Fully Dynamic Representations of Interval Graphs

Christophe Crespelle

LIP6, CNRS - Université Paris 6
christophe.crespelle@lip6.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.

1 Introduction

In this paper, we are interested in the *dynamic recognition and representation problem* for the class of interval graphs. For a family $\mathcal{F}$ of graphs, this problem is to maintain a characteristic representation of dynamically changing graphs as long as the modified graph belongs to $\mathcal{F}$ [3, 4, 6, 8, 15, 16]. 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.

Related works. The seminal paper for the recognition of interval graphs [1] solved the problem in linear time by introducing a data structure called PQ-tree. The algorithm of [1] is not dynamic: even though the consecutiveness constraints of each vertex are added one by one, the maximal cliques of the graph need to be computed in advance. 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 and must be precomputed statically. On the opposite, the algorithm of [7] is truly incremental on vertices. In the worst case, the cost of a vertex insertion may be up to $\Theta(n)$. But unfortunately, as mentioned by the author, the data structure he uses does not allow

to treat vertex deletion, while our algorithm is able to do so, within the same worst case time complexity. For edge modifications, [9] designed a fully dynamic algorithm that runs in $O(n \log n)$ time per operation. Here, we lower this complexity to $O(n)$.

Our results. Our algorithm treats the insertion of a vertex in an interval graph in a truly dynamic manner and is the first one also treating the deletion of a vertex; both operations being handled in $O(n)$ time. We also lower the complexity of the best dynamic algorithm for edges [9] 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, within the same complexity, three useful representations of the graph: a minimal interval model, the PQ-tree and the modular decomposition.

Beside our algorithmic results, we give new insight into the structure of interval graphs by showing strong connections between the PQ-tree and the modular decomposition of an interval graph. It should also be noted that Theorem 3 gives a characterisation of the neighbourhood of a vertex in an interval graph. Complete proofs of all results presented here can be found in [2].

2 Preliminaries

Every graph considered here will be finite, undirected, loopless and simple. Throughout the paper, V denotes the vertex set of graph G and E its edge set; we write $G = (V, E)$. n stands for $|V|$ and an edge between x and y is denoted indifferently xy or yx. The neighbourhood of a vertex $x \in V$ is denoted $N(x)$ and its non-neighbourhood $\overline{N}(x)$. $\mathcal{K}(G)$ is the set of maximal cliques of G. A vertex x is simplicial in G iff its neighbourhood is a clique. A subset $S \subsetneq V$ of vertices is *uniform* wrt. 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. When there is no confusion, we omit to mention the vertex x referred to. For a rooted tree T and a node u of T, we denote $parent(u)$ for the parent of u in T, $\mathcal{C}(u)$ for its set of children, $Anc(u)$ for the ancestors of u in T ($u \in Anc(u)$), and T_u for the subtree of T rooted at u. We sometimes identify the tree and its set of nodes by denoting $u \in T$. For a linear ordering σ, we denote $min(\sigma)$ and $max(\sigma)$ for respectively the first and last element of σ.

Interval graphs. An interval model of a graph G 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. Interval graphs are the graphs that admit such a model. In all the models considered in the following, intervals will be closed and will have integer bounds (the class remains the same under this restriction). Associating with each vertex the two integer bounds of its corresponding interval yields an efficient data structure providing adjacency testing in constant time. Interval graphs are well known to be *chordal*, that is, they do not contain any induced cycle of length ≥ 4. One of their nicest characterisations is the following.

Theorem 1. [5] *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 (or $\mathcal{K}(G)$). Numbering the maximal cliques with their rank in a consecutive ordering σ and assigning to each vertex x of G the interval of σ consisting of the cliques containing x results in a model of G. The minimal models are precisely those that can be obtained this way.

It is shown in [1] that all the consecutive orderings of the maximal cliques of a graph G can be represented by an $O(|\mathcal{K}(G)|)$-space structure called PQ-tree. The PQ-tree of G, denoted T^c, is a rooted tree whose leaves are the maximal cliques of G. Its internal nodes are labeled P (*degenerate nodes*) or Q (*prime nodes*). Any Q-node q 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 an assignation, to each node u of T, of a linear ordering on its children: any linear ordering if u is a P-node, σ_u or $\bar{\sigma}_u$ if u 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 $u \in T$ are explored in the order defined by s. A result of [1] states that the frontier is a one to one mapping from the set of solidifications of T^c onto the set of consecutive orderings of G.

Modular decomposition. The reader which is not familiar with the basic notions of modular decomposition theory such as *module*, *strong module*, *maximal strong module* (whose set is denoted $\mathcal{MSM}(G)$) and *prime graph* may refer to [13].

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. Similarly, for a family $\mathcal{P}$ of pairwise disjoint modules, we define the quotient graph $G/\mathcal{P}$ by choosing a representative vertex for each module in $\mathcal{P}$.

The modular decomposition tree of G is denoted T^m, its leaves are the vertices of G and a node $p \in T^m$ represents the strong module of G, denoted $V(p)$, 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 $G[V(p)]$. To each node p of T^m, we associate its quotient graph $G_p = G[V(p)]/\mathcal{MSM}(G[V(p)])$. From the well-known modular decomposition theorem, the quotient G_p is either a stable set, then p is labeled *parallel*, or a clique, then p is labeled *series*, or a prime graph, then p is labeled *prime*. The parallel and series nodes are also called degenerate nodes. We will need the following lemma.

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.*

3 Three representations of interval graphs

Minimal interval models of an interval graph G consist of a consecutive ordering σ of G stored as a list. Each cell contains its position in the list and each vertex of G is assigned two pointers (possibly the same) toward the cells representing the first and the last (wrt. σ) maximal clique of G containing x. The size of such a structure is $O(n + |\mathcal{K}(G)|) = O(n)$ as $|\mathcal{K}(G)| \leq n - 1$ for any interval graph.

The *PQ*-representation is essentially the same structure as the MPQ-tree introduced in [10]. In the classic PQ-tree, the maximal clique corresponding to a leave of T^c is stored by the list of its vertices, which results in an $O(n+m)$ space structure, while the number of nodes in the PQ-tree is only $O(n)$. In the PQ-representation, the vertices of G are stored in the internal nodes of T^c (thanks to the pointers defined below) instead of being stored in its leaves. Let u be a node of T^c, we denote $\mathcal{K}_{T^c}(u)$ for the maximal cliques of G corresponding to the leaves of T^c_u. For a subset $S \subseteq V$ of vertices, we denote $\mathcal{K}(S)$ for the set of maximal cliques of G containing S; and for a singleton we denote $\mathcal{K}(x)$ instead of $\mathcal{K}(\{x\})$. For a vertex $x \in V$, we denote e_x for the least common ancestor of the leaves of T^c corresponding to the maximal cliques of G containing x.

Lemma 2. *[11] For any vertex x of an interval graph G, exactly one of the two following conditions holds: **(i)** $\mathcal{K}(x) = \mathcal{K}_{T^c}(e_x)$, or **(ii)** e_x is a prime node and*

$\exists (u_1, u_2) \in (\mathcal{C}(e_x))^2 \setminus \{(min(\sigma_{e_x}), max(\sigma_{e_x}))\}, u_1 <_{\sigma_{e_x}} u_2$ *and* $\mathcal{K}(x) = \bigcup_{u_1 \leq v \leq u_2} \mathcal{K}_{T^c}(v)$

When (ii) is satisfied, we denote e_x^1 and e_x^2 for the children u_1 and u_2 of e_x. The PQ-representation of an interval graph G, denoted $PQ(G)$, is made of T^c and the set of vertices of G, where each vertex x stores a *primary pointer* toward e_x, and two *secondary pointers* toward resp. e_x^1 and e_x^2 when x satisfies (ii). These pointers encode which maximal cliques of G (i.e. the leaves of T^c) contain x. Since the number of nodes in T^c is $O(n)$ and since each vertex of G stores at most three pointers, it follows that the total size of the PQ-representation is $O(n)$.

Notation 1 *(cf. Fig 1) Let ρ be the root of T^c. For each node u of T^c, we define the following sets:*

$X_u = \{y \in V \mid e_y = u \text{ and } y \text{ has no secondary pointers}\}$

$Y_u = \{y \in V \mid e_y = u \text{ and } y \text{ has secondary pointers toward the children of } u\}$

$u^* = \{y \in V \mid e_y \in T_u^c\}$

$\Delta_u = \begin{cases} \{y \in Y_{\hat{u}} \mid e_y^1 \leq_{\sigma_{\hat{u}}} u \leq_{\sigma_{\hat{u}}} e_y^2\} & \text{if } u \neq \rho \quad (\text{where } \hat{u} = parent(u)) \\ \varnothing & \text{if } u = \rho \end{cases}$

$B_u = \bigcup_{v \in Anc(u)} X_v \cup \Delta_v$

Note that, by definition, if u is degenerate then $Y_u = \varnothing$, and if $parent(u)$ is degenerate then $\Delta_u = \varnothing$. B_u is the set of vertices that belong to all the maximal cliques corresponding to the leaves of T_u^c, and u^* is the set of vertices that are involved only in those cliques. The maximal clique of G corresponding to a leaf $f \in T^c$ is precisely B_f.

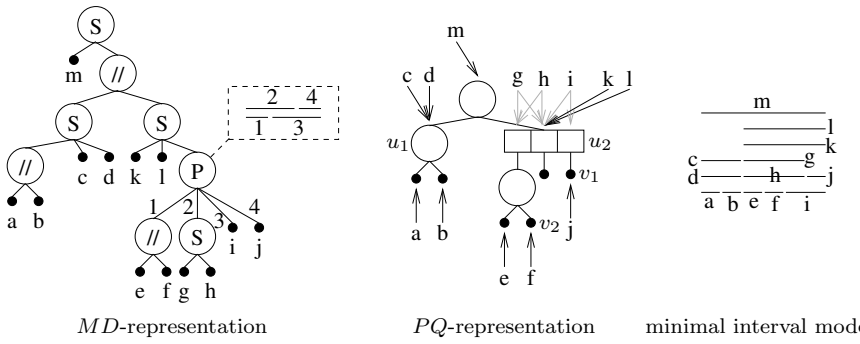

Fig. 1. Three representations of an interval graph. In the PQ-representation, the degenerate nodes are represented by circles and the prime nodes by rectangles. The primary pointers are black, while the secondary pointers are grey. The primary pointers of the vertices that have secondary pointers are not represented. We have $u_1^* = \{a, b, c, d\}$, $X_{u_1} = \{c, d\}$, $X_{u_2} = \{k, l\}$, $Y_{u_2} = \{g, h, i\}$, $B_{v_2} = \{f, g, h, k, l, m\}$ and $\Delta_{v_1} = \{i\}$.

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 and a mapping from the vertices of G_p onto $\mathcal{C}(p)$. In the case where G is an interval graph, the quotient graphs are stored as minimal interval models. As the model of a prime node p takes $O(|\mathcal{C}(p)|)$ space, it yields an $O(n)$ space representation of G.

3.1 Linear-time equivalence of PQ-representation and MD-representation

The equivalence is based on the fact that the PQ-representation of an interval graph is quite well structured regarding its strong modules.

Theorem 2. *Let G be an interval graph. M is a non-trivial strong module of G iff $|M| > 1$ and there exists some node $u \in T^c$ satisfying one of the two following conditions:*

1. *$M = u^*$ or $M = u^* \setminus X_u$; or*
2. *u is prime and $\exists u_1, u_2 \in \mathcal{C}(u), M = \{y \in Y_u \mid e_y^1 = u_1 \text{ and } e_y^2 = u_2\}$.*

Theorem 2 justifies that the MD-representation can be obtained from the PQ-representation as follows. Hereinafter, we refer to this process as the PQ-MD-transformation. In a bottom-up manner, for each node u of T^c, we compute $T^m(G[u^*])$. For an internal node u, whose set of children is $\{u_1, \ldots, u_k\}$, we have to consider three different cases:

1. If $X_u = \varnothing$ and u is a degenerate node, the root $\tilde{u}$ of $T^m(G[u^*])$ is a parallel node whose children are the roots of the trees $T^m(G[u_1^*]), \ldots, T^m(G[u_k^*])$.
2. If $X_u = \varnothing$ and u is a prime node, the root $\tilde{u}$ of $T^m(G[u^*])$ 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 $T^m(G[v^*])$ 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 $y \in \bar{y}$.
3. If $X_u \neq \varnothing$, we first ignore the vertices of X_u and build $T^m(G[u^* \setminus X_u])$ like described above. Then, we introduce a new series node whose children are the leaves representing vertices of X_u and the root of $T^m(G[u^* \setminus X_u])$.

Processing the leaves of T^c takes $O(n)$ time. For a degenerate node u, treatment 1 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 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 $O(|Y_u|+|\mathcal{C}(u)|)$. Finally, treatment 3 takes $O(|X_u|)$ time. Thus, the total computation time of $T^m(G)$ is $O(\sum_{u \in T^c} |X_u| + \sum_{u \in T^c} |Y_u| + \sum_{u \in T^c} |\mathcal{C}(u)|) = O(n)$.

For lack of space, we do not detail the converse operation that gives the PQ-representation from the MD-representation. It leans on a bottom-up search of T^m, similar to the one of T^c, which also runs in $O(n)$ time.

4 The dynamic algorithm

Since our three representations are $O(n)$ time equivalent, and since we want to get an $O(n)$ time algorithm, we can focus on maintaining only one of them and get the others within the same complexity. We chose to concentrate on showing how to maintain the MD-representation. However, when they are more convenient, we will also use the other representations and the equivalence between them.

Since edge modifications can be handled by one vertex deletion followed by one vertex insertion, we will not specifically consider them.

For lack of space, we do not present the deletion algorithm but only give its general idea. When the parent $u \in T^m$ of the vertex x to be deleted is degenerate, we simply remove the leaf corresponding to x and make some local cleaning of the tree, exactly as in [3]. When u is prime, we use the algorithm of [12] that computes the PQ-tree from an interval model, and we use the PQ-MD-transformation to get the updated MD-representation. We now concentrate on vertex insertion.

4.1 Focus on the key node

[3] showed that, for an arbitrary graph, the modifications of T^m under vertex insertion are located in the subtree of T^m rooted at the *insertion node* w_m, defined further. Here, we introduce the *key node* w that plays the same role in T^c and we show that, in order to determine whether $G + x$ is an interval graph, we can restrict our attention to $G[w^*] + x$.

From now on, x will be a vertex to be inserted in an interval graph G, and we denote G' for $G + x$. We will say that a node $u \in T^m$ (resp. T^c)

is uniform, mixed, full, hollow or linked (see definitions p. 2) referring to the set $V(u)$ (resp. u^*). A node $u \in T^c$ is *saturated* iff u^* and B_u are full.

Definition 1. *[3] A node $u \in T^m$ is* proper *iff either u is uniform wrt. x, or u is a mixed node with a (unique) mixed child f such that $V(f) \cup \{x\}$ is a module of $G'[V(u) \cup \{x\}]$. The* insertion node*, denoted w_m, is the least common ancestor of the non-proper nodes of T^m.*
Node w_m is said to be cut *iff w_m has no mixed child and either w_m is prime and has a child f such that $V(f) \cup \{x\}$ is a module of $G'[V(w_m) \cup \{x\}]$, or w_m is degenerate.*

In the following, we do not consider the trivial case where V is uniform. Moreover, from now on, we only consider the case where w_m is not cut and the neighbourhood of $V(w_m)$ is a clique: the other cases are easy to deal with. We adopt the following definition.

Definition 2. *The* key node *w is the node of T^c such that $V(w_m) = w^*$ or $V(w_m) = w^* \setminus X_w$.*

Note that in Condition 2 of Theorem 2, the neighbourhood of M is not a clique. Thus, in the present case, $V(w_m)$ satisfies Condition 1 of Theorem 2, which ensures the existence of node w. It is straightforward to see that $V(w_m) \cup \{x\}$ is a module of $G + x$, and since w^* is a module of G and $V(w_m) \subseteq w^*$, 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 a clique. Then, the lemma below follows from Lemma 1.

Lemma 3. *$G + x$ is an interval graph iff $G[w^*] + x$ is an interval graph.*

4.2 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. We start with the definitions and notations we use in our characterisation.

Definition 3. *Let u be a prime node of T^c and $v \in \mathcal{C}(u)$. v* satisfies the left (resp. right) property *iff $\forall y \in Y_u \cap N(x), e_y^2 \geq v$ (resp. $e_y^1 \leq v$).*

Notation 2 *If the saturated children of u form an interval of σ_u, we denote I_u for this interval and l_u (resp. r_u), if it exists, for the child of u immediately preceding (resp. following) I_u.*

Lemmas 4 to 7 give some necessary conditions for $G + x$ to be an interval graph, and Theorem 3 states that they are also sufficient. We omitted the proofs of the Lemmas since they are too technical to be sketched within the space limitation. As a general hint, we can say that their statement widely lean on the fact that deleting x in all the cliques of a consecutive ordering σ' of G' and removing the obtained cliques that are not maximal in G results in a consecutive ordering of G. Roughly speaking, this implies that, in σ', the maximal cliques of the new graph G' appear in an order that somehow respects the constraints previously imposed by the nodes of T^c.

Lemma 4. *If $G + x$ is an interval graph, any node $u \in T^c_w \setminus \{w\}$ has at most one mixed child, and w has at most two mixed children. Furthermore, for all $u \in T^c_w$, if B_u is not full, then u has at most one linked child.*

The other necessary conditions for G' to be an interval graph apply only to prime nodes of T^c_w.

Lemma 5. *If $G+x$ is an interval graph, for all prime nodes $u \in T^c_w$, the set of saturated children of u is an interval I_u of σ_u. And if $I_u \neq \varnothing$, then any node $v_1 \in \mathcal{C}(u) \setminus (I_u \cup \{l_u, r_u\})$ is hollow.*

Lemma 6. *If $G + x$ is an interval graph, any prime node $u \neq w$ of T^c_w satisfies one of the following conditions:*

1. *B_u is full and $I_u \neq \varnothing$; and, up to reversing σ_u, $max(\sigma_u) \in I_u$ and l_u satisfies the left property.*
2. *B_u is full and $I_u = \varnothing$, 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.*

Lemma 7. *If $G + x$ is an interval graph and if w is a prime node, it satisfies one of the following conditions:*

1. *B_w is full and $I_w \neq \varnothing$; and l_w and r_w satisfy respectively the left and right property.*
2. *B_w is full and $I_w = \varnothing$; 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 respectively the left and right property, and the nodes of $\mathcal{C}(w) \setminus \{l, r\}$ are hollow, and $\Delta_l \cap \Delta_r \subseteq N(x)$; 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.*

Theorem 3. *$G{+}x$ is an interval graph iff the conditions of Lemmas 4 to 7 are satisfied.*

Sketch of proof. If the conditions of Lemmas 4 to 7 are satisfied, we can build a consecutive ordering of $G[w^*] + x$. To that purpose, we first build, for every full node u, a consecutive ordering of $G[u^*] + x$. Then, inductively, in a bottom up traversal of T^c_w, we build, for every mixed node $u \in T^c_w \setminus \{w\}$, a consecutive ordering of $G[u^*] + x$ st. the last maximal clique contains x. There are several cases to be considered. We cannot discuss each of them but we detail, as an example, the case where u satisfies Cond. 1 of Lemma 6 and l_u is mixed and B_{l_u} is full. In this case we obtain a consecutive ordering σ' of $G[u^*] + x$ by appending, in the order defined by σ_u, the consecutive orderings of $G[v^*]$ of nodes $v <_{\sigma_u} l_u$, the consecutive ordering of $G[l^*_u] + x$ built previously in the induction, and the consecutive orderings of $G[v^*] + x$ of nodes $v >_{\sigma_u} l_u$. Moreover, for any $v \in \mathcal{C}(u)$, we add the vertices of $\Delta_v \cup X_u$ to the cliques of $G[v^*]$ (or $G[v^*] + x$ if $v \geq_{\sigma_u} l_u$). Since we use a consecutive ordering of $G[l^*_u] + x$ whose last clique contains x, the cliques of σ' containing x form an interval. Once we get the consecutive orderings related to the mixed children of w, as w satisfies the conditions of Lemma 7, a last induction step allow us to obtain, in a similar way, a consecutive ordering of $G[w^*] + x$. □

4.3 Overview of the algorithm and complexity

The first step of our algorithm collects some information about T^m and T^c, and finds the key node w. The second step checks whether T^c_w satisfies the conditions of Lemmas 4 to 7, that is whether $G{+}x$ is an interval graph. In the positive, the third step updates $MD(G)$ by building $MD(G')$.

Marking step. We first determine for each node of T^m whether it is full, mixed or hollow by a well-known bottom-up marking process of the tree (see [14]), in $O(n)$ time. Then, we find the insertion node w_m by following a path from the root to w, while the visited node u is proper, we visit its unique mixed child (see [3]); the first non-proper node found is w_m. As we mentionned, the cases where w_m is cut or where the neighbourhood of $V(w_m)$ is not a clique are easy to deal with. We now describe the algorithm in the opposite case. Thanks to the correspondence between T^c and T^m, we find the key node w and determine for each node of T^c whether it is full, mixed or hollow. Finally, a simple top-down search of

T^c allows us to determine for each node u whether B_u is full, mixed or hollow. The first step runs in $O(n)$ time.

Testing step. The conditions of Lemma 4 can be tested in $O(|\mathcal{C}(u)|)$ time by a simple search of $\mathcal{C}(u)$. The difficulty of checking the conditions of Lemma 5 is to decide whether the saturated children of u form an interval. To that purpose, we determine the set $S = \{v \in \mathcal{C}(u) \mid \Delta_v \subseteq N(x)\}$. We first bucket sort the vertices of $y \in Y_u \cap \overline{N}(x)$ by increasing e_y^1. As $1 \leq e_y^1 \leq |\mathcal{C}(u)|$, it takes $O(|\mathcal{C}(u)|)$ time. Examining the vertices of $Y_u \cap \overline{N}(x)$ in this order, we are able to maintain a partition of the children v of u such that Δ_v contains none of the vertices $y \in Y_u \cap \overline{N}(x)$ examined so far; each set of this partition being an interval of $\mathcal{C}(u)$. At the end of the routine, we obtain a partition of S. Then, checking the conditions of Lemma 5 becomes easy. It can be done in $O(|\mathcal{C}(u)|+|Y_u|)$ time. For a child v of a prime node u, it is easy to check whether v satisfies the left or right property by scanning Y_u. It follows that the only difficulty in checking the conditions of Lemmas 6 and 7 is to check Cond. 2a of Lemma 7. Let $w_1 = \min_{\sigma_w}\{e_y^2 \mid y \in Y_w \cap N(x)\}$. The children of w satisfying the left property are exactly the nodes $v \leq_{\sigma_w} w_1$. In the same way, we find the children of w satisfying the right property. Then, the couples (f, l) st. f and l resp. satisfy the left and right property define an interval of σ_w. The same technique as the one used to check the conditions of Lemma 5 determines the couples (f, l) such that $\Delta_f \cap \Delta_l \subseteq N(x)$. Hence, Cond. 2a of Lemma 7 can be tested in $O(|\mathcal{C}(w)| + |Y_w|)$ time. Finally, since all the conditions can be tested for a node u in $O(|\mathcal{C}(u)| + |Y_u|)$ time, we can determine whether $G + x$ is an interval graph in $O(n)$ time.

Insertion step. If $G + x$ is not an interval graph, then the algorithm stops. Otherwise, the MD-representation is updated. Since $V(w_m) \cup \{x\}$ is a strong module of $G' = G + x$ (cf. [3]), we can obtain $MD(G')$ by replacing node w_m of $MD(G)$ with the root of $MD(G[V(w_m)] + x)$. In order to get $MD(G[V(w_m)] + x)$ we first compute an interval model of $G[w^*]+x$. In the proof of Theorem 3, it is shown how to build a consecutive ordering of $G[w^*] + x$ by a bottom-up traversal of T_w^c. At each step, we concatenate the orderings computed for the children of the current node u, and we assign pointers to the vertices of $X_u \cup Y_u$; this takes $O(|\mathcal{C}(u)| + |X_u|+|Y_u|)$ time. Thus, the whole processing of T_w^c takes $O(n)$ time. Once we get an interval model of $G[w^*] + x$ we can easily extract a model of $G[V(w_m)]+x$, and thanks to the algorithm of [12] that computes the PQ-tree from an interval model, we get $PQ(G[V(w_m)]+x)$ in $O(n)$ time. The

PQ-MD-transformation (see p.6) provides us with $MD(G[V(w_m)] + x)$ within the same complexity. Thus, the total computation time of $MD(G')$ is $O(n)$.

Acknoledgements

The author thanks Christophe Paul for useful discussions on the subject.

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
2. C. Crespelle. Dynamic representations of interval graphs. Manuscript, 2009. Available at http://www-npa.lip6.fr/∼crespell/publications/DynInt_long.pdf.
3. C. Crespelle and C. Paul. Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. *Algorithmica*. Available at http://www.springerlink.com/content/u6x0054g3h348810/. Ext. abs. in WG'05.
4. C. Crespelle and C. Paul. Fully dynamic recognition algorithm and certificate for directed cographs. *Discrete Applied Mathematics*, 154(12):1722–1741, 2006. Ext. abs. in WG'04.
5. P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canad. J. Math.*, 16:539–548, 1964.
6. P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM J. Comput.*, 31(1):289–305, 2002.
7. W.-L. Hsu. On-line recognition of interval graphs in $O(m + nlogn)$ time. In *Combinatorics and Computer Science*, pages 27–38, 1996.
8. L. Ibarra. Fully dynamic algorithms for chordal graphs. In *SODA*, pages 923–924, 1999.
9. L. Ibarra. A fully dynamic algorithm for recognizing interval graphs using the clique-separator graph. Tech. Report DCS-263-IR, Dept. of Computer Science, University of Victoria, 2001.
10. N. Korte and R. H. Möhring. Transitive orientation of graphs with side constraints. In *WG*, pages 143–160, 1985.
11. N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18:68–81, 1989.
12. R. M. McConnell and F. de Montgolfier. Algebraic operations on PQ trees and modular decomposition trees. In *WG*, number 3787 in LNCS, pages 421–432, 2005.
13. R. H. Möhring and F.J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19:257–356, 1984.
14. J.H. Muller and J.P. Spinrad. Incremental modular decomposition algorithm. *JACM*, 36(1):1–19, 1989.
15. S. D. Nikolopoulos, L. Palios, and C. Papadopoulos. A fully dynamic algorithm for the recognition of p_4-sparse graphs. In *WG*, number 4271 in LNCS, pages 256–268, 2006.
16. M. Tedder and D. G. Corneil. An optimal, edges-only fully dynamic algorithm for distance-hereditary graphs. In *STACS*, number 4393 in LNCS, pages 344–355, 2007.