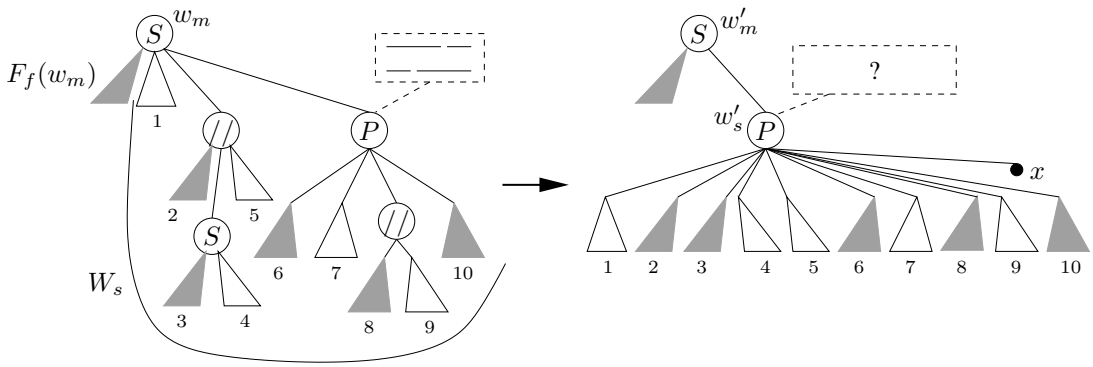

Figure 13: Modification of T^{m} under the insertion of x when w_m is an uncut series node.

1. If $M = F_f(u)$ (resp. $M = F_h(u)$) for some degenerate node u of $T^{\mathrm{m}}_{w_m}$, then $T^{\mathrm{m}}(G[M])$ is the tree formed by making all the nodes in $\mathcal{C}_f(u)$ (resp. $\mathcal{C}_h(u)$) children of a degenerate node having the same label as u, and which is the root of $T^{\mathrm{m}}(G[M])$.
2. If $M = V(u)$ for some uniform child u of some prime node of $T^{\mathrm{m}}_{w_m}$, then $T^{\mathrm{m}}(G[M])$ is simply equal to T^{m}_u.

As a summary, the tree $T^{\mathrm{m}}(G[W_s]+x)$ is made of a prime node as its root, namely w'_s, and the children of w'_s are the leaf corresponding to vertex x and the roots of the trees $T^{\mathrm{m}}(G[M])$ for each maximal uniform module M of $G[W_s]$, as described above (see Figure 13).

Now that we have entirely determined $T^{\mathrm{m}}(G[W_s]+x)$, it is straightforward to obtain $T^{\mathrm{m}}(G[V(w_m)]+x)$ as follows. In the case where w_m is a prime node or w_m is a series (resp. parallel) node such that $F_f(w_m) = \varnothing$ (resp. $F_h(w_m) = \varnothing$), we have $W_s = V(w_m)$ and then $T^{\mathrm{m}}(G[V(w_m)]+x) = T^{\mathrm{m}}(G[W_s]+x)$. Otherwise, if w_m is a series (resp. parallel) node such that $F_f(w_m) \neq \varnothing$ (resp. $F_h(w_m) \neq \varnothing$), the root w'_m of $T^{\mathrm{m}}(G[V(w_m)]+x)$ is a series (resp. parallel) node whose children are the full (resp. hollow) children of w_m and the root w'_s of $T^{\mathrm{m}}(G[W_s]+x)$ (this is precisely the case depicted on Figure 13).

Thus, when w_m is uncut, the shape of the tree $T^{\mathrm{m}}(G[V(w_m)]+x)$ is entirely determined by the description above. Let us now examine what happens for the MD-representation, that is the tree $T^{\mathrm{m}}(G[V(w_m)]+x)$ augmented with the interval models of the representative graphs of prime nodes. From what precedes, the uniform prime nodes of $T^{\mathrm{m}}_{w_m}$ and their representative graphs are preserved in $T^{\mathrm{m}}(G[V(w_m)]+x)$, and so are their minimal interval models. For the mixed prime nodes, the situation is quite different as they simply disappear from T^{m}, and so do the interval models of their representative graphs in the MD-representation. On the other hand, $MD(G')$ contains a new prime node w'_s for which we know the representative graph, which is $G[W_s]+x/\mathcal{MUM}(G[W_s]) \cup \{\{x\}\}$, but we do not have a minimal interval model of it. Actually, there is even no reason why this representative graph should admit an interval model, as we did not worry until now to determine whether G' is an interval graph or not. This is the purpose of Section 5.2 below. From the algorithmic point of view, we will show in Section 5.3 how to compute a model of the representative graph of w'_s when G' is an interval graph.

5.2. Dynamic characterisation of interval graphs

In this section, we characterise the insertions of a vertex x in an interval graph G that result in an interval graph $G' = G + x$. This is done by Lemmas 20 to 23 and Theorem 10. Note that this gives a structural characterisation of the neighbourhood of a vertex in an interval graph which is explicit and independent of the algorithm we design. To characterise the cases where $G+x$ is an interval graph, we will check whether it admits an interval model. That is the reason

why we use the PQ-representation, which is more directly related to interval models, instead of the MD-representation. As previously, we don't need to consider the whole graph but only part of it (cf. Theorem 9). It is worth to note that the interval model we build to show that G' is an interval graph (see the proof of Theorem 10) contains, as an induced sub-model, a model of $G[W_s] + x/\mathcal{MUM}(G[W_s]) \cup \{\{x\}\}$, which is the unique model we need in order to complete the MD-representation of G', i.e. the model of the representative graph of the new prime node w'_s (see Section 5.1.2 above).

5.2.1. The key node w

The aim of this section is to show that in order to determine whether G' is an interval graph, we only need to concentrate on the part of T^{c} rooted at a particular node w, which we call the *key node* and which is, roughly speaking, the node of T^{c} corresponding to node w_m of T^{m}.

But before being able to properly define node w and to state the general characterisation of a successful insertion, we need to exclude the case where the neighbourhood of $V(w_m)$ in G is not a clique or w_m is cut. We start by dealing with the case where the neighbourhood of $V(w_m)$ in G is not a clique. In this case, the insertion of x is always unsuccessful.

Lemma 10. *Let G be an interval graph and let x be a vertex to be inserted in G. If the neighbourhood of $V(w_m)$ in G is not a clique, then $G + x$ is not an interval graph.*

Proof. Indeed, as we saw previously (Lemma 9), $V(w_m) \cup \{x\}$ is a module of G'. Then, the neighbourhood of $V(w_m) \cup \{x\}$ in G' is the same as the one of $V(w_m)$ in G, which is not a clique. Moreover, since w_m is mixed (Lemma 9), it follows that $V(w_m) \cup \{x\}$ is not a clique. Thus, Lemma 1 implies that G' is not an interval graph. □

Note that, from the algorithmic point of view, it will be easy to check whether the condition of Lemma 10 holds: the neighbourhood of $V(w_m)$ is not a clique if and only if parent(w_m) is prime and w_m is non-simplicial in the representative graph $G_{\mathrm{parent}(w_m)}$, that is its interval in the corresponding model contains strictly more than one maximal clique (i.e. strictly more than one cell in the list implementing the model).

As a consequence, from now on, we only consider the case where the neighbourhood of $V(w_m)$ in G is a clique. And we now treat once and for all the case where w_m is cut. As we saw in Section 5.1, when w_m is cut, it is easy to determine $T^{\mathrm{m}}(G')$. It turns out that it is also easy to check whether $G + x$ is an interval graph.

Lemma 11. *When the neighbourhood of $V(w_m)$ in G is a clique and w_m is cut, $G+x$ is not an interval graph iff w_m is series and has a non-leaf full child or w_m is prime and its child γ such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(w_m) \cup \{x\}]$ (cf. Definition 8) is hollow and non-simplicial in the representative graph G_{w_m} of w_m.*

Proof. Let us analyse the different cases that can occur. Along the proof, keep in mind that since G is an interval graph, for any subset $S \subseteq V$ of vertices which is uniform, $G'[S \cup \{x\}]$ is an interval graph.

- If w_m is degenerate, as previously, we denote $\mathcal{C}_h(w_m)$ for the set of its hollow children and $\mathcal{C}_f(w_m)$ for the set of its full children.
  - If w_m is parallel, then $M' = \bigcup_{v \in \mathcal{C}_f(w_m)} V(v) \cup \{x\}$ is a module of G' and the neighbourhood of M' in G' is the same as the one of $V(w_m)$ in G, which is a clique. Then, Lemma 1 concludes that G' is an interval graph.

- If w_m is series and its full children are all leaves, then $M' = \bigcup_{v \in \mathcal{C}_h(w_m)} V(v) \cup \{x\}$ is a module of G' and the neighbourhood of M' in G' is the union of $\bigcup_{v \in \mathcal{C}_f(w_m)} V(v)$ and the neighbourhood of $V(w_m)$ in G, which is a clique since all the full children of w_m are leaves. Then, from Lemma 1, G' is an interval graph.
- If w_m is series and has a non-leaf full child, then $\bigcup_{v \in \mathcal{C}_h(w_m)} V(v) \cup \{x\}$ is a module of G' and its neighbourhood in G' is the union of $\bigcup_{v \in \mathcal{C}_f(w_m)} V(v)$ and the neighbourhood of $V(w_m)$ in G, which is not a clique since some full child of w_m (which is series) is not a leaf. And since $\bigcup_{v \in \mathcal{C}_h(w_m)} V(v) \cup \{x\}$ is not a clique, then, from Lemma 1, G' is not an interval graph.

- If w_m is prime, let γ be its child such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(w_m) \cup \{x\}]$ (cf. Definition 8).
  - If γ is simplicial in the representative graph G_{w_m} of w_m, then the neighbourhood of $V(\gamma)$ in G is a clique and so is the neighbourhood of $V(\gamma) \cup \{x\}$ in G', since $V(\gamma) \cup \{x\}$ is a module of G'. Then, Lemma 1 implies that G' is an interval graph.
  - If γ is not simplicial in G_{w_m} but full, since, from Lemma 1, $V(\gamma)$ is a clique, then $V(\gamma) \cup \{x\}$ is a clique. Since $V(\gamma) \cup \{x\}$ is a module of G', consequently, from Lemma 1, G' is an interval graph.
  - If γ is not simplicial in G_{w_m} and hollow, then the neighbourhood of $V(\gamma)$ in G is not a clique and neither is the one of $V(\gamma) \cup \{x\}$ in G'. Moreover, since γ is hollow, $V(\gamma) \cup \{x\}$ is not a clique. Thus, from Lemma 1 again, G' is not an interval graph.

□

Again, from the algorithmic point of view, the conditions of Lemma 11 are easy to check. Then, in Section 5.3, we check these conditions and if they are satisfied, that is $G + x$ is not an interval graph, then the algorithm stops; otherwise, when w_m is cut and $G + x$ is an interval graph, our algorithm computes $MD(G')$ as described in Section 5.1.1.

Consequently, in the following, we do not consider any more the case where the neighbourhood of $V(w_m)$ in G is not a clique and the case where w_m is cut: until the end of Section 5.2, the neighbourhood of $V(w_m)$ in G is a clique and w_m is uncut. Then, since the neighbourhood of $V(w_m)$ in G is a clique, it follows that the module $V(w_m)$ of G does not satisfy Condition 2 of Theorem 5, which justifies the following definition for the *key node* w.

Definition 9. *The* key node w *is the node of* T^c *such that* $V(w_m) = w^*$ *or* $V(w_m) = w^* \setminus X_w$.

The main property of the key node is the following. This is the property that will allow us to restrict our attention to the subtree T^c_w of T^c rooted at w in all the rest of Section 5.2.

Theorem 9. *$G + x$ is an interval graph iff $G[w^*] + x$ is an interval graph.*

Proof. Since $V(w_m) \cup \{x\}$ is a module of $G + x$ and since w^* is a module of G and $V(w_m) \subseteq w^*$ (see Definition 9), it follows that $w^* \cup \{x\}$ is a module of $G + x$. Furthermore, since the neighbourhood of $V(w_m)$ in G is a clique, the neighbourhood of $w^* \cup \{x\}$ in $G + x$ is also a clique. Then, Lemma 1 implies the equivalence of the claim. □

Thanks to Theorem 9, in our characterisation of a successful insertion (Section 5.2.3), we will be able to restrict our attention to the subtree T^c_w of T^c rooted at w. We will also need to express the fact that w_m is uncut in terms of properties of T^c_w, since the rest of the section on vertex insertion deals with this case. This is done by the lemma below.

Lemma 12. *When w_m is uncut, w has no child v such that $v^* \cup \{x\}$ is a module of $G[w^*] + x$.*

.

Proof. Let v be a child of w such that $v^* \cup \{x\}$ is a module of $G[w^*] + x$. Then, $v^* \cup \{x\}$ is also a module of $G[w^* \setminus X_w] + x$. From Definition 9, either $w^* \setminus X_w = V(w_m)$ or $w^* = V(w_m)$. In the former case, where $w^* \setminus X_w = V(w_m)$, since from Corollary 5, v^* is a maximal strong module of $G[w^* \setminus X_w]$, then there exists a child γ of w_m such that $V(\gamma) = v^*$, and $V(\gamma) \cup \{x\}$ is a module of $G[V(w_m)] + x$. Thus, w_m is cut (see Definition 8).
In the case where $w^* = V(w_m)$, from Corollary 5, $w^* \setminus X_w$ is a maximal strong module of $V(w_m)$. That is, there exists a child γ of w_m such that $V(\gamma) = w^* \setminus X_w$. Since $v^* \cup \{x\}$ is a module of $G[V(w_m)] + x$, and since $V(\gamma)$ is a module of $G[V(w_m)]$ and $v^* \subseteq V(\gamma)$, it follows that $V(\gamma) \cup \{x\}$ is a module of $G[V(w_m)] + x$. And again, w_m is cut. □

5.2.2. Preliminary definitions and properties

In this section, we introduce some notations and establish some technical properties that we use in the proof of our characterisation theorem (Theorem 10 of Section 5.2.3). The proofs of these properties contain some key ideas of our characterisation, but still, they can be skipped in a first lecture. From now on, we consider the case where the neighbourhood of $V(w_m)$ is a clique and w_m is uncut, the other cases have already been treated in Section 5.2.1.

The two lemmas below show that there is a natural way to obtain a consecutive ordering of G from one of G'. This construction plays a key role in the rest of the paper. First, Lemma 13 shows that when removing x from a consecutive ordering of $G + x$, the only cliques that may no longer be maximal after the removal of x, i.e. included in some other clique of the ordering, are the bounds of the interval of x.

Lemma 13. *For any consecutive ordering σ' of $\mathcal{K}(G')$, and for any clique $K' \in \mathcal{K}(G')$, if $K' \setminus \{x\} \subseteq K'_1 \setminus \{x\}$ for some $K'_1 \in \mathcal{K}(G')$, then $x \in K'$ and there exists $K \in \mathcal{K}(G')$ such that K is next to K' in σ' and $x \notin K$ (and then, necessarily, K is maximal in G).*

Proof. If $K' \in \mathcal{K}(G')$ is such that $K' \setminus \{x\} \subseteq K'_1 \setminus \{x\}$ for some $K'_1 \in \mathcal{K}(G')$, then, because of the consecutiveness constraints, it follows that the clique $K \in \mathcal{K}(G')$ that is between K' and K'_1 and next to K' in σ' is such that $K' \setminus \{x\} \subseteq K \setminus \{x\}$. Since K' is maximal in G', necessarily, $x \in K'$ and $x \notin K$. □

Remark 10. *Moreover, note that for all $K', K'' \in \mathcal{K}(G')$, we have $K' \setminus \{x\} \neq K'' \setminus \{x\}$, because otherwise at least one of K', K'' would be included in the other one.*

Lemma 14 states that removing x from a consecutive ordering σ' of G', and possibly the at most two cliques that are the bounds of its interval (if they are included in some other clique), results in a consecutive ordering of G.

Lemma 14. *For any consecutive ordering σ' of $\mathcal{K}(G')$, consider the linear ordering $\sigma' - x$ on some cliques of G obtained by:*

1. *removing x from all the maximal cliques in σ', and*
2. *removing each of the at most two cliques $K'_1 \setminus \{x\}$ and $K'_2 \setminus \{x\}$ that were the bounds of the interval of x in σ' if they are included in some other clique after the removal of x.*

Then $\sigma' - x$ is a consecutive ordering of $\mathcal{K}(G)$.

Proof. Clearly, both removing x from all the maximal cliques in σ' and removing some of the resulting cliques preserve consecutiveness constraints of the vertices of $V(G') \setminus \{x\}$. Then, we just have to check that after those removals, the obtained sequence of cliques contains all the maximal cliques of G exactly once. It is easy to see that after the removal of x, all the maximal cliques of G are in the sequence. Indeed, consider a maximal clique K of G. Either all the vertices of K are adjacent to x, and then $K \cup \{x\}$ is a maximal clique of G', or some vertex of K is not adjacent to x, and then K is a maximal clique of G'. In any case, K or $K \cup \{x\}$ is in $\mathcal{K}(G')$, and it follows that there is some clique equal to K after the removal of x.
On the other hand, Lemma 13 states that the only clique that may be included in some other after the removal of x are the bounds of its interval. Then, from the definition of $\sigma' - x$, it follows that the obtained sequence contains only the maximal cliques of G, and at most once. Consequently, $\sigma' - x$ is a consecutive ordering of $\mathcal{K}(G)$. □

We now state some basic relationships between the consecutive orderings of the maximal cliques of $G + x$ and those of the maximal cliques of G.

Remark 11. *Let σ' be a consecutive ordering of G', let u be a node of $T = T^c(G)$ and let K' be a maximal clique of G' such that $K' \setminus \{x\}$ is not a leaf of the subtree T_u but the maximal cliques of G containing $K' \setminus \{x\}$ are all leaves of T_u. Then, there exists a clique K'' of G' that is next to K' in σ' and such that $K'' \setminus \{x\}$ is a leaf of T_u.*

Proof. This is a corollary of Lemma 13. First, observe that if $K' \setminus \{x\}$ is maximal in G, then, since all the maximal cliques of G containing $K' \setminus \{x\}$ are leaves of T_u, $K' \setminus \{x\}$ must be a leaf of T_u. Thus, $K' \setminus \{x\}$ is not maximal in G. It follows, from Lemma 13, that $K' \setminus \{x\}$ is included in some maximal clique K'' of G' that is next to K in σ' and that does not contain x. Therefore K'' is also a maximal clique of G. Moreover, since the maximal cliques of G containing $K' \setminus \{x\}$ are leaves of T_u, then K'' is a leaf of T_u and so is $K'' \setminus \{x\}$, since $K'' = K'' \setminus \{x\}$. □

In the following, we will often use some intervals $I_{\sigma'}(v)$ of a consecutive ordering σ' of $G + x$ that are associated with the nodes $v \in T^c(G)$. Lemma 15 states that these intervals are disjoint for two nodes incomparable for the ancestor relationship and that they appear in σ' in an order that respects the order on the children of prime nodes.

Lemma 15. *Let σ' be a consecutive ordering of $\mathcal{K}(G')$ and $T = T^c(G)$. For any node $u \in T$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. Then, the two following properties hold.*

1. *For all pairs u_1, u_2 of nodes of T such that none of them is an ancestor of the other one, $I_{\sigma'}(u_1)$ and $I_{\sigma'}(u_2)$ are disjoint.*
2. *For all prime nodes $u \in T$, up to reversing σ_u, we have $\forall u_1, u_2 \in \mathcal{C}(u), u_1 <_{\sigma_u} u_2 \Rightarrow I_{\sigma'}(u_1)$ is before $I_{\sigma'}(u_2)$ in σ'.*

Proof. Let σ' be a consecutive ordering of $\mathcal{K}(G')$. Let $u_1, u_2 \in T$ independent for the parent relationship. From Lemma 14, $\sigma' - x$ is a consecutive ordering of $\mathcal{K}(G)$. Since the leaves of T_{u_1} and the leaves of T_{u_2} are disjoint intervals of $\sigma' - x$, so are $I_{\sigma'}(u_1)$ and $I_{\sigma'}(u_2)$ in σ'.

Moreover, for a prime node $u \in T$, up to reversing σ_u, the leaves of T_v for each $v \in \mathcal{C}(u)$ form an interval of $\sigma' - x$ and these intervals appear in the same order in $\sigma' - x$ than the children of u do in σ_u. So do the intervals $I_{\sigma'}(v)$ in σ'. □

The following lemma establishes a relationship between the fact that a node $u \in T^c(G)$ is non-full or linked or saturated and the presence of x in the sets of cliques of G' associated to u.

Lemma 16. *Let u be a node of $T = T^c(G)$. The following properties hold.*

1. *If u is not full, then there exists a clique $K' \in \mathcal{K}(G')$ such that $x \notin K'$ and K' is a leaf of T_u.*
2. *If u is linked, then there exists a clique $K' \in \mathcal{K}(G')$ such that $x \in K'$ and all the maximal cliques of G containing $K' \setminus \{x\}$ are leaves of T_u.*
3. *u is saturated iff x belongs to all the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u.*

Proof.

If u is not full, then there exists $y \in u^*$ such that y is not adjacent to x. Let K' be a leaf of T_u that contains y. Because y is not adjacent to x, then K' is also a maximal clique of G', which proves the first property of the lemma.

If u is linked, there exists $y \in u^* \cap N(x)$. Then there is a maximal clique K' of G' containing both y and x. Since $y \in K' \setminus \{x\}$, necessarily the maximal cliques of G containing $K' \setminus \{x\}$ are leaves of $T^c_{e_y}$ and so of T^c_u, which proves the second property of the lemma.

If u is saturated, let $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. If $x \notin K'$, then K' itself is a leaf of T_u. Consequently, since u is saturated, x is adjacent to all the vertices of K' which is therefore not maximal in G'. Thus, x must belong to K'. Conversely, if u is not saturated, there exists a maximal clique K of G that is a leaf of T_u and that is not full. Then K is maximal in G', K does not contain x and $K \setminus \{x\}$ is a leaf of T_u. □

Lemma 17 gives some conditions on the way x touches a node u of $T^c_w(G)$ under which it is always possible, in any consecutive ordering σ' of G', to find a clique containing x which is out of $I_{\sigma'}(u)$.

Lemma 17. *For any consecutive ordering σ' of $\mathcal{K}(G')$ and any node u of $T = T^c(G)$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. If $u \in T_w \setminus \{w\}$ is such that B_u is full (resp. if w is such that B_w is not full), then there exists a clique K' of $\mathcal{K}(G')$ such that K' contains x and for any consecutive ordering σ' of $\mathcal{K}(G')$, $K' \notin I_{\sigma'}(u)$ (resp. $K' \notin I_{\sigma'}(w)$).*

Proof. Let σ' be a consecutive ordering of $\mathcal{K}(G')$. Let us start with the case of node w. Since B_w is not full, $B_w \setminus N(x) \neq \varnothing$ and there is a maximal clique K_x of G' containing x and not containing some vertex y of B_w. Since all the cliques of $I_{\sigma'}(w)$ contain all the vertices of B_w, $K_x \notin I_{\sigma'}(w)$.

Let u be a node of $T_w \setminus \{w\}$ such that B_u is full, and let v be the unique node in $\mathcal{C}(w) \cap \mathrm{Anc}(u)$. Since w is not proper and not cut, from Lemma 12, $v^* \cup \{x\}$ is not a module of G'. The neighbours of the nodes of v^* in G' are the nodes of B_v, which are also adjacent to x since B_u, and so B_v, is full. It follows, since $v^* \cup \{x\}$ is not a module of G', that there exists some $y \in V(G) \setminus (B_v \cup v^*)$ such that y is adjacent to x (while y is not adjacent to the vertices of v^*). Consequently, there is a clique $K' \in \mathcal{K}(G')$ containing both x and y. Clearly, since $y \in K'$, the maximal cliques of G that contain $K' \setminus \{x\}$ are leaves of T_{e_y}. Then, from Remark 11, $K' \in I_{\sigma'}(e_y)$ or K' is next to some clique of $I_{\sigma'}(e_y)$. On the other hand, since e_y and v are independent for the ancestor relationship, from the first property of Lemma 15, it follows that $I_{\sigma'}(e_y)$ and $I_{\sigma'}(v)$ are disjoint. Therefore, either K' is out of $I_{\sigma'}(v)$ or K' is a bound of $I_{\sigma'}(v)$. But since $y \in K'$, $K' \setminus \{x\}$ is not a leaf of T_v and it follows that K' is not a bound of $I_{\sigma'}(v)$. Consequently, $K' \notin I_{\sigma'}(v)$ and as $I_{\sigma'}(u) \subseteq I_{\sigma'}(v)$, we obtain that $K' \notin I_{\sigma'}(u)$, which ends the proof. □

Lemmas 18 and 19 give some conditions on the way x touches a prime node u of $T^c(G)$ which force some children of u to be saturated.

Lemma 18. *Let u be a prime node of $T^c(G)$ and let $u_1, u_2 \in \mathcal{C}(u)$ such that $u_1 <_{\sigma_u} u_2$. If u_1 is linked or u_1 is saturated or there exists $y \in Y_u \cap N(x)$ such that $e_y^2 = u_1$, and if u_2 is linked or u_2 is saturated or there exists $z \in Y_u \cap N(x)$ such that $e_z^1 = u_2$, then all the children v of u such that $u_1 <_{\sigma_u} v <_{\sigma_u} u_2$ are saturated.*

Proof. Let σ' be a consecutive ordering of $\mathcal{K}(G')$ and let $v \in \mathcal{C}(u)$ such that $u_1 <_{\sigma_u} v <_{\sigma_u} u_2$. As previously, for any node u of $T = T^{\mathrm{c}}(G)$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. If u_1 is linked or saturated then, from Lemma 16, there exists a clique $K_a \in \mathcal{K}(G')$ that contains x and such that $K_a \setminus \{x\}$ is a leaf of T_u. If there exists $y \in Y_u \cap N(x)$ such that $\mathrm{e}_y^2 = u_1$, then there exists a clique K_b containing both x and y. Since y belongs to no other maximal cliques of G than those that are leaves of T_α for $\alpha \in \mathcal{C}_\Delta(y)$, from Remark 11, $K_b \in \bigcup_{\alpha \in \mathcal{C}_\Delta(y)} I_{\sigma'}(\alpha)$ or K_b is next to a clique in $\bigcup_{\alpha \in \mathcal{C}_\Delta(y)} I_{\sigma'}(\alpha)$. It follows from Lemma 15 that up to reversing σ', both K_a and K_b are before the cliques of $I_{\sigma'}(v)$ in σ'. Then, in any case, there always exists some clique, denoted K_{x1}, that contains x and that is before the cliques of $I_{\sigma'}(v)$ in σ'. And similarly, because the same conditions are satisfied for u_2, there is always a clique K_{x2} containing x that is after the cliques of $I_{\sigma'}(v)$ in σ'. From the third property of Lemma 16, this implies that v is saturated. □

Lemma 19. *Let σ' be a consecutive ordering of $\mathcal{K}(G')$. For any node u of $T = T^c(G)$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. Let u be a prime node of T such that for any children v_1, v_2 of u, $v_1 <_{\sigma_u} v_2$ iff $I_{\sigma'}(v_1)$ is before $I_{\sigma'}(v_2)$ in σ'. Let u_1 be a child of u. If u_1 is linked or u_1 is saturated or there exists $y \in Y_u \cap N(x)$ such that $e_y^2 = u_1$, and if there exists a clique $K_x \in \mathcal{K}(G')$ that contains x and that is after $I_{\sigma'}(u)$ in σ', then for all $v \in \mathcal{C}(u)$ such that $u_1 <_{\sigma_u} v$, v is saturated.*

Proof. The proof is the same as the one of Lemma 18, replacing K_{x2} with K_x. □

The notions of left and right properties defined below play a key role in our characterisation of successful insertions.

Definition 10 (left and right properties). *A child v of a prime node u satisfies the* left property *(resp. right property) iff for all $y \in Y_u \cap N(x)$, we have $v \leq_{\sigma_u} e_y^2$ (resp. $e_y^1 \leq_{\sigma_u} v$).*

With the notations we adopted and the technical properties shown above, we are now ready to state the characterisation we aim at.

5.2.3. The main theorem

The goal of this section is to establish necessary and sufficient conditions on the way x touches the nodes of $T_w^{\mathrm{c}}(G)$ such that G' is an interval graph: Lemmas 20 to 23 give some necessary conditions, and Theorem 10 states that they are also sufficient.

One key property of our characterisation is that these conditions are local to each of the nodes of T_w^{c}. This will allow us to test them in $O(n)$ total time in the algorithmic part of the paper (Section 5.3). Another very important feature of our proof of the fact that the conditions we require are sufficient is that it is constructive: we explicitly build a consecutive ordering of $G[w^*] + x$. We will use this construction in our algorithm in order to get a minimal interval model of the new prime node w'_s created in T^{m} after the insertion of x.

Remind that in all the rest of the section we deal with the insertion case where the neighbourhood of $V(w_m)$ is a clique and w_m is uncut.

The first necessary conditions apply on the types (linked or not) of the nodes of T^{c}_w. In particular, it requires that the mixed nodes of T^{c}_w are spread on at most two branches rooted at w.

Lemma 20. *If $G + x$ is an interval graph, then any node $u \in T^c_w(G) \setminus \{w\}$ has at most one mixed child and w has at most two mixed children. Furthermore, for all $u \in T^c_w(G)$, if B_u is not full, then u has at most one linked child.*

Proof. For simplicity of notations, let us denote $T = T^{\mathrm{c}}(G)$. If B_u is not full, there exists $z \in B_u \setminus N(x)$. Suppose that u has two linked children u_1, u_2, then there exist $y_1 \in u_1^* \cap N(x)$ and $y_2 \in u_2^* \cap N(x)$. Note that z is adjacent to both y_1 and y_2. Then, the four vertices x, y_1, z, y_2 induce a C_4 in $G + x$, which is impossible since any interval graph is chordal.

For any node u of T, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. From Lemma 15, in any linear ordering σ' of $\mathcal{K}(G')$, the intervals $I_{\sigma'}(w_1), \ldots, I_{\sigma'}(w_k)$, where $\{w_1, \ldots, w_k\} = \mathcal{C}(w)$, are disjoint. In addition, from Lemma 16, for any non-full node $u \in T$, $I_{\sigma'}(u)$ contains a clique that does not contain x. On the other hand, also from Lemma 16, any linked node $u \in T$ is such that there exists a clique $K' \in \mathcal{K}(G')$ such that $x \in K'$ and all the maximal cliques of G containing $K' \setminus \{x\}$ are leaves of T_u. And from Remark 11, it follows that this clique K' is next to some clique of $I_{\sigma'}(u)$. Then, the consecutiveness constraint imposed by x on σ' implies that w cannot have three (or more) mixed children, i.e. that are both linked and non-full.

At last, for a node $u \in T_w \setminus \{w\}$ such that B_u is full, Lemma 17 implies that there exists a clique containing x that does not belong to $I_{\sigma'}(u)$, for any consecutive ordering σ' of $\mathcal{K}(G')$. Again, Lemmas 15 and 16, together with the consecutiveness constraint imposed by x on σ', imply that u cannot have two (or more) mixed children. □

The other necessary conditions for G' to be an interval graph apply only to prime nodes of $T^{\mathrm{c}}_w(G)$. Lemma 21 states that the saturated children of a prime node form an interval and that (almost) all the children out of this interval are hollow.

Lemma 21. *If $G + x$ is an interval graph, then, for all prime nodes $u \in T^c_w(G)$, the set of saturated children of u is an interval S_u of σ_u. And if $S_u \neq \varnothing$, then any node $v_1 \in \mathcal{C}(u) \setminus (S_u \cup \{l_u, r_u\})$ is hollow, where l_u (resp. r_u) denotes the child of u immediately preceding (resp. following) the interval S_u in σ_u, when such a node exists, and l_u (resp. r_u) denotes the first (resp. last) child of u in σ_u otherwise.*

Proof. Lemma 18 implies that all the children of u that lie in σ_u between two saturated children are themselves saturated. Thus, the set of saturated children of u is an interval S_u. If $S_u \neq \varnothing$ and if there exists a child v of u such that $v <_{\sigma_u} l_u$, then, necessarily, $l_u \notin S_u$ and there exists a saturated child u_s of u such that $u_s >_{\sigma_u} l_u$. It follows that the children v of u such that $v <_{\sigma_u} l_u$ are necessarily hollow, since if they were not, from Lemma 18, l_u would be saturated. By symmetry, the same holds for children v of u such that $r_u <_{\sigma_u} v$. □

Then, from now on, when $G + x$ is an interval graph, from Lemma 21, we can adopt the following notation.

Notation 6 (S_u, l_u, r_u). *When $G + x$ is an interval graph, we denote S_u the interval of σ_u formed by the saturated children of u. And if $S_u \neq \varnothing$, then l_u (resp. r_u) denotes the child of u immediately preceding (resp. following) the interval S_u in σ_u, when such a node exists, and l_u (resp. r_u) denotes the first (resp. last) child of u in σ_u otherwise.*

Lemmas 22 and 23 claim that the children, l_u and r_u, of a prime node u that are located on the side of the interval of saturated children of u must satisfy the left or right property. Lemma 22 is for nodes u different from w, while Lemma 23 treats the specific case of node w.

Lemma 22. *If $G + x$ is an interval graph, then any prime node $u \neq w$ of $T^c_w(G)$ satisfies one of the following conditions.*

1. *B_u is full and $S_u \neq \varnothing$; and, up to reversing σ_u, $\max(\sigma_u) \in S_u$ and l_u satisfies the left property.*
2. *B_u is full and $S_u = \varnothing$, or B_u is not full; and, up to reversing σ_u, $\max(\sigma_u)$ satisfies the left property and the nodes of $\mathcal{C}(u) \setminus \{\max(\sigma_u)\}$ are hollow.*

Proof. Let σ' be a consecutive ordering of $\mathcal{K}(G')$. For any node u of $T = T^{\mathrm{c}}(G)$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u.

- Let us first examine the cases where B_u is full. Lemma 17 implies that there is a clique $K_x \in \mathcal{K}(G')$ that contains x and that does not belong to $I_{\sigma'}(u)$. Up to reversing σ', we can assume that K_x is after $I_{\sigma'}(u)$ in σ'. And, from Property 2 of Lemma 15, we can assume that, up to reversing σ_u, $\max(\sigma_u)$ is the child v of u whose interval $I_{\sigma'}(v)$ is the closest from K_x.

  If $S_u \neq \varnothing$, then there exists a child u_1 of u which is saturated, and Lemma 19 implies that $\max(\sigma_u)$ is also saturated.

  If $S_u = \varnothing$, and if there exists a child u_1 of u which is linked, then Lemma 19 implies that all the children v of u such that $u_1 <_{\sigma_u} v$ are saturated. Since u has no saturated child, it implies that $u_1 = \max(\sigma_u)$. Thus, the nodes of $\mathcal{C}(u) \setminus \{\max(\sigma_u)\}$ are hollow.

  Now, let us denote λ for l_u when $S_u \neq \varnothing$ and for $\max(\sigma_u)$ when $S_u = \varnothing$. Note that, in both cases, λ is not saturated. Suppose for contradiction that there exists $y \in Y_u$ such that $\mathrm{e}^2_y <_{\sigma_u} \lambda$. Since there exists a clique K_x containing x that is after $I_{\sigma'}(u)$ in σ', Lemma 19 implies that λ is saturated, which is a contradiction. Thus, λ satisfies the left property and u satisfies either condition 1 (when $S_u \neq \varnothing$) or condition 2 (when $S_u = \varnothing$).

- In the case where B_u is not full, since necessarily B_w is not full, Lemma 17 implies that there exists a clique $K_x \in \mathcal{K}(G')$ that contains x and that does not belong to $I_{\sigma'}(w)$. Since $I_{\sigma'}(u) \subseteq I_{\sigma'}(w)$, $K_x \notin I_{\sigma'}(u)$. Then, up to reversing σ' and/or σ_u as done previously in the case where B_u is full, we can assume wlog. that K_x is after $I_{\sigma'}(u)$ in σ' and that $\max(\sigma_u)$ is the child v of u whose interval $I_{\sigma'}(v)$ is the closest from K_x.

  Since B_u is not full, Lemma 20 implies that u has at most one linked child u_1. Suppose that $u_1 \neq \max(\sigma_u)$. Then, Lemma 19 implies that $\max(\sigma_u)$ is saturated. Since $\max(\sigma_u)^* \neq \varnothing$ (this is a consequence of Lemma 6), it follows that $\max(\sigma_u)$ is linked, which is a contradiction with the fact that u has at most one linked child. Thus, $u_1 = \max(\sigma_u)$ is the only possibly linked child of u. That is, the nodes of $\mathcal{C}(u) \setminus \{\max(\sigma_u)\}$ are hollow.

  Now, suppose that there exists $y \in Y_u \cap N(x)$ such that $\mathrm{e}^2_y <_{\sigma_u} \max(\sigma_u)$. From Lemma 19, this implies that $\max(\sigma_u)$ is saturated. Let $z_2 \in \max(\sigma_u)^* \cap N(x)$. Since B_u is not full, there exists $z_1 \in B_u \setminus N(x)$, and the four vertices x, y, z_1, z_2 induces a C_4 in $G + x$, which is a contradiction. Thus, $\max(\sigma_u)$ satisfies the left property and u satisfies Condition 2.

□

The conditions of Lemma 23 are essentially the same than those of Lemma 22, except that the interval S_w of saturated children does not need to reach one extremity of σ_w. This implies that the node on the left of S_w must satisfy the left property, while the node on the right of S_w must satisfy the right property.

Lemma 23. *If $G+x$ is an interval graph and if w is a prime node, then w satisfies one of the following conditions:*

1. *B_w is full and $S_w \neq \varnothing$; and l_w and r_w satisfy respectively the left and right property.*
2. *B_w is full and $S_w = \varnothing$; and, one of the two following conditions holds:*
   (a) *there exist two consecutive elements l and r in σ_w, with $l <_{\sigma_w} r$, that satisfy resp. the left and right property, and the nodes of $\mathcal{C}(w) \setminus \{l, r\}$ are hollow, and all the vertices of Y_w that cover both l and r are linked; or*
   (b) *up to reversing σ_w, $\max(\sigma_w)$ satisfies the left property and the nodes of $\mathcal{C}(w) \setminus \{\max(\sigma_w)\}$ are hollow.*
3. *B_w is not full; and, up to reversing σ_w, $\max(\sigma_w)$ satisfies the left property and the nodes of $\mathcal{C}(w) \setminus \{\max(\sigma_w)\}$ are hollow.*

Proof. Like previously, for any consecutive ordering σ' of $\mathcal{K}(G')$ and for any node u of $T = T^{\mathrm{c}}(G)$, we denote $I_{\sigma'}(u)$ the smallest interval of σ' containing the cliques $K' \in \mathcal{K}(G')$ such that $K' \setminus \{x\}$ is a leaf of T_u. The conditions of Lemma 23 are close to but different from those of Lemma 22 because of the special position of w. In particular, when B_w is full, there may not be any clique containing x out of $I_{\sigma'}(w)$ for any consecutive ordering σ' of $\mathcal{K}(G')$. The numbers of the case by case analysis below refer to those numbering conditions of Lemma 23.

1. The proof that l_w satisfies the left property is exactly the same as the proof that l_u satisfies the left property in Condition 1 of Lemma 22. By symmetry, r_w must satisfy the right property.
2. Let $\alpha = min\{\mathrm{e}_y^2 \mid y \in Y_w \cap N(x)\}$ and $\beta = max\{\mathrm{e}_y^1 \mid y \in Y_w \cap N(x)\}$. The children of w that are not after α in σ_w are those that satisfy the left property, and those that are not before β are those that satisfy the right property. If $\alpha <_{\sigma_w} \beta$, then, from Lemma 18 all the nodes strictly between α and β have to be saturated. Consequently, since w has no saturated child, we have $\beta \leq_{\sigma_w} \alpha$ or α is the predecessor of β. It follows that every child of w satisfies at least one of the left and right property.
   Since none of the children of w are saturated, and since Lemma 18 implies that the children strictly between two linked children are saturated, it follows that w has at most two linked children, and they must be consecutive. Moreover, if there is a linked child v of w and a vertex $y \in Y_w \cap N(x)$ such that $v <_{\sigma_w} \mathrm{e}_y^1$ (resp. $\mathrm{e}_y^2 <_{\sigma_w} v$), then Lemma 18 implies that all the nodes strictly between v and e_y^1 (resp. e_y^2) have to be saturated. It follows that for any linked child v of w, its successor (if it exists) satisfies the right property and its predecessor satisfies the left property.
   - Consequently, if w has two linked children u_1 and u_2 such that $u_1 <_{\sigma_w} u_2$, then they are consecutive in σ_w, u_1 satisfies the left property and u_2 satisfies the right property. Suppose that there exists $y \in (\Delta_{u_1} \cap \Delta_{u_2}) \setminus N(x)$. Let $z_1 \in u_1^* \cap N(x)$ and let $z_2 \in u_2^* \cap N(x)$, then the four vertices x, z_1, y, z_2 induces a C_4 in $G+x$, which is a contradiction. Thus, $\Delta_{u_1} \cap \Delta_{u_2} \subseteq N(x)$ and Condition 2a is satisfied.
   - Let us examine the case where w has exactly one linked child v.
     - If v does not satisfy one of the left and right properties, then we can assume, up to reversing σ_w, that v does not satisfy the right property. Then, $v \neq \max(\sigma_w)$

since $\max(\sigma_w)$ satisfies the right property. As shown before, the successor v_+ of v must satisfy the right property. Since v_+ satisfies it but v does not, there exists $y \in Y_u$ such that $\mathrm{e}_y^1 = v_+$. It follows that any $z \in \Delta_v \cap \Delta_{v_+}$ must be adjacent to x: otherwise, x, y, z and any vertex $y_2 \in v^* \cap N(x)$ would induce a C_4 in $G + x$. Thus, the couple (v, v_+) shows that w satisfies Condition 2a.

- Consider now the case where v satisfies both the left and right properties. If v is the first or last node of σ_w, then w satisfies Condition 2b. Otherwise, if v has a predecessor v_- and a successor v_+ in σ_w, then suppose for contradiction that there exists $y_- \in (\Delta_{v_-} \cap \Delta_v) \setminus N(x)$ and $y_+ \in (\Delta_v \cap \Delta_{v_+}) \setminus N(x)$. Let σ' be a consecutive ordering of G'. Since $v^* \cap N(x) \neq \varnothing$, there is a clique $K_{xy} \in \mathcal{K}(G')$ that contains both x and some vertex $y \in v^* \cap N(x)$. Since y_- and y_+ are not adjacent to x, K_{xy} contains neither y_- nor y_+. It follows that $K_{xy} \notin I_{\sigma'}(v_-) \cup I_{\sigma'}(v) \cup I_{\sigma'}(v_+)$, since the cliques in these three intervals all contain either y_- or y_+. Moreover, since K_{xy} contains $y \in v^*$, it follows from Remark 11 that K_{xy} is next to some clique of $I_{\sigma'}(v)$. Consequently, K_{xy} is either between the cliques of $I_{\sigma'}(v_-)$ and those of $I_{\sigma'}(v)$, or between the cliques of $I_{\sigma'}(v)$ and those of $I_{\sigma'}(v_+)$. The former would contradict the fact that $y_- \notin K_{xy}$ and the later would contradict the fact that $y_+ \notin K_{xy}$. Thus, $\Delta_{v_-} \cap \Delta_v \subseteq N(x)$ or $\Delta_v \cap \Delta_{v_+} \subseteq N(x)$. Since v_- satisfies the left property, v satisfies both the left and right properties and v_+ the right one, it follows that either (v_-, v) or (v, v_+) suits for the couple (l, r) of Condition 2a.

- If w has no linked child, and if $\beta = \min(\sigma_w)$ or $\alpha = \max(\sigma_w)$ then w satisfies Condition 2b. Otherwise, we consider a consecutive ordering σ' of G' and distinguish two cases.

  If α is the predecessor of β, suppose for contradiction that there is a vertex $y \in (\Delta_\alpha \cap \Delta_\beta) \setminus N(x)$. Let $z_1 \in Y_w \cap N(x)$ such that $e_{z_1}^2 = \alpha$ and let $z_2 \in Y_w \cap N(x)$ such that $e_{z_2}^1 = \beta$. Then, vertices x, z_1, y, z_2 induce a C_4 in $G+x$, which is a contradiction. Thus, $\Delta_\alpha \cap \Delta_\beta \subseteq N(x)$ and w satisfies Condition 2a.

  If $\beta \leq_{\sigma_w} \alpha$, and if the interval $[\![\beta, \alpha]\!]$ has length k, with $k \geq 1$, then we denote $(v_1, \ldots, v_k)$ for $[\![\beta, \alpha]\!]$. Furthermore, we denote v_0 and v_{k+1} for respectively the child preceding and following $[\![\beta, \alpha]\!]$ in $\mathcal{C}(w)$. By definition, all the vertices of $Y_w \cap N(x)$ cover the nodes of $[\![\beta, \alpha]\!]$. And there is a clique K_x containing x and the vertices of $Y_w \cap N(x)$. Suppose for contradiction that for all $i \in [\![0, k]\!]$, there exists a vertex $y_i \in Y_w \setminus N(x)$ that covers both v_i and v_{i+1}. Since K_x does not contain any y_i for $1 \leq i \leq k$, K_x cannot belong to $\bigcup_{1 \leq i \leq k} I_{\sigma'}(v_i)$. But since the only children of w containing all the vertices of $Y_w \cap N(x)$ are the nodes in $[\![\beta, \alpha]\!]$, then, from Lemma 13, K_x is next to some clique of $\bigcup_{1 \leq i \leq k} I_{\sigma'}(v_i)$. Thus, there exists $i \in [\![0, k]\!]$ such that K_x is between the cliques of $I_{\sigma'}(v_i)$ and those of $I_{\sigma'}(v_{i+1})$. But y_i is in all the cliques of $I_{\sigma'}(v_i)$ and $I_{\sigma'}(v_{i+1})$ and does not belong to K_x. This is a contradiction with the consecutiveness constraint of y_i. Thus, there exists $i \in [\![0, k]\!]$ such that $\Delta_{v_i} \cap \Delta_{v_{i+1}} \subseteq N(x)$ and (v_i, v_{i+1}) suits for the couple (l, r) of Condition 2a.

3. From Lemma 17, there exists a clique K_x containing x such that for any consecutive ordering σ' of $\mathcal{K}(G')$, K_x does not belong to $I_{\sigma'}(w)$. Then, the rest of the proof follows like in Case 2 of Lemma 22 when B_u is not full.

□

Lemmas 20 to 23 give some necessary conditions so that the insertion of a vertex x results in an interval graph. We will now show (Theorem 10) that these conditions are also sufficient, which

provides us with the characterisation we need to algorithmically decide whether an insertion is valid or not.

The proof of Theorem 10 is constructive, in the sense that we explicitly build a consecutive ordering of $G[w^*] + x$, provided that the conditions of Lemma 20 to 23 are satisfied. The construction is bottom-up: we start by building consecutive orderings for the lower nodes of T_w^{c} and we compose these orderings in order to obtain consecutive orderings of the nodes located above in the tree. In order to conveniently describe these compositions of orderings, and make the proof as light as possible, we make use of the following notations.

Notation 7. *As we did previously for T^m, for any node $u \in T^c$, we denote $\mathcal{C}_h(u)$ for the set of its hollow children, $\mathcal{C}_m(u)$ for the set of its mixed children and $\mathcal{C}_f(u)$ for the set of its full children.*

Notation 8. *Let σ_1 be a linear ordering on set S_1 and let σ_2 be a linear ordering on set S_2, we denote $\sigma_1 + \sigma_2$ the linear ordering on set $S = S_1 \cup S_2$ defined by:*
$\forall x, y \in S, x \leq y \Leftrightarrow (x \in S_1$ and $y \in S_2)$ or $(x, y \in S_1$ and $x \leq_{\sigma_1} y)$ or $(x, y \in S_2$ and $x \leq_{\sigma_2} y)$.
When S_2 (resp. S_1) is a singleton $\{e\}$, we simply denote $\sigma_1 + e$ (resp. $e + \sigma_2$) instead of $\sigma_1 + \sigma_2$.
We define a neutral element, denoted $\perp$, such that for all orderings σ we have $\sigma + \perp = \perp + \sigma = \sigma$.
Since the operator $+$ defined above is associative, for a set $I = \{i_1, \ldots, i_k\}$ where $k \geq 1$ is an integer, we use the classic notation $\sum_{i \in I} \sigma_i = \sigma_{i_1} + \ldots + \sigma_{i_k}$. And by convention, $\sum_{i \in \varnothing} \sigma_i = \perp$

Notation 9. *Let $\sigma = (S_1, \ldots, S_k)$ be a linear ordering where $k \geq 1$ is an integer and for all $i \in [\![1, k]\!], S_i$ is a set. Let X be a set disjoint from $\bigcup_{1 \leq i \leq k} S_i$, we denote $\sigma_{<X>}$ for the linear ordering $(S_1 \cup X, \ldots, S_k \cup X)$.*

Let us now state and prove our characterisation theorem. As we mentioned earlier, the constructive proof we provide will also constitute part of our algorithm: it gives us a minimal interval model of $G[w^*] + x$ from which we extract a model of the representative graph of w'_s, which is the new prime node created in T^{m} after the insertion of x. Later, in the algorithmic part of the paper, we show how to implement the construction of the proof below in $O(n)$ total running time.

Theorem 10. *$G + x$ is an interval graph if and only if the conditions of Lemmas 20 to 23 are satisfied.*

Proof. We show that if the necessary conditions of Lemmas 20 to 23 are satisfied, then $G[w^*]+x$ admits a consecutive ordering, which, from Theorem 9, is enough for $G+x$ itself to be an interval graph. To that purpose, we will first show (Proposition 1) that the mixed nodes of $T_w^{\mathrm{c}} \setminus \{w\}$ satisfy property $\mathcal{P}$ below, then, we will use this property to build a consecutive ordering of $G[w^*] + x$ (Proposition 2).

Property $\mathcal{P}(u)$: $G[u^*] + x$ admits a consecutive ordering whose last clique contains x, and if B_u is not full, then no other maximal clique contains x.

Remark 12. *Every full node u satisfies Property $\mathcal{P}$. Moreover, if u is full and B_u is not, then u^* is a clique.*

Proof of Remark 12: For a full node u, all the maximal cliques of $G[u^*] + x$ contain x, so does the last one of any consecutive ordering. In addition, if B_u is not full, since the conditions of Lemma 20 are satisfied, then u has at most one linked child. But any internal node has at least two children c_1, c_2 such that $c_1^* \neq \varnothing$ and $c_2^* \neq \varnothing$. Then, since u is full, it is not an internal node but a leaf. Thus, u^* is a clique and u satisfies Property $\mathcal{P}$. □

Proposition 1. *If the conditions of Lemmas 20 to 22 are satisfied, then every mixed node $u \in T_w^c \setminus \{w\}$ satisfies property $\mathcal{P}$.*

Proof of Proposition 1: When a mixed node u satisfies property $\mathcal{P}$, we denote σ^{u+x} for a consecutive ordering of $G[u^*] + x$ whose last clique contains x. If u is uniform and $u^* \neq \varnothing$, we denote σ^u for a consecutive ordering of $G[u^*]$, and if $u^* = \varnothing$, σ^u denotes for $(\varnothing)$, i.e. the order on one unique element which is the empty set.

We use an induction to show that Property $\mathcal{P}$ holds for the mixed nodes of $T_w^c \setminus \{w\}$. Let $u \in T_w^c \setminus \{w\}$ be a mixed node such that every mixed child of u satisfies $\mathcal{P}$. Let us show that u satisfies $\mathcal{P}$.

Let us first consider the case where u is a degenerate node. Since u is degenerate, any child u_c of u is such that $u_c^* \neq \varnothing$. Then, the children of u are unambiguously partitioned by $\mathcal{C}_h(u)$, $\mathcal{C}_m(u)$ and $\mathcal{C}_f(u)$. Since the conditions of Lemma 22 are satisfied, $\mathcal{C}_m(u)$ contains at most one element q (possibly none) and from induction hypothesis $G[q^*] + x$ has a consecutive ordering σ^{q+x} whose last clique contains x; and σ^{q+x} denotes for $\perp$ if $\mathcal{C}_m(u) = \varnothing$.
If B_u is full, then $\sigma^{u+x} = \sum_{v \in \mathcal{C}_h(u)} \sigma^v_{<X_u>} + \sigma^{q+x} + \sum_{v \in \mathcal{C}_f(u)} \sigma^v_{<X_u \cup \{x\}>}$ is a consecutive ordering of $G[u^*] + x$ whose last clique contains x. Thus, property $\mathcal{P}(u)$ holds.
If B_u is not full, the conditions of Lemma 20 implies that u has at most one linked child.

- If u has one full child u_1, since B_{u_1} is not full, then, as we showed previously, $u_1^* \cup \{x\}$ is a clique. If $X_u \setminus N(x) \neq \varnothing$, we denote K_1 for $u_1^* \cup X_u$, and $K_1 = \perp$ otherwise. Then, $\sigma^{u+x} = \sum_{v \in \mathcal{C}_h(u)} \sigma^v_{<X_u>} + K_1 + u_1^* \cup (X_u \cap N(x)) \cup \{x\}$ is a consecutive ordering of $G[u^*] + x$ whose last clique is the only one containing x.

- If u has no full child but one mixed child. From induction hypothesis, u_1 admits a consecutive ordering $\sigma^{u_1} = \sigma^{\text{beg}} + K_x$ whose last clique K_x is the only one containing x. Moreover, if $K_x \setminus \{x\}$ is maximal in $G[u^*]$ and $X_u \setminus N(x) \neq \varnothing$, then we denote K_1 for $(K_x \setminus \{x\}) \cup X_u$ and $K_1 = \perp$ otherwise. Then $\sigma^{u+x} = \sum_{v \in \mathcal{C}_h(u)} \sigma^v_{<X_u>} + \sigma^{\text{beg}}_{<X_u>} + K_1 + K_x \cup (X_u \cap N(x))$ is a consecutive ordering of $G[u^*] + x$ whose last clique is the only one containing x.

- If u has only hollow children, since u is mixed, then $X_u \cap N(x) \neq \varnothing$. $\sigma^{u+x} = \sum_{v \in \mathcal{C}_h(u)} \sigma^v_{<X_u>} + (X_u \cap N(x)) \cup \{x\}$ is a consecutive ordering of $G[u^*] + x$ whose last clique is the only one containing x.

We now deal with the case where u is a prime node. We denote α for l_u when u satisfies Condition 1 of Lemma 22, and for $\max(\sigma_u)$ when u satisfies Condition 2 of Lemma 22. In any case, up to reversing σ_u, the nodes before α in σ_u are hollow, the nodes after α are saturated and α is not saturated and satisfies the left property.

- Let us first consider the case where B_u is full. Note that α may be the first or last child of u and that, in that case, thanks to the conventions we adopted, what follows remains valid.

  If α is hollow, let $\sigma^{u+x} = \sum_{v \leq \alpha} \sigma^v_{<\Delta_v \cup X_u>} + K_{x\alpha} + \sum_{v > \alpha} \sigma^v_{<\Delta_v \cup X_u \cup \{x\}>}$, where $K_{x\alpha} = (\Delta_\alpha \cap N(x)) \cup X_u \cup \{x\}$ if there exists $y \in Y_u \cap N(x)$ such that $\mathrm{e}_y^2 = \alpha$ and $K_{x\alpha} = \varnothing$ otherwise. Then σ^{u+x} shows that $\mathcal{P}(u)$ is true.

  If α is not hollow, we distinguish two cases.

  - If B_α is full, then α is not full and $\sigma^{u+x} = \sum_{v < \alpha} \sigma^v_{<\Delta_v \cup X_u>} + \sigma^\alpha_{<\Delta_\alpha \cup X_u>} + \sum_{v > \alpha} \sigma^v_{<\Delta_v \cup X_u \cup \{x\}>}$ is a consecutive ordering of $G[u^*] + x$ whose last clique contains x.

- If B_α is not full then, from induction hypothesis, we denote $\sigma^{\alpha+x} = \sigma^{\text{beg}} + K_x$ for a consecutive ordering of $G[\alpha^*]+x$ whose only clique containing x is K_x. And we denote $K_{\bar{x}u} = (K_x \setminus \{x\}) \cup \Delta_\alpha \cup X_u$ if $K_x \setminus \{x\}$ is maximal in $G[\alpha^*]$ and $K_{\bar{x}u} = \perp$ otherwise. And we denote $K_{xu} = K_x \cup (\Delta_\alpha \cap N(x)) \cup X_u$. Then, $\sigma^{u+x} = \sum_{v<\alpha} \sigma^v_{<\Delta_v \cup X_u>} + \sigma^{\text{beg}}_{<\Delta_\alpha \cup X_u>} + K_{\bar{x}u} + K_{xu} + \sum_{v>\alpha} \sigma^v_{<\Delta_v \cup X_u \cup \{x\}>}$ is a consecutive ordering of $G[u^*]+x$ whose last clique contains x.

- If B_u is not full, again, we distinguish two cases.

  - If α is hollow, then $\sigma^{u+x} = \sum_{v<\alpha} \sigma^v_{<\Delta_v \cup X_u>} + K_{x\alpha}$, where $K_{x\alpha} = ((\Delta_\alpha \cup X_u) \cap N(x)) \cup \{x\}$, is a consecutive ordering of $G[u^*]+x$ whose last clique is the only one containing x. Thus u satisfies Property $\mathcal{P}$.
  - When α is linked, since B_u is not full, neither is B_α and, from induction hypothesis, we denote $\sigma^{\alpha+x} = \sigma^{\text{beg}} + K_x$ for a consecutive ordering of $G[\alpha^*]+x$ whose only clique containing x is K_x. We also denote $K_{\bar{x}\alpha} = (K_x \setminus \{x\}) \cup \Delta_\alpha \cup X_u$ if $K_x \setminus \{x\}$ is maximal in $G[\alpha^*]$ and $(\Delta_\alpha \cup X_u) \setminus N(x) \neq \varnothing$, and we denote $K_{\bar{x}\alpha} = \varnothing$ otherwise. In addition, we denote $K_{x\alpha} = K_x \cup ((\Delta_\alpha \cup X_u) \cap N(x))$. Then, $\sigma^{u+x} = \sum_{v<\alpha} \sigma^v_{<\Delta_v \cup X_u>} + \sigma^{\text{beg}}_{<\Delta_v \cup X_u>} + K_{\bar{x}\alpha} + K_{x\alpha}$ is a consecutive ordering of $G[u^*]+x$ whose last clique is the only one containing x. Thus, $\mathcal{P}(u)$ is true.

This ends the proof of Proposition 1. □

Proposition 2. *If the conditions of Lemma 23 are satisfied and if the mixed children of w satisfy Property $\mathcal{P}$, then $G[w^*]+x$ is an interval graph.*

Proof of Proposition 2: We build a consecutive ordering of $G[w^*]+x$, denoted σ^{w+x}, as we did for the mixed[7] nodes u of $T^c_w \setminus \{w\}$. Since w satisfies the conditions of Lemma 20, it has at most two mixed children v_1 and v_2.

Let us first consider the case where w is a degenerate node. If B_w is full, then $\sigma^{w+x} = \sum_{v \in \mathcal{C}_h(w)} \sigma^v_{<X_w>} + \sigma^{v_1+x}_{<X_w>} + \sum_{v \in \mathcal{C}_f(w)} \sigma^v_{<X_w \cup \{x\}>} + \bar{\sigma}^{v_2+x}_{<X_w>}$ is a consecutive ordering of $G[w^*]+x$. If B_w is not full, then w has at most one mixed child and we obtain σ^{w+x} exactly in the same way as we obtained σ^{u+x} for a degenerate mixed node $u \in T^c_w \setminus \{w\}$.

We now deal with the case where w is a prime node.

- The case where B_w is full and the set S_w of the saturated children of w is not empty is very similar to the corresponding case for a prime node $u \in T^c_w \setminus \{w\}$. The main difference is that S_w does not need to include an extremal element of σ_w. As a consequence, there are two possibly linked non-saturated children of w, namely l_w and r_w. l_w plays exactly the same role as l_u while r_w plays the symmetric role. Then the case by case analysis led on l_u (depending on whether l_u is linked or not and whether B_{l_u} is full or not) has to be led on both l_w and r_w, resulting in numerous cases. Since these cases are all identical to the cases previously treated for u, we just give as an example the case where S_w contains no extremal element of σ_w, l_w and r_w are mixed, and B_{l_w} and B_{r_w} are full. Since l_w satisfies Property $\mathcal{P}$, $G[l^*_w]+x$ admits a consecutive ordering σ^{l_w+x} such that its last clique contains x; and r_w satisfies the same property. Then, $\sigma^{w+x} = \sum_{v<l_w} \sigma^v_{<\Delta_v \cup X_w>} + \sigma^{l_w+x}_{<\Delta_{l_w} \cup X_w>} + \sum_{l_w<v<r_w} \sigma^v_{<\Delta_v \cup X_w \cup \{x\}>} + \bar{\sigma}^{r_w+x}_{<\Delta_{r_w} \cup X_w>} + \sum_{v>r_w} \sigma^v_{<\Delta_v \cup X_w>}$, where $\bar{\sigma}$ denotes for the reverse order of σ, is a consecutive ordering of $G[w^*]+x$.

[7] Remember that, by definition, w is mixed.

- In the case where B_w is full and $S_w = \varnothing$, we have to distinguish two cases.
  - If w satisfies Condition 2b of Lemma 23, the case is strictly similar to the one of a node $u \in T_w^{\mathrm{c}} \setminus \{w\}$ satisfying Condition 2 of Lemma 22, and the construction made in that later case gives a consecutive ordering of $G[w^*] + x$ in the present case.
  - The case where w satisfies Condition 2a of Lemma 23 is a particular case of the case where $S_w \neq \varnothing$. In the construction made previously, simply remove the cliques of $\sum_{v \in \mathcal{C}_f(w)} \sigma^v_{<\Delta_v \cup X_w \cup \{x\}>}$, since $\mathcal{C}_f(w) = \varnothing$, to obtain a consecutive ordering in the present case. The condition that no vertex $y \in Y_w \setminus N(x)$ covers both l_w and r_w guaranties that the ordering of the cliques you obtain contains all maximal cliques of $G[w^*] + x$. In other words, it prevents $K = X_w \cup (\Delta_{l_w} \cap \Delta_{r_w})$ to be maximal in $G[w^*] + x$; which could compromise the consecutiveness constraint of x, since K does not contain x and would necessarily be between the maximal cliques K' of G' such that $K' \setminus \{x\}$ is a leaf of $T^c_{l_w}$, some of which do contain x, and those such that $K' \setminus \{x\}$ is a leaf of $T^c_{r_w}$, some of which also contain x.
- The case where B_w is not full is identical to the one of a node $u \in T_w^{\mathrm{c}} \setminus \{w\}$ satisfying Condition 2 of Lemma 22, and the construction made in that later case gives a consecutive ordering of $G[w^*] + x$ in the present case.

Thus, in every case, we can build a consecutive ordering of $G[w^*] + x$ which then turns out to be an interval graph. This completes the proof of Proposition 2. □

Finally, if the conditions of Lemmas 20 to 23 are satisfied, then Proposition 1 implies that the mixed children of w satisfy Property $\mathcal{P}$ and it follows from Proposition 2 that $G[w^*] + x$ is an interval graph. Thus, from Theorem 9, $G + x$ itself is an interval graph, which achieves the proof of Theorem 10. □

Thanks to Theorem 10, we know how to test whether the insertion of a vertex results in an interval graph or not. And in the positive, the proof also gives us an interval model of $G[w^*] + x$. From this model, we can derive a model for the representative graph of new prime node w'_s created in T^{m} after the insertion of x. Since we already determined in Section 5.1 what is the rest of tree $T^{\mathrm{m}}(G')$, we can now describe the algorithm maintaining T^{m} under vertex insertion and analyse its complexity.

5.3. Algorithm and complexity

We now present the algorithm for vertex insertion. As we noticed earlier, since the three representations we aim to maintain are linear-time equivalent, we can maintain only one of them and get the two others within the same $O(n)$ time complexity thanks to the transformations described in Sections 3.2.2 and 3.2.3. For vertex insertion operations, we chose to focus on the maintain of the MD-representation, but, in fact, we both use the MD-representation and the PQ-representation. The reason is that, for sake of simplicity, we separate in the algorithm the maintain of the tree and the decision whether the augmented graph is still an interval graph.

It turns out that maintaining the MD-representation under vertex insertion has already been done in [36, 24] for general graphs. Thus we are able to directly use these results to get the shape of the MD-representation of the augmented graph. Roughly speaking, the new MD-tree is the same as the old one, except that a part of the tree, rooted at the insertion node w_m (see Section 5.1), collapses in one single prime node, named w'_s in this paper. Actually, [36, 24] determine the whole MD-representation of the augmented graph, except the quotient

of w'_s. And the main constraint so that $G+x$ is an interval graph is precisely that the quotient of this new prime node w'_s is itself an interval graph.

Testing this later condition is more convenient with the PQ-representation. Indeed, this representation, which is based on the notion of consecutive ordering of the cliques, offers a more direct way to check that consecutiveness constraints introduced by the new incoming vertex x can integrate the current constraints imposed by the vertices of G in such a way that the new graph $G+x$ is an interval graph. The purpose of Section 5.2 was precisely to give necessary and sufficient conditions on the way x touches the PQ-representation of G so that $G+x$ is an interval graph. Our proof of Theorem 10 is constructive : it shows how to build an interval model when the necessary conditions we gave in Lemmas 20 to 23 are satisfied. We make use of this construction in the algorithm to build an interval model of the quotient of the new prime node w'_s created by the insertion of x, thus completing the MD-representation.

Our algorithm is in three steps. The first step, called the *marking step*, collects some information about T^{m} and T^{c}, and finds the key node w (introduced in Section 5.2.1), which is the node of T^{c} corresponding to the insertion node w_m of T^{m}. The second step, called the *testing step*, checks whether T^{c}_w satisfies the conditions of Lemmas 20 to 23, that is whether $G+x$ is an interval graph. In the positive, the third and last step, called the *insertion step*, updates $MD(G)$ by building $MD(G')$.

5.3.1. Some useful routines

In this subsection we describe two routines that we use in the marking step and in the testing step of our algorithm. Routine `FindTwin` checks whether a vertex x to be inserted in a prime interval graph G, given by an interval model, has a twin in $G+x$ and it finds it in the positive. We use it in the marking step to find the insertion node w_m in T^{m}. Routine `PartNonCov` computes the subset of elements of a linear order that are not covered by a family of intervals of this linear order. We use it in the marking step to determine the types associated to branches of nodes of T^{c}, that is, whether these branches are full, hollow or mixed. We also use it twice in the testing step : once to check whether the saturated children of each prime node form an interval, and once to check whether there exists some couple (l,r) of children of node w that satisfies Condition 2a of Lemma 23.

In the following, for an interval I of a linear ordering σ, we denote $l(I)$ and $r(I)$ for respectively the left and right bound of I in σ ($l(I)$ is less than or equal to $r(I)$ in σ).

Routine ***`FindTwin`***. Let G be a prime interval graph given with an interval model σ, and let x be a vertex to be inserted in G. Routine `FindTwin` determines whether x has a twin in $G+x$, and outputs it in the positive, with a total complexity of $O(n)$ time, where n is the number of vertices of G. Note that, since G is prime, x has at most one twin in $G+x$.

For any vertex a of G, we denote I_a the interval of a in σ. We can identify a unique candidate y to be the twin of x by computing $l_x = min\{r(I_z) \mid z \in N(x)\}$ and $r_x = max\{l(I_z) \mid z \in N(y)\}$. We now show that if x is the twin of some vertex y of G, then $l_x = l(I_y)$ and $r_x = r(I_y)$. Indeed, since all the neighbours of x are also neighbours of y, and since x has some neighbour whose interval right bound is l_x, then $l(I_y) \leq_\sigma l_x$. Moreover, from Lemma 6, there is an interval whose right bound is precisely $l(I_y)$. Since the vertex z corresponding to this interval is adjacent to y, and since y and x are twins, then z is adjacent to x. It follows that $l(I_y) \leq_\sigma l_x$, and finally we obtain $l_x = l(I_y)$. And by symmetry, we also have $r_x = r(I_y)$.

Consequently, to check whether x has a twin, we compute l_x and r_x by a simple parse of $N(x)$, which takes $O(n)$ time. Then, by parsing the vertices of G, we check whether there is some vertex y of G such that $l(I_y) = l_x$ and $r(I_y) = r_x$. In the negative, x has no twin. In the positive, we check whether y, which is necessarily unique, has the same neighbours as x among

the vertices of G different from y. We can do so by parsing the vertices of G one more time. Thus, we can determine whether x has a twin in $G + x$, and finds it, in $O(n)$ time.

Routine `PartNonCov`$(\sigma, \mathcal{I})$. Let σ be a linear ordering on a set S, and let $\mathcal{I}$ be a subset of intervals of σ. We define $\mathcal{NC}(\sigma, \mathcal{I})$ as the subset of elements of S that do not belong to any of the intervals in $\mathcal{I}$, i.e. $\mathcal{NC}(\sigma, \mathcal{I}) = S \setminus \bigcup_{I \in \mathcal{I}} I$. For any subset $S_1 \subseteq S$, the set of intervals I of σ such that $I \subseteq S_1$ and I is maximal for inclusion forms a partition of S_1, which we call the *interval partition* of S_1. Given σ and $\mathcal{I}$, Routine `PartNonCov` outputs the interval partition of $\mathcal{NC}(\sigma, \mathcal{I})$ with its intervals sorted by increasing order wrt. σ (recall that these intervals do not intersect). It proceeds as described below.

First, we bucket sort the intervals $I \in \mathcal{I}$ with key $l(I)$. It takes $O(|S| + |\mathcal{I}|)$ time. We denote π for the resulting order on $\mathcal{I}$. We then parse $\mathcal{I}$ according to π and, at each step, we maintain the interval partition $\mathcal{P}_i$ of the elements of S that do not belong to any of the first i intervals of π, in the following way.

Initially, partition $\mathcal{P}_0$ contains a single interval which is σ itself. Along the process, $\mathcal{P}_i$ is a list whose elements are pairs $(l(J), r(J))$ for every interval J in the partition, the list being sorted with increasing $l(J)$ (and so with increasing $r(J)$ since the intervals do not intersect). The process stops either when the last (according to order π) interval in $\mathcal{I}$ has been examined, or when the rightmost element $\max(\sigma)$ of σ leaves the partition: then, $\mathcal{P}_i$ is the interval partition of $\mathcal{NC}(\sigma, \mathcal{I})$. We now show that updating $\mathcal{P}_i$ to get $\mathcal{P}_{i+1}$, when examining the $i + 1^{th}$ interval $I \in \mathcal{I}$, takes only constant time. Thus, the time needed for parsing $\mathcal{I}$ and maintaining $\mathcal{P}_i$ is $O(|\mathcal{I}|)$.

We claim the two following invariants, whose proof is straightforward (and omitted). When the routine has examined the i first elements of π and is about to examine interval I which is the $i + 1^{th}$, we have:

1. The rightmost element $\max(\sigma)$ of σ belongs to $\mathcal{P}_i$; and
2. I does not intersect, and is on the right of, any interval of $\mathcal{P}_i$ except possibly the rightmost one.

Property 2 is a consequence of the fact that we examine the intervals $I \in \mathcal{I}$ with increasing $l(I)$. From this property, it follows that, treating I, the only elements of the intervals of $\mathcal{P}_i$ it may contain are necessarily in the rightmost interval I_{last} of $\mathcal{P}_i$. Then, we get $\mathcal{P}_{i+1}$ by dividing I_{last} into two new (possibly empty) intervals, namely $I_{last} \cap [\![\min(\sigma), l(I)]\!]$ and $I_{last} \cap [\![r(I), \max(\sigma)]\!]$. This takes constant time.

Thus, Routine `PartNonCov` computes the interval partition of $\mathcal{NC}(\sigma, \mathcal{I})$ in $O(|S| + |\mathcal{I}|)$ time.

5.3.2. Marking step

During this step, we determine for each node of T^{m} and T^{c} and for the branches of the nodes of T^{c}, whether they are full, mixed or hollow. We call this piece of information the *type* of nodes and branches. These types are at the core of our characterisation of successful insertions. Computing them once and for all allows us to save a great amount of time in the rest of the algorithm, and to achieve the desired $O(n)$ time complexity. We also identify the insertion node w_m of T^{m}.

We first determine for each node u of T^{m} whether it is full, mixed or hollow, which we call the *type* of u. We proceed by a well-known bottom-up marking process of the tree which is exactly the one of [36], where the marks 1, -1 and 0 have been replaced by the types $full$, $hollow$ and $mixed$. Each node receives its type in a bottom-up process. A leaf y of T^{m} is typed $full$ if $y \in N(x)$ and $hollow$ otherwise. Each node forwards its type to its parent node. If a node u receives the same type from all its children, then u is given this type. Otherwise, u is

given the type *mixed*. The process ends when the root is given a type. It is clear that the types given to the nodes along this process are the right ones. Since the number of nodes in T^{m} is $O(n)$ (see Section 3.1.3) and since each edge of T^{m} is crossed once, the typing routine runs in $O(n)$ time.

Then, we find the insertion node w_m by following a path from the root to w_m. While the node u visited on this path is proper (see Definition 6), we visit its unique mixed child. The first non-proper node u_{np} found is precisely w_m. Indeed, From Definition 6, it follows that all the descendants of a proper node that are not descendant of its unique mixed child are either full or hollow, and thus proper. Then, the first non-proper node u_{np} found on the path is clearly the least common ancestor of all non-proper nodes. Consequently, by definition, u_{np} is the insertion node w_m. This search in T^{m} can be implemented in $O(n)$ time, since testing whether node u is proper can be done as follows. If u is a series node (resp. parallel node), then u is proper iff all its children but one are typed *full* (resp. *hollow*) and the remaining child is *mixed*. If u is a prime node, then u is proper iff it has a unique mixed child γ, and, in G_u, γ is adjacent to $v \in \mathcal{C}(u)$ iff v is full wrt. x. This can be checked by examining all the nodes of $\mathcal{C}(u) \setminus \{\gamma\}$. In any case, testing whether u is a proper node can be done in $O(|\mathcal{C}(u)|)$ time. Thus, finding w_m takes $O(n)$ time.

We also compute the types of the nodes of T^{c}, within the same $O(n)$ time complexity. Remember that the type of a node u of T^{c} refers to the set u^* of vertices of G. We can easily get the type of X_u for each $u \in T^{\mathrm{c}}$ by simply parsing T^{c} and parse the list of vertices in X_u for each node u. This takes $O(n)$ time. Then, in order to get the type of u^*, we just have to determine the type of $u^* \setminus X_u$. This type is precisely the type of the node of T^{m} associated to u (see Definition 5). Thanks to the MD-PQ-transformation (see Section 3.2.3), we obtain the PQ-representation in $O(n)$ time, as well as the correspondences between associate nodes of T^{m} and T^{c}. Since we already computed the types of the nodes of T^{m}, we can deduce the type of node u of T^{c} using the types of X_u and of its associate node in T^{m}. This is done in $O(n)$ time complexity.

Finally, we need to determine for each node u of T^{c} the type of B_u. We proceed by a top-down search of T^{c}. For a child u of v, since $B_u = B_v \cup X_u \cup \Delta_u$ and since these three sets are disjoint, we can deduce the type of B_u from those of B_v, X_u and Δ_u. The type of X_u has been determined previously and when treating node u during the top-down search, we already know the type of B_v. Then, we only have to determine the type of Δ_u. When v is prime, we can determine the type of Δ_u for each child u of v as follows. Thanks to Routine `PartNonCov`, we can compute $\mathcal{NC}(\sigma_v, \mathcal{N})$ and $\mathcal{NC}(\sigma_v, \bar{\mathcal{N}})$, where $\mathcal{N} = \{I_y \mid y \in Y_v \cap N(x)\}$ and $\bar{\mathcal{N}} = \{I_y \mid y \in Y_v \setminus N(x)\}$ and I_y is the interval of y in σ_v. The set of children u of v such that Δ_u is full is $\mathcal{NC}(\sigma_v, \bar{\mathcal{N}})$, its set of children u such that Δ_u is hollow is $\mathcal{NC}(\sigma_v, \mathcal{N})$, and its set of children u such that Δ_u is mixed is $\mathcal{C}(v) \setminus (\mathcal{NC}(\sigma_v, \mathcal{N}) \cup \mathcal{NC}(\sigma_v, \bar{\mathcal{N}}))$. These three sets can be computed in $O(|\mathcal{C}(u)| + |Y_u|)$ time, which is the complexity of `PartNonCov`, and the total complexity of the top-down search for determining the types of the branches of T^{c} is therefore $O(n)$.

Finally, the whole marking step runs in $O(n)$ time.

Treating particular cases

Thanks to the information collected about T^{m} and T^{c}, we are now ready to check whether $G + x$ is an interval graph and to update the MD-representation in the positive. The general case is the one where w_m is uncut and the neighbourhood of $V(w_m)$ in G is a clique. These are the conditions of applications of Lemmas 20 to 23 and Theorem 10. In this paragraph, we treat the remaining cases. In these cases, either $G + x$ is not an interval graph, and the algorithm stops, or $G + x$ is an interval graph and we can easily update the MD-representation without

going through the difficulty of the general case.

First we check whether w_m is cut or not (see Definition 8). When w_m is degenerate, it is cut iff it has no mixed child. When w_m is a prime node, in order to check whether w_m is cut, we first check that it has only uniform children. In the positive, the adjacency relationships between x and vertices of $V(w_m)$ are compatible with the quotient G_{w_m} and consequently, the graph $G_{w_m} + x$ can be naturally defined as $G[V(w_m) \cup \{x\}]/(\mathcal{MSM}(G[V(w_m)]) \cup \{x\})$. From Definition 8, w_m is cut iff there exists a child γ of w_m such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(w_m) \cup \{x\}]$. This condition is equivalent to the fact that there exists a child γ of w_m such that x is a twin of γ in $G_{w_m} + x$, which can be checked thanks to a call to Routine `FindTwin`(x, G_{w_m}). Thus, the test whether w_m is cut takes $O(n)$ time.

If w_m is cut, our algorithm determines whether $G + x$ is an interval graph and updates $MD(G)$ in the positive. From Lemma 11, the fact that $G + x$ is an interval graph can be determined by examining the labels and the types of w_m and its children. In the case where w_m is prime, we also need to check whether its child γ (the one such that $V(\gamma) \cup \{x\}$ is a module of $G'[V(w_m) \cup \{x\}]$) is non-simplicial, which can be done in constant time by checking that the left bound and the right bound of the interval of γ in the quotient G_{w_m} are different. Thus, the conditions of Lemma 11 can be tested in $O(|\mathcal{C}(w_m)|) = O(n)$ time. If $G + x$ is not an interval graph, then the algorithm stops. Otherwise, we update the MD-representation as shown in Section 5.1.1. This requires only to handle children of w_m and to create at most one new node in the tree. It is easy to see that all the modifications of the tree described in Section 5.1.1 can be done in $O(|\mathcal{C}(w_m)|) = O(n)$ time.

In the case where w_m is uncut, we test whether the neighbourhood of $V(w_m)$ is a clique. This test is easy to perform since the neighbourhood of $V(w_m)$ is a clique if and only if w_m is the root of T^{m} or w_m is simplicial in the quotient $G_{\mathrm{parent}(w_m)}$. That is, parent(w_m) is degenerate, or parent(w_m) is prime and w_m belongs to a unique maximal clique of the interval model of $G_{\mathrm{parent}(w_m)}$, which is the case when the left bound and the right bound of w_m in this model are the same. If the neighbourhood of $V(w_m)$ is not a clique, then $G + x$ is not an interval graph and the algorithm stops. Otherwise, we are in the general case where w_m is uncut and the neighbourhood of $V(w_m)$ is a clique. Thus, all the particular cases can be treated in $O(n)$ time.

When we are in the general case mentioned above, the algorithm goes on by finding the key node w. From Definition 9, the key node w is the unique node such that $V(w_m) = w^*$ or $V(w_m) = w^* \setminus X_w$. When w_m is a prime node or a parallel node, from Lemma 8, we are in the case where $V(w_m) = w^* \setminus X_w$ and w is precisely the associate node of w_m in T^{c}. When w_m is series, from Lemma 6, w_m has a unique non-leaf child w_{nl}, which is either prime or parallel, and which is such that its corresponding node w in T^{c} satisfies $V(w_m) = w^*$ (see case 1 of the MD-PQ-transformation). Then, the key node w can be determined in $O(|\mathcal{C}(w_m)|) = O(n)$ time. Once the key node w of T^{c} has been determined, we can check whether the conditions of Lemmas 20 to 23 are satisfied, and update the MD-representation in the positive. These are the purpose of the last two steps of the algorithm, detailed below.

5.3.3. Testing step

Theorem 10 states that $G+x$ is an interval graph iff all the nodes of T^{c}_w satisfy the conditions of Lemmas 20 to 23. Thanks to the information collected about T^{c} in the first step, we can check whether the nodes of T^{c}_w satisfy these conditions in $O(n)$ time as follows.

Testing the conditions of Lemma 20. For a node $u \in T^{\mathrm{c}}_w$, these conditions can be tested in $O(|\mathcal{C}(u)|)$ time by first examining the type of B_u, computed in the marking step, and then performing a simple search of the children of u where we examine their types (remember that a node is linked iff it is not hollow).

Testing the conditions of Lemma 21. The difficulty of checking these conditions for a prime node $u \in T_w^{\mathrm{c}}$ is to decide whether the saturated children of u form an interval S_u. To that purpose, we determine the set $S = \{v \in \mathcal{C}(u) \mid \Delta_v \subseteq N(x)\}$ thanks to a call to $\texttt{PartNonCov}(\sigma_u, \{[\![\mathrm{e}_y^1, \mathrm{e}_y^2]\!] \mid y \in Y_u \setminus N(x)\})$. At the end of the routine, we obtain an interval partition of S. Then, we can check that the full nodes belonging to S form an interval S_u of σ_u. And we can parse the children of u again in order to check that the nodes in $\mathcal{C}(u) \setminus (S_u \cup \{l_u, r_u\})$ are hollow. The whole test takes $O(|\mathcal{C}(u)| + |Y_u|)$ time, and in the case where the test is positive, we also determined S_u in the same complexity.

Testing the conditions of Lemma 22. These conditions apply only for a prime node u, for which we already computed the interval S_u of its saturated children when testing the conditions of Lemma 21. Then, the key property to test, in the conditions of Lemma 22, is to determine whether a given child v of u satisfies one of the left or right property: this can be done very easily by scanning the vertices $y \in Y_u$ and by comparing the positions of e_y^1 and e_y^2 to the position of v in σ_u. This takes $O(|Y_u|)$ time and we have to perform this test at most twice: for l_u and r_u in Case 1, or for the first node and the last node of σ_u in Case 2. The rest of the conditions of Case 2 can be tested simply by examining the types of the children of u. Then, the whole complexity of testing conditions of Lemma 22 is $O(|\mathcal{C}(u)| + |Y_u|)$.

Testing the conditions of Lemma 23. These conditions apply only to node w and they are very similar, except Condition 2a, to the conditions of Lemma 22, which have been showed above to be testable in $O(|C(w)| + |Y_w|)$ time. Consequently, we now concentrate on Condition 2a and we show how to identify a set of candidates for being the couple l, r. Let $w_1 = \max_{\sigma_w}\{\mathrm{e}_y^1 \mid y \in Y_w \cap N(x)\}$ and $w_2 = \min_{\sigma_w}\{\mathrm{e}_y^2 \mid y \in Y_w \cap N(x)\}$. The children of w satisfying the left property are exactly the nodes $v \leq_{\sigma_w} w_2$, and those satisfying the right property are the nodes $v \geq_{\sigma_w} w_1$. Let w_1^- be the predecessor of w_1 in σ_w. If $w_2 < w_1^-$ then no couple (l, r) of consecutive nodes of σ_w is such that l satisfies the left property and r satisfies the right property. In this case, Condition 2a of Lemma 23 is not satisfied. Otherwise, i.e. if $w_1^- \leq w_2$, set $A_w = \{(l, r) \in \mathcal{C}(w)^2 \mid w_1^- \leq l \leq w_2 \text{ and } r \text{ is the successor of } l\}$ is exactly the set of couples (l, r) such that l satisfies the left property and r satisfies the right property. We denote π the linear order over A_w induced by order σ_w on the first component l of couples $(l, r) \in A_w$. For any interval I of σ_w, we define $tr(I) = \{(l, r) \in A_w \mid \{l, r\} \subseteq I\}$. Clearly, $tr(I)$ is an interval of π. We denote $\mathcal{I} = \{tr([\![\mathrm{e}_y^1, \mathrm{e}_y^2]\!]) \mid y \notin N(x)\}$. By definition, couples (l, r) satisfying Condition 2a of Lemma 23 are all in $\mathcal{NC}(\pi, \mathcal{I})$, which can be computed by a call to $\texttt{PartNonCov}(\pi, \mathcal{I})$ in $O(|A_w| + |\mathcal{I}|) = O(|\mathcal{C}(w)| + |Y_w|)$ time. There is an additional condition for a couple (l, r) to satisfy Condition 2a: all the nodes of $\mathcal{C}(w) \setminus \{l, r\}$ must be hollow. Then, if all the children of w are hollow, it follows that any of the couple (l, r) belonging to $\mathcal{NC}(\pi, \mathcal{I})$ satisfies Condition 2a. If there is only one linked node v in σ_w, we have to check that at least one of the two couples (l, r) containing v is in $\mathcal{NC}(\pi, \mathcal{I})$. And if there are exactly two consecutive linked nodes v_1, v_2 in σ_w, we must check that (v_1, v_2) belongs to $\mathcal{NC}(\pi, \mathcal{I})$. In all other cases, Condition 2a is not satisfied. Thus, we can check whether Condition 2a is satisfied and, in the positive, determine a couple (l, r) satisfying it in $O(|\mathcal{C}(w)| + |Y_w|)$ time.

Finally, since all the conditions of Lemmas 20 to 23 can be tested for a node $u \in T_w^{\mathrm{c}}$ in $O(|\mathcal{C}(u)| + |Y_u|)$ time[8], then, by an arbitrary traversal of T_w^{c}, we can determine whether $G + x$ is an interval graph in $O(\sum_{u \in T_w^{\mathrm{c}}} |\mathcal{C}(u)| + |Y_u|) = O(n)$ time.

[8]One may be surprised that this complexity does not depend on $|X_u|$. But the vertices of X_u have already been examined and counted in the complexity in Step 1, when determining for each node u of T^{c} whether B_u is full or not.

5.3.4. Insertion step

If $G + x$ is not an interval graph, then the algorithm stops. Otherwise, the three representations are updated. As we said previously, thanks to the linear-time equivalence between the three structures, we only need to show how to maintain the MD-representation in $O(n)$ time. If w_m is cut (which has been tested during the marking step of the algorithm), the updates to be performed on T^{m} are described in Section 5.1.1. From the algorithmic point of view, as noted in the "Treating particular cases" paragraph of Section 5.3.2, it is straightforward to see that these updates can be handled in $O(n)$ time.

Consider now the general case where w_m is uncut. The modifications of T^{m} in this case have been described in Section 5.1.2 (Theorem 8). Roughly speaking, Theorem 8 states that after the insertion of x, the part of $T^{\mathrm{m}}(G)$ rooted at w_m collapses into a new prime node w'_s (see Notation 5), whose children are completely characterised by Theorem 8. Thus, in order to obtain $MD(G')$, two main tasks have to be accomplished:

1. compute the trees rooted at the children of w'_s in $T^{\mathrm{m}}(G')$,
2. compute an interval model of the quotient $G'[V(w'_s)]/\mathcal{MSM}(G'[V(w'_s)])$ of node w'_s in $MD(G')$.

For the first task, from Theorem 8, the maximal strong modules of $G'[V(w'_s)]$ (i.e. the children of w'_s in $T^{\mathrm{m}}(G')$) are the singleton $\{x\}$ and the maximal uniform modules of $G[W_s]$. These modules are either the union of uniform children of the same type (linked or not linked) of a mixed degenerate node of $T^{\mathrm{m}}_{w_m}$ or a uniform child of a mixed prime node of $T^{\mathrm{m}}_{w_m}$. Since the types of the nodes of $T^{\mathrm{m}}(G)$ have been already determined in the marking step of our algorithm, we can find all these maximal uniform modules M by an arbitrary traversal of $T^{\mathrm{m}}_{w_m}$ in $O(n)$ time. Note that once such a module M is found, it comes together with $T^{\mathrm{m}}(G[M])$ which is simply the restriction of $T^{\mathrm{m}}(G)$ to M. More precisely, if M is a uniform child u of a mixed prime node then $T^{\mathrm{m}}(G[M]) = T^{\mathrm{m}}_u$, and if M is the union of uniform children $u_1, \ldots, u_k$ of a degenerate node u, then $T^{\mathrm{m}}(G[M])$ is formed by the trees $T^{\mathrm{m}}_{u_1}, \ldots, T^{\mathrm{m}}_{u_k}$ linked by a parent node having the same label as u.

Then, $T^{\mathrm{m}}(G')$ is obtained from $T^{\mathrm{m}}(G)$ by replacing node w_m by the new prime node w'_s for which we have just shown above how to compute its children trees. Note that, according to Section 5.1.2, when w_m is a series (resp. parallel) node such that $F_f(w_m) \neq \varnothing$ (resp. $F_h(w_m) \neq \varnothing$), there is some additional minor rearrangement of the tree around w_m, which can be very easily handled in $O(n)$ time as well. Finally, the total computation time of $T^{\mathrm{m}}(G')$ from $T^{\mathrm{m}}(G)$ is $O(n)$.

Thus, in order to complete the computation of $MD(G')$, we only have to determine an interval model of the quotient graph $G_{w'_s}$ of the new prime node w'_s (remind that we have tested in the previous step that G' is an interval graph). We achieve this goal in two steps: first, we compute a minimal interval model of $G[w^*] + x$, and then, we extract from it a minimal model of $G_{w'_s}$.

Let us first show that $G[w^*] + x$ indeed contains a model of $G_{w'_s}$ and how to extract it in $O(n)$ time. From Notation 5, $W_s \subseteq V(w_m)$, and from Definition 9, $V(w_m) \subseteq w^*$. It follows that $V(w'_s) = W_s \cup \{x\} \subseteq w^* \cup \{x\}$. Then, if we have a minimal interval model of $G[w^*] + x$, we get a minimal interval model of $G_{w'_s}$ as follows. We first remove from this model all the vertices of $w^* \setminus W_s$. Then, we keep in the model one single representative vertex in $V(u')$ for each child u' of w'_s in $T^{\mathrm{m}}(G')$. Since we already computed the subtrees $T^{\mathrm{m}}_{u'}$ of $T^{\mathrm{m}}(G')$, this can be done in constant time for each u', by simply choosing the leftmost leaf of T^{m}_u. Then, now that we obtained a model on the desire set of vertices, we parse this model in order to remove the cliques that are no longer maximal: these cliques are precisely those which have no left pointer or no right pointer from any remaining vertex. This clearly takes $O(n)$ time.

Thus, the only task left in order to be able to compute a minimal interval model of $G_{w'_s}$ is to compute a minimal interval model of $G[w^*]+x$. To that purpose, we can follow the constructive proof of Theorem 10 : it shows how to build a consecutive ordering of $G[w^*]+x$ by a bottom-up traversal of T^c_w. At each step, we build a consecutive ordering of $G[u^*]+x$ for the current node u in T^c_w by concatenating the orderings computed for the children of u, and assigning their correct pointers to the vertices of X_u,Y_u and x. The concatenation of the orderings takes $O(\mathcal{C}(u))$ time, as we simply have to concatenate the lists constituting these orderings. Note that for complexity issues, we do not number the cells of the resulting list. We will compute this numbering only at the end of the process when we obtain the consecutive ordering of $G[w^*]+x$.

It is worth noticing that, actually, according to the proof of Theorem 10, this bottom-up concatenation process has to be done only for mixed nodes u of T^c_w, which are spread on two branches rooted at w according to Lemma 20. Indeed, for the uniform children v of the current mixed node u, the constructive proof of Theorem 10 only requires a consecutive ordering of $G[v^*]$ which can be obtained directly from T^c_v; the reason for this being that, for a uniform node v, a consecutive ordering of $G[v^*]$ and $G[v^*]+x$ are essentially the same.

At each step, considering a mixed node u in T^c_w, we have to assign the correct pointers for the vertices of X_u, Y_u and x. The way to do so is entirely described in the proof of Theorem 10, we will not describe it again completely, but it is rather easy to see that it can be achieved in $O(|X_u|+|Y_u|)$ time. Roughly speaking, we have to do the following:

- the vertices of X_u are involved in (almost) all the maximal cliques and are then assigned pointers on the first and last clique of the consecutive ordering of $G[u^*]+x$,
- when u is prime, the vertices y of Y_u are assigned pointers to the first clique of the consecutive ordering of $G[v_1^*]$ and to the last clique of the consecutive ordering of $G[v_2^*]$, where $v_1 = e_y^1$ and $v_2 = e_y^2$,
- x is assigned a pointer to the first clique containing x in the consecutive ordering of $G[v^*]+x$ if u has a mixed child v (already computed in the bottom-up concatenation process), or to the first clique of the consecutive ordering of the first full child v of u otherwise; and the second pointer of x is on the last clique of the consecutive ordering of $G[u^*]+x$.

Note that in the case where u is prime, we don't have to re-order its children. On the other hand, if u is degenerate, then we re-order its children starting with the hollow ones, then the possible mixed one and finally the full ones; this takes $O(|\mathcal{C}(u)|)$ time thanks to the information on the nodes computed in the marking step. Then, each assignment of pointers described above is made in constant time, which results in a total complexity of $O(|\mathcal{C}(u)|+|X_u|+|Y_u|)$ time for assigning their pointers to the vertices of X_u, Y_u and x.

It must be clear that the assignments of pointers described above constitute only a rough general scheme that has to be tempered by many particular cases, all described in extension in the proof of Theorem 10. In particular there may be the creation of an extra maximal clique on the side of the interval of x. In this case, we may need to differentiate the vertices of $X_u \cap N(x)$ from those of $X_u \setminus N(x)$, which will be assigned different pointers. To that purpose, we can simply parse the vertices of X_u and separate those that are adjacent to x from those that are not, this takes $O(|X_u|)$ time. A similar situation may also occur with the node denoted α in the proof of Theorem 10, when u is a prime node. In this case, we have to separate the vertices of $\Delta_\alpha \cap N(x)$ from those of $\Delta_\alpha \setminus N(x)$. This can be done by first parsing the vertices y of Y_u and determine which ones belong to Δ_α by examining the pointers of y on the children of u and using the local numbering on the children of u in $T^c(G)$, and then separate those that are adjacent to x from those that are not, this takes $O(|Y_u|)$ time. Thus, these particular cases, as

well as the others, can still be treated within the $O(|\mathcal{C}(u)| + |X_u| + |Y_u|)$ time complexity. And consequently, the whole processing of T_w^c takes $O(n)$ time and gives a minimal interval model of $G[w^*] + x$.

Finally, putting everything together, we conclude that the total computation time of $MD(G')$ is $O(n)$. And since $PQ(G')$ and a minimal interval model of G' can be obtained from $MD(G')$ in $O(n)$ time, then all the three structures we aim at maintaining for G' can be obtained within this complexity.

6. Conclusion

We have just shown that it is possible to maintain a minimal interval model, the PQ-representation and the MD-representation of a dynamically changing interval graph within $O(n)$ time complexity, under addition or deletion of an edge or a vertex (along with the edges defining its neighbourhood). In addition, we also showed the algorithmic equivalence of the three structures we maintain, which implies that any algorithm taking one of these structures as input or producing one of them as output can equivalently work with any of the two other structures and achieve the same time complexity, provided that the complexity of the algorithm is at least $O(n)$.

The $O(n)$ time complexity of our dynamic algorithm is to be compared with the $O(n + m)$ time complexity of static recognition algorithms for interval graphs, which is also the best known complexity for building an interval model, the PQ-representation and the MD-representation from the adjacency lists of the graph. It shows that these representations of interval graphs can be efficiently maintained dynamically when the graph is subject to elementary modifications. As we emphasized in the introduction, this is both a fundamental theoretic question and a key issue in practice. Beside purely dynamic concerns, our result is of great interest to solve other algorithmic problems, using incremental approaches. For example, [22] used our dynamic algorithm for the maintain of the PQ-representation of an interval graph in order to design the fastest known algorithm for minimal interval completion of an arbitrary graph, in $O(n^2)$ time complexity.

This asks for the question whether it is possible to follow the same approach to obtain similar results for other classes of graphs, and in particular for chordal graphs which generalise the class of interval graphs (see e.g. [3]). Indeed, the problem of finding a minimal completion of an arbitrary graph into a chordal graph has received a lot of attention (see [45] for an excellent survey on the subject), mainly because of its connections with the computation of treewidth and the resolution of sparse linear systems. The best known complexity for the problem is $O(nm)$ or $O(n^\alpha \log n)$, where $O(n^\alpha)$ is the complexity of matrix multiplication. Then, it is natural to ask whether it is possible to obtain an $O(n^2)$ complexity following the approach of [22]. The key to do so is to be able to dynamically maintain a representation of all the intersection models of a graph in the class in $O(n)$ time, as for the PQ-representation in the case of interval graphs. It turns out that, while interval graphs are the intersection graphs of subpaths of a path, chordal graphs are the intersection graphs of subtrees of a tree. Moreover, the clique graph (see e.g. [20]) is a compact representation of all minimal intersection models of a chordal graph. Is it possible to maintain the clique graph of a dynamically changing chordal graph in $O(n)$ time under vertex insertion? Answering this question is of great interest for the field of algorithmic graph theory, but may not be easy; and the clique graph representation seems actually quite different from the PQ-representation. As a first step toward this goal, one may rather consider the same question for the intermediate class of path graphs (properly containing interval graphs and properly included in chordal graphs) for which a representation quite similar to the PQ-representation, called PR-trees, has been discovered [46].

Acknowledgements

The author thanks the first reviewer for his/her in-depth reading of the paper and his/her numerous remarks which substantially improved its writing.

References

[1] C. Crespelle, Fully dynamic representations of interval graphs, in: Graph-Theoretic Concepts in Computer Science: 35th International Workshop (WG'09), no. 5911 in LNCS, 2009, pp. 77–87.

[2] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the third annual ACM symposium on Theory of computing, STOC'71, ACM, 1971, pp. 151–158.

[3] A. Brandstädt, V. Le, J. Spinrad, Graph Classes: a Survey, SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, 1999.

[4] G. Hajös, Über eine art von graphen, Inter. Math. Nachr. 11.

[5] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, 2nd Edition, Vol. 57 of Annals of Discrete Mathematics, Elsevier, 2004.

[6] K. S. Booth, G. S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, J. Comput. Syst. Sci. 13 (3) (1976) 335–379.

[7] D. G. Corneil, S. Olariu, L. Stewart, The ultimate interval graph recognition algorithm?, in: ninth annual ACM-SIAM symposium on Discrete algorithms, 1998, pp. 175 – 180.

[8] M. Habib, R. McConnell, C. Paul, L. Viennot, Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, Theoretical Computer Science 234 (1-2) (2000) 59–84.

[9] W.-L. Hsu, A simple test for interval graphs, in: 18th International Workshop on Graph-Theoretic Concept in Computer Science (WG92), no. 657 in Lecture Notes in Computer Science, 1993, pp. 11–16.

[10] W.-L. Hsu, T.-H. Ma, Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs, SIAM Journal on Computing 28 (3) (1999) 1004–1020.

[11] N. Korte, R. H. Möhring, An incremental linear-time algorithm for recognizing interval graphs, SIAM J. Comput. 18 (1989) 68–81.

[12] D. Kratsch, R. McConnell, K. Mehlhorn, J. Spinrad, Certifying algorithms for recognizing interval graphs and permutation graphs, SIAM Journal on Computing 36 (2) (2006) 326–353.

[13] R. H. Möhring, Algorithmic aspects of comparability graphs and interval graphs, in: I. Rival (Ed.), Graphs and Orders, D. Reidel, Boston, 1985, pp. 41–101.

[14] K. Simon, A new simple linear algorithm to recognize interval graphs, in: International Workshop on Computational Geometry - Methods, Algorithms and Applications, Vol. 553 of Lecture Notes In Computer Science, 1991, pp. 289 – 308.

[15] D. Corneil, A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs, Discrete Applied Mathematics 138 (3) (2004) 371–379.

[16] D. G. Corneil, H. Kim, S. Natarajan, S. Olariu, A. P. Sprague, Simple linear time recognition of unit interval graphs, Information Processing Letters 55 (2) (1995) 99–104.

[17] C. M. H. de Figueiredo, J. Meidanis, C. P. de Mello, A linear-time algorithm for proper interval graph recognition, Information Processing Letters 56 (3) (1995) 179–184.

[18] P. Hell, J. Huang, Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs, SIAM J. Discrete Math. 18 (3) (2004) 554–570.

[19] B. S. Panda, S. K. Das, A linear time recognition algorithm for proper interval graphs, Information Processing Letters 87 (3) (2003) 153–161.

[20] J. P. Spinrad, Efficient graph representations, Vol. 19 of Fields Institute Monographs, American Mathematical Society, 2003.

[21] N. Korte, R. Möhring, Transitive orientation of graphs with side constraints, in: WG, 1985, pp. 143–160.

[22] C. Crespelle, I. Todinca, An $O(n^2)$-time algorithm for the minimal interval completion problem, Theor. Comput. Sci. 494 (2013) 75–85.

[23] G. D. Battista, R. Tamassia, On-line planarity testing, SIAM J. Comput. 25 (5) (1996) 956–997.

[24] C. Crespelle, C. Paul, Fully dynamic algorithm for recognition and modular decomposition of permutation graphs, Algorithmica 58 (2) (2010) 405–432.

[25] C. Crespelle, C. Paul, Fully dynamic recognition algorithm and certificate for directed cographs, Discrete Applied Mathematics 154 (12) (2006) 1722–1741.

[26] D. Eppstein, Z. Galil, G. F. Italiano, T. H. Spencer, Separator based sparsification II: edge and vertex connectivity, SIAM J. Comput. 28 (1) (1998) 341–381.

[27] P. Hell, R. Shamir, R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs, SIAM Journal on Computing 31 (1) (2002) 289–305.

[28] L. Ibarra, Fully dynamic algorithms for chordal graphs and split graphs, ACM Trans. Algorithms 4 (4) (2008) 40:1–40:20.

[29] S. D. Nikolopoulos, L. Palios, C. Papadopoulos, A fully dynamic algorithm for the recognition of P_4-sparse graphs, Theoretical Computer Science 439 (2012) 41 – 57.

[30] R. Shamir, R. Sharan, A fully dynamic algorithm for modular decomposition and recognition of cographs, Discrete Applied Mathematics 136 (2-3) (2004) 329–340.

[31] M. Tedder, D. G. Corneil, An optimal, edges-only fully dynamic algorithm for distance-hereditary graphs, in: STACS, Vol. 4393 of LNCS, 2007, pp. 344–355.

[32] W.-L. Hsu, On-line recognition of interval graphs in $O(m+n\log n)$ time, in: Combinatorics and Computer Science, 1996, pp. 27–38.

[33] W. Springer, Dynamic representation of consecutive-ones matrices and interval graphs, Ph.d. thesis, Colorado State University (2015).

[34] L. Ibarra, A fully dynamic graph algorithm for recognizing interval graphs, Algorithmica 58 (3) (2010) 637–678.

[35] L. Ibarra, A fully dynamic graph algorithm for recognizing proper interval graphs, in: International Workshop on Algorithms and Computation (WALCOM 2009), Springer, 2009, pp. 190–201.

[36] J. Muller, J. Spinrad, Incremental modular decomposition algorithm, Journal of the Association for Computing Machinery 36 (1) (1989) 1–19.

[37] R. M. McConnell, F. de Montgolfier, Algebraic operations on PQ trees and modular decomposition trees, in: 31th Int. Workshop on Graph Theoretical Concepts in Computer Science (WG05), no. 3787 in Lecture Notes in Computer Science, 2005, pp. 421–432.

[38] A. Bergeron, C. Chauve, F. de Montgolfier, M. Raffinot, Computing common intervals of k permutations, with applications to modular decomposition of graphs, SIAM Journal on Discrete Mathematics 22 (3) (2008) 1022–1039.

[39] B. Bui-Xuan, M. Habib, C. Paul, Revisiting Uno and Yagiura's algorithm, in: 16th International Symposium on Algorithms and Computation (ISAAC05), no. 3827 in Lecture Notes in Computer Science, 2005, pp. 146–155.

[40] T. Uno, M. Yagiura, Fast algorithms to enumerate all common intervals of two permutations, Algorithmica 26 (2) (2000) 290–309.

[41] P. C. Gilmore, A. J. Hoffman, A characterization of comparability graphs and of interval graphs, Canad. J. Math. 16 (1964) 539–548.

[42] R. Möhring, Algorithmic aspect of the substitution decomposition in optimization over relations, set systems and boolean functions, Annals of Operations Research 4 (1985) 195–225.

[43] R. Möhring, F. Radermacher, Substitution decomposition for discrete structures and connections with combinatorial optimization, Annals of Discrete Mathematics 19 (1984) 257–356.

[44] W.-L. Hsu, R. M. McConnell, PC trees and circular-ones arrangements, Theoretical Computer Science 296 (1) (2003) 99–116.

[45] P. Heggernes, Minimal triangulations of graphs: A survey, Discrete Math. 306 (3) (2006) 297–317.

[46] S. Chaplick, Path graphs and PR-trees, Ph.d. thesis, University of Toronto (2012).