An $\mathcal{O}(n^2)$-time algorithm for the minimal interval completion problem*

Christophe Crespelle
Université Claude Bernard Lyon 1, D-NET/INRIA
LIP, CNRS, ENS de Lyon
Université de Lyon
christophe.crespelle@inria.fr

Ioan Todinca
LIFO, Université d'Orleans
BP 6759, F-45067 Orleans Cedex 2
ioan.todinca@univ-orleans.fr

Abstract

An interval completion of an arbitrary graph G is an interval graph H, on the same vertex set, obtained from G by adding new edges. If the set of newly added edges is inclusion-minimal among all possibilities, we say that H is a *minimal interval completion* of G. We give an $\mathcal{O}(n^2)$-time algorithm to obtain a minimal interval completion of an arbitrary graph. This improves the previous $O(nm)$ time bound for the problem and lowers this bound for the first time below the best known bound for minimal chordal completion.

1 Introduction

The *interval completion* of a graph $G = (V, E)$ consists in adding a set of edges F to G so that the resulting graph $H = (V, E \cup F)$ is an *interval graph*, that is, the intersection graph of some intervals of the real line. The problem of computing such a completion that realizes the minimum number of *fill edges* $|F|$ is known as the *Minimum Interval Completion* problem. If the set F is only required to be minimal for inclusion among all sets resulting in an interval completion, the problem is referred to as the *Minimal Interval Completion* problem. Applications of Minimum Interval Completion arise in various contexts such as computational biology [2, 8], archeology [16], and clone fingerprinting [15]. In addition, interval completion has been studied for its connection with another fundamental problem of computer science known

*A preliminary version of this paper appeared as [5], in the proceedings of TAMC 2010.

as *chordal completion* (see [10] for a survey). A *chordal graph* is a graph that contains no chordless induced cycle on four or more vertices. The *Minimum Chordal Completion* problem (also known as *minimum fill-in* or *minimum triangulation*) received much attention, in particular because it plays a key role in *sparse matrix multiplication* [24]. Note that interval graphs are a subclass of chordal graphs. Another chordal completion problem is the one consisting in minimizing the maximum clique size of the completed graph. Indeed, this parameter is nothing else but *treewidth* (plus one), which is extensively studied, notably because many NP-complete problems become polynomial on graphs of bounded treewidth. Interval completion is similarly related to another famous graph parameter called *pathwidth*.

Both for interval completion and chordal completion, it turns out that minimizing the number of fill edges or minimizing the maximum clique-size of the completed graph are NP-complete problems [6, 26, 1]. In all these cases, optimal solutions can be found among inclusion-minimal completions. Considering that computing a minimal completion is polynomial (in both cases), this significantly increases the importance of the problem and motivated many works. Minimal completion algorithms are often used as heuristics for minimum completion and for computation of pathwidth or treewidth.

Related work The first algorithm solving the Minimal Chordal Completion problem in polynomial time is due to Rose, Tarjan and Lueker [23], with an $\mathcal{O}(nm)$ time complexity. As usual, n denotes the number of vertices and m denotes the number of edges of the input. Several authors gave different approaches with the same running time, but it took almost 30 years to improve the $\mathcal{O}(n^3)$ worst-case complexity. Using the algorithm of Heggernes, Telle and Villanger [14], one can compute a minimal chordal completion in $\mathcal{O}(n^\alpha \log n)$ time, where $\mathcal{O}(n^\alpha)$ is the time required for the multiplication of two $n \times n$ matrices.

The first polynomial-time algorithm for the Minimal Interval Completion problem was given by Ohtsuki et al. [21], running in $\mathcal{O}(nm')$ time; here m' denotes the number of edges of the resulting completed graph. A similar approach has been rediscovered in [13]. Using a completely different technique, Suchan and Todinca gave an $\mathcal{O}(nm)$-time algorithm in [25]. Note that several recent articles consider different types of minimal completion problems, e.g. into split graphs [11], comparability graphs [12] or proper interval graphs [22].

Our results The aim of this paper is to show the following theorem.

Theorem 1 *There is an $\mathcal{O}(n^2)$-time algorithm computing a minimal interval completion of an arbitrary graph.*

Note that our approach is faster than the previous $\mathcal{O}(nm')$ algorithm of [21] and the $\mathcal{O}(nm)$ algorithm of [25]. It is also faster than the best algorithms for the Minimal Chordal Completion problem, and this is the first time that the time bound for interval completion goes below the best known bound for chordal completion. Moreover, unlike the algorithm of [14] which uses matrix multiplication techniques, ours is purely graph-theoretic. Like in [21], our algorithm is incremental in the sense that we add the vertices of G one by one, and each time a new vertex v_{i+1} arrives, the new minimal interval completion is computed from the one obtained at step i by only adding edges incident to v_{i+1}. The second common feature is that we use PQ-trees, which capture all interval representations of the interval completion computed so far. But our procedure for choosing the set F of fill edges is completely different from [21] and simpler; and we rely on the results of [4] for efficiently updating, at each step, the PQ-tree of the new completion.

It is worth to note that our approach could equivalently make use of the *modular decomposition* of graphs (see [20, 19] for a definition and surveys on the subject) instead of the PQ-tree. Indeed, the PQ-tree of interval graphs is a particular case of the modular decomposition of interval graphs (see e.g. [17, 4]). Here, we chose to use the PQ-tree because, being specific to interval graphs, it offers a very convenient charecterization of their structure. Nevertheless, our approach could be entirely rewritten to use the modular decomposition tree instead of the PQ-tree, and it would give the same results. In particular, this connection may be of high interest for dealing with completion problems with other graph classes that have special structure with regard to the modular decomposition.

After giving, in next section, some basic definitions and preliminary results, we present in Section 3 our main combinatorial tool for the minimal interval completion (a particular case is treated in Section 4), while the algorithmic details and data-structures are described in Section 5.

2 Preliminaries

We consider simple and connected input graphs. A graph is denoted by $G = (V, E)$, with $n = |V|$, and $m = |E|$. For a set $U \subseteq V$, $G[U]$ denotes the subgraph of G induced by the vertices in U. Set U of vertices is called a *clique* if $G[U]$ is complete. For a vertex $v \in V$ or a subset $U \subseteq V$, we will informally use $G - v$ and $G - U$ to denote the graphs $G[V \setminus \{v\}]$ and $G[V \setminus U]$, respectively. We also consider the operation of inserting a new vertex x together with the edges defining its neighborhood in a graph G and we denote the resulting graph by $G + x$. A path is a sequence $[v_1, v_2, \ldots, v_p]$ of vertices such that v_i is adjacent to v_{i+1}, for all $1 \leq i < p - 1$. A cycle is a path such that the first and last vertices are adjacent. The *neighborhood*

of a vertex v in G is $N_G(v) = \{u \mid uv \in E\}$. Similarly, for a set $U \subseteq V$, $N_G(U) = \bigcup_{v \in U} N_G(v) \setminus U$. When graph G is clear from the context, we will omit subscript G; in particular, the neighborhood of vertex x in $G + x$ will often be denoted simply $N(x)$.

A graph H is an *interval* graph if continuous intervals of the real line can be assigned to each vertex of H such that two vertices are neighbors if and only if their intervals intersect. A graph $H = (V, E \cup F)$ is called an *interval completion* of an arbitrary graph $G = (V, E)$ if H is an interval graph. If no proper subgraph of H is an interval completion of G, we say that H is a *minimal interval completion* of G. In a broader sense, we say that a graph H is *inclusion-minimal* among a set of graphs on the same vertex set referring to the inclusion relationship on edge sets of graphs. An edge that is added to the input graph G is called a *fill edge*, and the process of adding edges between a fixed vertex x and a set U of vertices is called *filling* U.

Theorem 2 ([7]) *A graph G is interval if and only if there is a path CP_G whose vertex set is the set of all maximal cliques of G, such that the subgraph of CP_G induced by the maximal cliques of G containing vertex v forms a connected subpath, for each vertex v of G. Such a path will be called a* clique path *of G.*

Let the maximal cliques of an interval graph G be labeled $1, 2, ..., k$, according to the order in which they appear in a clique path of G. Then, as a consequence of Theorem 2, an interval representation of G can be obtained by assigning to each vertex v the closed interval that consists of the labels of the maximal cliques containing v. In this way, every clique path of G defines an interval representation of G.

A vertex set $S \subseteq V$ is a *minimal separator* of G if there exist two vertices u and v such that S separates them (i.e. u and v are in different connected components of $G - S$) and S is inclusion-minimal among the sets of vertices separating u and v. The following lemma shows that minimal separators can be easily found on any clique path of the graph, and so on its PQ-tree (defined further).

Lemma 1 (see e.g. [9]) *Let G be an interval graph and let CP_G be any clique path of G. A set of vertices S is a minimal separator of G if and only if S is the intersection of two maximal cliques of G that are neighbors in CP_G. In particular, all minimal separators of G are cliques.*

It is shown in [3] that all clique paths of an interval graph G can be represented by a structure called PQ-tree. The PQ-tree of G, denoted T in the rest of the paper, is a rooted tree whose leaves are the maximal cliques of

G. Its internal nodes are labeled P (*degenerate nodes*) or Q (*prime nodes*)[1]. 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 *valid* linear ordering on its children, that is: any linear ordering if u is a P-node, σ_u or $\bar{\sigma}_u$ if u is a Q-node. Choosing a solidification of the PQ-tree, we obtain an order on the leaves by reading them from left to right. The main property of the PQ-tree of G is that the set of orders obtained this way is precisely the set of clique paths of G (see [3]).

In this document, the subtree of T rooted at node u will be denoted by T_u. The set of children of u will be denoted by $\mathcal{C}(u)$ and its parent by $parent(u)$.

Let us observe that a minimal interval completion can be obtained incrementally. This result is due to [21].

Lemma 2 ([21]) *Let H be a minimal interval completion of an arbitrary graph G. Let G' be a graph obtained from G by adding a new vertex x, with neighborhood $N_{G'}(x)$. There is a minimal interval completion H' of G' such that $H' - x = H$.*

Hence, for computing a minimal interval completion of G, we introduce the vertices of G one by one in the order $x_1, x_2, \ldots, x_n$. Given a minimal interval completion H_i of $G_i = G[\{x_1, \ldots, x_i\}]$, we compute an interval completion H_{i+1} of G_{i+1} by adding to H_i the vertex x_{i+1} together with the edges between x_{i+1} and $N_{G_{i+1}}(x_{i+1})$, plus a well chosen set of additional edges incident to x_{i+1}. Thus, for proving Theorem 1, it is sufficient to solve the following problem in $\mathcal{O}(n)$ time.

The new problem From now, we consider as input an *interval* graph $G = (V, E)$ on n vertices and a new vertex x to be inserted in G, together with a set of edges incident to x. We want to compute a minimal interval completion H of $G + x$, obtained by adding edges incident to x only. Moreover, the PQ-tree of graph G will be part of the input, and we will also compute the PQ-tree of H.

For the rest of this document, let G' denote the graph $G + x$. Consider any clique path CP_H of the obtained completion H of G'. By property of clique paths, the cliques containing x form a subpath P_x of CP_H. Now, let us get back to G. Delete x from every clique in CP_H, and possibly remove the cliques that do not correspond to maximal cliques of G. This yields

[1]The terms degenerate nodes and prime nodes, which we use throughout the paper, come from the context of modular decomposition of graphs. Indeed, in the PQ-tree, the P nodes correspond to some degenerate nodes of the modular decomposition, while the Q nodes correspond to its prime nodes (see e.g. [17, 4] for details).

a clique path CP_G of G, which is said to be obtained by *pruning* vertex x from CP_H. Clearly the maximal cliques that come from P_x still form a subpath of CP_G. Our aim is to do the converse: to find a clique path CP_G of G and a subpath of CP_G in which, by adding vertex x to every clique of the subpath and possibly transforming the bordering separators into new maximal cliques of H (with x contained), we obtain a minimal interval completion of G'.

Definition 1 (completion respecting a clique path) *An interval completion H of G* respects *a clique path CP_G of G iff there exists some clique path CP_H of H such that CP_G is obtained by pruning x from CP_H.*

Definition 2 (nice clique path) *A clique path CP_G is called* nice *if there exists a minimal interval completion H respecting CP_G.*

For obtaining a nice clique path we have to distinguish between two cases. The case when the neighborhood of x in G' is a clique is treated in Section 4. From now on, we rather concentrate on the more general case where the neighborhood of x is not a clique.

3 When the neighborhood of x is not a clique

This is the main and most difficult case of our algorithm. We show later how to handle it using the PQ-tree. But for now, let us first note that, as stated in [13], any clique path of G is associated with a canonical interval completion respecting it (Lemma 3 below). We need the following definition.

Definition 3 *Fix a clique path CP_G of G. In the case where $N(x)$ is not a clique, we denote by $K_L^{CP(G)}$ (resp. $K_R^{CP(G)}$) the leftmost (resp. rightmost) clique of CP_G such that x has a neighbor in $K_L^{CP(G)} \setminus K_{L+1}^{CP(G)}$ (resp. $K_R^{CP(G)} \setminus K_{R-1}^{CP(G)}$), in graph G'.*

As usual, when the clique path $CP(G)$ is clear from the context, we will simply write K_L and K_R instead of $K_L^{CP(G)}$ and $K_R^{CP(G)}$. Here K_{L+1} denotes the clique appearing right after K_L in the clique path.

Lemma 3 (Canonical completion respecting a clique path [13]) *Given any clique path CP_G of G, there is a unique interval completion H respecting CP_G which is inclusion-minimal among such completions.*

Moreover the neighborhood of x in H is formed exactly by $N(x)$ augmented with the vertices of the cliques strictly between K_L and K_R in CP_G, if there are some, or augmented with the vertices of the minimal separator $K_L \cap K_R$ otherwise.

Note that if CP_G is a nice clique path, then the canonical completion H respecting CP_G, which is inclusion minimal among all completions respecting CP_G, is necessarily a minimal completion of G'. But this is not the case when CP_G is not a nice clique path.

Remark 1 *Any clique path CP' obtained from a nice one CP_G by rearranging the maximal cliques of G in an order such that the cliques of the interval $[\![K_L^{CP_G}, K_R^{CP_G}]\!]$ of CP_G still form an interval whose endpoints are $K_L^{CP_G}$ and $K_R^{CP_G}$ (not necessarily in this order) is a nice clique path.*

Proof. Up to reversing CP', we can assume without loss of generality that $K_L^{CP_G}$ is still on the left of $K_R^{CP_G}$ in CP'. Let us show that $K_L^{CP'} = K_L^{CP_G}$. $K_L^{CP_G}$ contains a neighbor y of x which is not included in any clique on the right of $K_L^{CP_G}$ in CP_G. Since the interval $[\![K_L^{CP_G}, K_R^{CP_G}]\!]$ is conserved, the clique immediately following $K_L^{CP_G}$ in CP' does not contain y. Now let z be a neighbor of x contained in a maximal clique K which is left to $K_L^{CP_G}$ in CP'. In CP_G, K does not belong to interval $[\![K_L^{CP_G}, K_R^{CP_G}]\!]$. Then, by definition of $K_L^{CP_G}$ and $K_R^{CP_G}$, z belongs to $K_L^{CP_G}$ or $K_R^{CP_G}$. It follows that, in CP', z is contained in the clique immediately following K. This shows that $K_L^{CP_G} = K_L^{CP'}$ and symmetrically $K_R^{CP_G} = K_R^{CP'}$. Finally, it implies that the canonical completion respecting CP' is the same as the canonical completion respecting CP_G, which is a minimal interval completion. Thus, CP' is a nice clique path. □

Before proving our main theorem (Theorem 3) which gives the tools for obtaining a nice clique path thanks to the PQ-tree, we need to introduce some definitions and notations. For any node u of the PQ-tree of G, we denote by $B[u]$ the set of vertices of G contained in the cliques of the subtree rooted in u. We call this set a *block*. The *border* of $B[u]$ is the set of vertices of the block having neighbors outside the block. The *interior* of $B[u]$ is formed by the vertices of the block that are not in the border. We say that $B[u]$ is *hit* if, in the graph G', x has a neighbor in the interior of the block. Otherwise, the block is called *clean*. If all vertices in the interior of the block are neighbors of x in G' then the block is called *full*. By extension, we also say that a node of the PQ-tree is hit, full or clean according to the state of its corresponding block.

Remark 2 *We point out that, for any* internal *node u of the PQ-tree, the block $B[u]$ has a non-empty interior. More precisely, for any solidification of T, the leftmost clique of T_u contains a vertex in the interior of $B[u]$.*

Proof. Consider any solidification of the PQ-tree, let CP be the corresponding clique path and KL^u be the clique corresponding to the leftmost leaf of the subtree rooted in u. Since KL^u is not included in its successor

in CP, there exists a vertex $y \in KL^u$ which does not appear to the right of KL^u in CP. y cannot appear to the left of KL^u either because, by reversing the order of all cliques in the subtree rooted in u, we also get a clique path, in which the cliques containing y must be consecutive. Thus y is involved in a unique maximal clique and therefore is in the interior of $B[u]$. □

The children $L_\sigma(u)$ and $R_\sigma(u)$ introduced in the following definition correspond to the cliques K_L and K_R of Definition 3, the difference being that here we consider blocks instead of cliques.

Definition 4 *Consider a node u of the PQ-tree and a valid order σ of its children. We denote by $L_\sigma(u)$:*

- *either the leftmost child v of u such that the corresponding block $B[v]$ contains a neighbor of x in G', and this neighbor is not in $B[v']$, where v' is the right-hand brother of v in σ,*
- *or the last (i.e., right-most) element of σ if u has no such child v.*

$R_\sigma(u)$ is defined symmetrically.

Lemma 4 *Let r be a lowest node of the PQ-tree such that $B[r]$ contains $N(x)$. Then (under the assumption that $N(x)$ is not a clique of G) r is uniquely defined and all nodes v satisfying $N(x) \subseteq B[v]$ are ancestors of r.*

Moreover, for any valid order σ on the children of r, $L_\sigma(r)$ appears before $R_\sigma(r)$ in this order.

Proof. For the first claim, suppose by contradiction there exists a node v such that $N(x) \subseteq B[v]$ and v is not an ancestor of r. Note that r cannot be an ancestor of v and let w be the least common ancestor of r and v. Let u and u' be the sons of w, on the branches leading to r and v respectively. Clearly $N(x) \subseteq B[u] \cap B[u']$. On the other hand, $B[u] \cap B[u']$ induces a clique in G, contradicting the fact that $N(x)$ is not a clique of G.

For the second claim, note there are two sons u and w such that $B[u]$ contains a neighbour of x which is not in $B[w]$ and vice-versa (otherwise r would not be the lowest node whose block contains $N(x)$). Assume w.l.o.g. that $u < w$ in the order σ. Therefore, in σ, we have $L_\sigma(r) \leq u < w \leq R_\sigma(r)$, which concludes the statement. □

Using the node r defined above, we can identify two branches of a solidified PQ-tree delimiting the part of the tree containing nodes whose blocks are filled in the canonical completion associated to the considered solidification.

Definition 5 *Let G be an interval graph and x be a vertex to be inserted in G. Let π be a solidification of T. Denote by $\pi(u)$ the ordering defined by*

this solidification on the children of node u. The left branch $LB(\pi)$ *of π is the set of nodes defined recursively as follows :*

- $L_{\pi(r)}(r)$ *is in the left branch.*
- *For any u in the left branch, $L_{\pi(u)}(u)$ is also in the left branch.*

The right branch $RB(\pi)$ of π is defined symmetrically.

The left and the right branch of π isolate a subpart of the solidified PQ-tree. Let CP_G be the clique path corresponding to this solidification π. Let H be the unique interval completion respecting CP_G that is inclusion-minimal (see Lemma 3). Observe that, by the definition of the left and right branch, the bottom of these branches correspond to K_L and K_R respectively (see Definition 3). By Lemma 3, all maximal cliques strictly in between K_L and K_R become filled in H. All cliques strictly outside this interval remain clean. An important consequence is that, for any node of the PQ-tree not belonging to one of the two branches and different from r, we can change the permutation of its children and the new solidification will yield the same interval completion.

We define a class of nodes u, that we call *forced*, which have the property that their corresponding block becomes filled in any interval completion of G (Lemma 5 below).

Definition 6 *A* forced *node is defined inductively (bottom-up) by: a node u of the PQ-tree is forced if and only if:*

- *u is full, or*
- *u is a degenerate node and every child v of u is forced.*
- *u is a prime node and the first and the last child of σ_u are forced.*

Lemma 5 (forced blocks) *Let u be an internal forced node of the PQ-tree of G. The block $B[u]$ is filled in every interval completion of G'.*

Proof. Consider any interval completion of G', it respects some clique path CP_G of G. Let π be the corresponding solidification. Since, from Lemma 3, there is a unique interval completion H respecting CP_G which is minimal for inclusion, it is enough to show that the block $B[u]$ of any internal forced node is filled in H. The proof is by induction from the leaves to the root. Let v and w be respectively the leftmost and the rightmost child of u in this ordering. Since u is forced, $B[v]$ and $B[w]$ are both forced by definition of forced blocks. We show that $B[v]$ and $B[w]$ are also filled. If v is internal, $B[v]$ is filled by the induction hypothesis. The other possibility is that v is a leaf and $B[v]$ is full (this also settles the base case of our induction). In this latter case the interior of $B[v]$ is filled (by definition of a full block),

we must show that the border is also filled. By definition of a PQ-tree, the border of $B[v]$ is precisely the minimal separator $B[v] \cap B[v']$, where v' is the brother of v immediately following it in solidification π. By Lemma 3, this border becomes filled in H, and therefore the whole block $B[v]$ is filled. By symmetry, $B[w]$ is also filled in H. In particular, $B[v]$ and $B[w]$ are hit. This implies that for all children $\tilde{u}$ of u different from v and w, the cliques corresponding to the leaves of $T_{\tilde{u}}$ are strictly between K_L and K_R (see Definition 3) in the clique path given by π. Thus, Lemma 3 gives that the corresponding blocks $B[\tilde{u}]$ are filled in H.

We conclude that the whole block $B[u]$ is filled. □

We now define a set of *nice* orderings on the children of a node u, such that, using these orderings, the corresponding solidification yields a nice clique path. The idea is to group hit nodes together as much as possible and to place non-forced nodes on the left and right branches, if possible, so that we don't need to fill sets of nodes that could avoid to be filled. As it is defined below, a nice ordering is designed for nodes in $\{r\} \cup LB(\pi)$: nodes in $RB(\pi)$ will be in fact assigned the reverse order of a nice ordering.

Definition 7 *For each node u in the subtree rooted at r we define the set Π_u^{nice} of nice orders of the children of u as follows:*

1. *if u is degenerate, then Π_u^{nice} is the set of orders σ such that the hit children of u form an interval $I = [\![L_\sigma(u), R_\sigma(u)]\!]$ such that $R_\sigma(u)$ is the last element of σ and such that $L_\sigma(u)$ is forced only if all the elements of I are forced. In the particular case when $u = r$, we also require that $R_\sigma(r)$ is forced only if the elements strictly between $L_\sigma(r)$ and $R_\sigma(r)$ are forced.*

2. *if u is prime, then Π_u^{nice} is the set of valid orders $\sigma \in \{\sigma_u, \bar{\sigma}_u\}$ such that the first element of σ is forced only if the last one is forced too, and, if possible, the first element v of σ is such that $(N(x) \cap B[u]) \not\subseteq B[v]$.*

Every node of the subtree rooted at r admits at least one nice ordering of its children. Recall that for a solidification π, and for a node u in T, we denote by $\pi(u)$ the order defined by π on the children of u.

Definition 8 *A* nice solidification π *is a solidification such that $\pi(r)$ is a nice order, for every node $u \in LB(\pi)$, $\pi(u)$ is a nice order, and for every node $v \in RB(\pi)$, $\bar{\pi}(v)$ is a nice order.*

The following theorem is our main combinatorial tool toward computing a nice clique path.

Theorem 3 *A clique path corresponding to a nice solidification of the PQ-tree is a nice clique path.*

Proof. Fix a nice solidification π of the PQ-tree and let CP_G be the corresponding clique path. Denote by H the interval completion respecting CP_G, minimal for this property (recall that it is unique, by Lemma 3).

Claim 1 *Let u be an internal node such that $u = r$ or u is on the left branch or on the right branch of the nice solidification. If u is not forced, then $B[u]$ is not filled. More precisely there is a vertex y in the interior of $B[u]$, such that xy is not an edge of the completion H.*

The proof of the claim is very similar to the proof of Lemma 5. We proceed by induction from bottom to top. Assume w.l.o.g. that $u = r$ or u is in the left-branch, the other case can be treated symmetrically. Let v be the leftmost child of u in $\pi(u)$. If v is not $L_{\pi(u)}(u)$, then all the maximal cliques in the subtree of v are outside the $[\![K_L, K_R]\!]$ interval (see Definition 3) and the conclusion follows. Otherwise, if $v = L_{\pi(u)}(u)$, then v is also on the left branch. First consider the case where v is a leaf of the PQ-tree, then $B[v]$ is a clique of G. If v is full, by definition, it is also forced. It follows from the definition of a nice order that the last element of $\pi(u)$ is forced, and so is u. Since u is not forced, then v is necessarily not full. From Lemma 3, the only edges that have been added between x and $B[v] = K_L$ are in the minimal separator between $B[v]$ and the clique right to it in the clique path induced by π. In particular, there are no fill edges between x and the interior of $B[v]$. Since $B[v]$ is not full, its interior contains a vertex y as required (see Remark 2). Finally, in the case where v is not a leaf of the PQ-tree, again v is not forced, otherwise u would be forced too, by the definition of a nice order. Then, the conclusion follows from the induction hypothesis. This achieves the proof of the claim.

Assume by contradiction that H is not a minimal interval completion of G, thus there exists a minimal interval completion H' strictly contained in H. H' respects the clique path of some solidification π' of the PQ-tree of G. Choose this solidification π' as similar as possible to π, in the following sense: (1) the minimum depth d of a node u such that $\pi(u) \neq \pi'(u)$ is as large as possible (by depth we mean the distance from u to the root of the PQ-tree) and (2) subject to the first condition, the number of nodes of depth d such that the two solidifications differ on these nodes is as small as possible. Let u be a node of minimum depth, with $\pi(u) \neq \pi'(u)$. We prove that, in the solidification π', we can replace the solidification of the subtree rooted in u such that $\pi'(u)$ becomes $\pi(u)$, and the clique path defined by this new solidification gives rise to the same interval completion H'. From the remarks following Definition 5, our node u is necessarily in the set $\{r\} \cup LB(\pi) \cup RB(\pi)$. Moreover, observe that the left and the right branch of the two solidifications π and π' are the same from the root of the PQ-tree down to level d. Thus, u is also in $\{r\} \cup LB(\pi') \cup RB(\pi')$.

Consider the case $u = r$. If r is a prime node then $\pi'(r) = \bar{\pi}(r)$. We reverse, in solidification π', the whole subtree rooted in r. That is, $\pi'(r)$ is replaced by $\pi(r)$, and the solidification of each descendant of r is also reversed. Thus, in the new clique path the $[\![K_L, K_R]\!]$ interval is preserved (although reversed), so by Lemma 3 the new clique path is also a clique path respected by H', contradicting our choice of π'. Assume now that r is a degenerate node. We claim that the nodes of the interval I' between $L_{\pi'(u)}$ and $R_{\pi'(u)}$ are the same as the nodes of the interval I, defined similarly on π. Clearly each node v of I' is also a node of I, otherwise H' contains a fill edge from x to an interior vertex of $B[v]$, while this edge does not appear in H. Conversely, by definition of the set Π_r^{nice}, I is formed exactly by the hit children of u, and I' also contains all the hit children, so the two intervals I and I' contain the same nodes. The children of u which are not in I' can be put, in π', in the same order as in π, to the left of I'; the new solidification still induces the same minimal completion H', by Lemma 3.

It remains to show that we can change $\pi'(u)$ to make it coincide with $\pi(u)$ on interval I, without changing the minimal completion H' induced by π'. Let us first show that we can ensure that I and I' have the same endpoints. Let v be a non-forced endpoint of I'. Assume for contradiction that v is not an endpoint of I, it implies that the two endpoints v_L and v_R of I are non-forced, by definition of a nice ordering on the children of u. By the claim above, $B[v_L]$ and $B[v_R]$ are not filled in H, therefore they cannot be filled in H'. This contradicts the fact that at least one of v_L, v_R are not endpoints of I'. Thus v is an endpoint of I. Conversely, since, by the claim above, the block of a non forced endpoint w of I is not filled in H, and consequently is not filled in H', then w is an endpoint of I'. It follows that I and I' have the same non-forced endpoints. Up to reversing interval I' in π', we can make the non-forced endpoints of I and I' coincide in π and π'. If we have to reverse I' in order to do so, we also reverse the entire subtree of each of its non-forced endpoint (as explained previously), so that the minimal interval completion defined by the new solidification remains H'.

Now, if I (or equivalently I') has a forced endpoint, then all the children of u in the interior of interval I (or equivalently I') are forced. Then, we can reorder, in π', the forced nodes of I' so that $\pi'(u)$ coincide with $\pi(u)$ on I. Since these nodes are forced, reordering them does not change the minimal completion H' defined by the solidification. In the case where I has no forced endpoint, then the nodes in the interiors of I and I' are the same and we can rearrange the order in $\pi'(u)$ of the nodes in the interior of I' to make it coincide with their order in $\pi(u)$. Since at the end of these transformations, we have $\pi'(u) = \pi(u)$ and we did not change the completion H' defined by π', we get a contradiction with our choice of solidification π'.

Now we consider the case when $u \neq r$. Recall that u must be on one of the left or right branches. Assume w.l.o.g. that u is in the left-branch of the two solidifications. Again we distinguish the case when u is degenerate from the case when it is prime.

In the case when u is degenerate, we can apply exactly the same arguments as for the case "$u = r$ and u is degenerate" in order to prove that we can replace, in π', the ordering $\pi'(u)$ by the ordering $\pi(u)$.

It remains to treat the case when u is prime. Then $\pi'(u) = \bar{\pi}(u)$. Let v be the leftmost child of u in $\pi(u)$. We distinguish two subcases. The first subcase is when $N(x) \cap B[u]$ is not contained in $B[v]$. We show that $B[v]$ is filled in H. By the definition of a nice ordering on the children of u, there is another child v' of u such that $N(x) \cap B[v']$ is not contained in $N(x) \cap B[v]$. Consequently, since u is on the left branch of π' and, in $\pi'(u)$, the child v' of u is to the left of v (recall that $\pi'(u) = \bar{\pi}(u)$), the block $B[v]$ is filled in H', and so is filled in H too. This implies that $v = L_{\pi(u)}(u)$, otherwise v would be clean, and then not filled. Then, since v is on the left branch of π and is filled in H, from the claim above, v is forced. It follows, by construction of nice orderings on prime nodes, that the rightmost child of u in $\pi(u)$ is also forced, and so is u. By Lemma 5, $B[u]$ is filled in H'. Then, in π', we can reverse $\pi'(u)$ without changing the corresponding interval completion, which is a contradiction with our choice of π'.

The second subcase is when $N(x) \cap B[u] \subseteq B[v]$. By definition of nice orders, $N(x) \cap B[u]$ is also included in $B[w]$, where w is the rightmost child of u in $\pi(u)$. We claim that $N(x) \cap B[u]$ is contained in all cliques corresponding to leaves of T_u. For proving the claim, we refer to the data structure of the PQ-tree described in Section 5 and Lemma 7. For any vertex y of $B[v] \cap B[w]$, its primary pointer is u or an ancestor of u. If the primary pointer is u, then the secondary pointers are precisely v and w. In all cases, $B[v] \cap B[w]$ is contained in all cliques of T_u, and so is $N(x) \cap B[u]$. Recall that u is on the left branch of the solidifications π and π'. Let $CP(G)$ and $CP'(G)$ be the corresponding clique paths of G. Observe that $K_L^{CP(G)}$ and $K_L^{CP'(G)}$ are the rightmost leaves of T_u in the corresponding solidifications. Thus, the fill edges of $B[u]$ are exactly the intersection of $B[u]$ and of $B[u']$, where u' is the right-hand neighbour of u in π and π'. Again, in π', we can reverse $\pi'(u)$ without changing the interval completion.

This achieves the proof of our theorem. □

4 When the neighborhood of x is a clique

In this case, if G' is not already an interval graph (which will be tested in $\mathcal{O}(n)$ time using the algorithm of [4]), is it sufficient to add edges from x to a well chosen minimal separator of G [13]:

Lemma 6 *Assume that $N(x)$ is a clique and G' is not an interval graph. Let (K_1, K_2) be a couple of maximal cliques of G such that $N(x) \subseteq K_1$ and K_2 neighbors K_1 in some clique path CP_G of G, and $(K_1 \cap K_2) \setminus N(x)$ is inclusion-minimal among such couples. The graph H obtained by filling the minimal separator $K_1 \cap K_2$ is a minimal interval completion of G'*

Proof. First note that, since $N(x)$ is a clique, it is included in at least one maximal clique of G. Therefore, there is always at least one couple (K_1, K_2) satisfying the conditions of the lemma. Adding a clique $(K_1 \cap K_2) \cup N(x) \cup \{x\}$ between K_1 and K_2 in CP_G, and deleting the possible non-maximal cliques of H, clearly results in a clique path of H. Thus H is an interval completion of G'.

Now, suppose for contradiction that there exists an interval completion H' such that $E(H') \subsetneq E(H)$, that is $N_{H'}(x) \subsetneq N_H(x)$. Let $CP_{H'}$ be a clique path of H'. Clique $N_{H'}(x) \cup \{x\}$ is clearly maximal in H' and cannot be at the extremity of $CP_{H'}$ since in that case, in this extremal clique of the clique path, we could delete the vertices of $N_{H'}(x) \setminus N(x)$ to obtain a clique path of G', which would then be an interval graph. Thus, $N_{H'}(x) \cup \{x\}$ is surrounded by two maximal cliques K'_1 and K'_2 of G' in $CP_{H'}$, which implies that $N_{H'}(x) \supseteq (K'_1 \cap K'_2) \cup N(x)$. Since $N_{H'}(x) \subsetneq N_H(x)$ and $N_H(x) = (K_1 \cap K_2) \cup N(x)$, necessarily, we have $(K'_1 \cap K'_2) \setminus N(x) \subsetneq (K_1 \cap K_2) \setminus N(x)$. This is a contradiction with the minimality of $(K_1 \cap K_2) \setminus N(x)$. □

Actually, when $N(x)$ is a clique, either there is a unique clique K_1 of G containing $N(x)$, or there is some minimal separator of G containg $N(x)$. In both cases (although, for our purposes, we do need to distinguish them) we need to fill a well-chosen minimal separator of G. This task of finding the appropriate separator can be quite easily achieved in linear time in the size of G, but more effort is required to do it in $O(n)$ time.

5 The algorithm

This section describes our $O(n)$ time incremental algorithm for computing a minimal interval completion of $G + x$.

5.1 Data-structure: PQ-representation

The set of leaves of the PQ-tree is the set of maximal cliques of the graph. Since an interval graph has at most $n - 1$ maximal cliques, it follows that the number of leaves of the PQ-tree is $O(n)$. And since every internal node has at least two children, the total number of nodes of a PQ-tree is $O(n)$. However, to get a complete representation of the graph, cliques have to be encoded. Classically, this is done by assigning to each leaf of the PQ-tree

the list of nodes involved in the corresponding maximal clique of the graph. This makes the overall size of the structure inflate to $\Omega(n+m)$.

Here, we use a variant of the PQ-tree, called PQ-representation [4]: instead of being stored in the leaves, the vertices of G are stored in the internal nodes of the PQ-tree (thanks to the pointers defined below). This results in a complete representation of the graph which takes only $O(n)$ space and has deeper structural properties. The PQ-representation is essentially the same structure as the MPQ-tree introduced in [17]. However, we formalize it in a different way that fits better our purposes.

Recall that, throughout the paper, T is the PQ-tree of G.

Notation 1 (e_x) *We denote e_x for the least common ancestor of the leaves of T corresponding to a maximal clique of G containing x.*

Lemma 7 ([18]) *For any vertex x of an interval graph G, at least one of the two following conditions holds:*

1. *the maximal cliques of G containing x are exactly those corresponding to the leaves of T_{e_x}, or*
2. *e_x is a prime node and there exist two distinct children e_x^1, e_x^2 of e_x such that the maximal cliques of G containing x are exactly those corresponding to the union of the sets of leaves of T_u for any child u of e_x between e_x^1 and e_x^2 in σ_{e_x}.*

The PQ-representation of an interval graph G, denoted $PQ(G)$, is made of T 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 does not satisfy Condition 1 of Lemma 7 (but Condition 2). These pointers encode which maximal cliques of G (i.e. leaves of T) contain x.

Notation 2 *For each node u of T, we define the following sets:*
$X_u = \{y \in V \mid e_y = u$ *and y has no secondary pointers*$\}$
$Y_u = \{y \in V \mid e_y = u$ *and y has secondary pointers toward the children of u*$\}$

Note that, by definition, if u is degenerate then $Y_u = \varnothing$.

In addition to the pointers from the vertices of G toward the nodes of T, in order to achieve the desired complexity, we also store for each node $u \in T$ the list of vertices in X_u, the list of vertices in Y_u, and, if $parent(u)$ is prime, the list of vertices $y \in Y_{parent(u)}$ such that $e_y^1 = u$ and the list of vertices $z \in Y_{parent(u)}$ such that $e_z^2 = u$.

Since the number of nodes in T is $O(n)$ and since each vertex of G stores at most three pointers and is stored in at most four lists associated to some nodes of T, it follows that the total size of the PQ-representation is $O(n)$.

Preliminary computations We first need to collect some information about the nodes of T, we do so in a way which is very similar to the *bubble* phase of [3]'s algorithm. We want to determine for each node of the tree whether it is hit or clean and whether it is forced or not. To that purpose, we make use of the fact that blocks, their interior and their border can be easily derived from the PQ-representation : the interior of $B[u]$ is the set of vertices whose primary pointer points to a node of the subtree T_u rooted at u; the border of $B[u]$ is the set of vertices y whose primary pointer points to a strict ancestor of u and such that the interval defined by the secondary pointers of y (if it has some) contains an ancestor of u; and finally, block $B[u]$ is the union of these two sets.

For each node, we determine whether it is hit or clean by a bottom-up marking process of the tree. For each vertex $y \in N(x)$ we mark e_y as hit. Then, at each step the newly marked-hit nodes propagate this information to their parent, which becomes hit if it was not before. Then in time bounded by the size of the tree, which is $O(n)$, we can mark all nodes as being hit or clean.

In a similar way, we can determine full nodes and forced nodes. By parsing the list of neighbors of x, we first determine what we call *superficially full* nodes, which are those nodes such that $X_u \cup Y_u \subseteq N(x)$. Then, by a bottom-up process from the leaves to the root of the tree we are able to mark each node as being full or not and as being forced or not. All superficially full leaves are initially marked as full and forced. Then, all full and forced leaves forward this information to their parents. For any internal node u, if all its children are marked full and u is superficially full, then u is marked full and forced. Moreover, if u is degenerate and all its children are marked forced, then u is marked forced; and if u is prime and its first child and last child are both marked forced, then u is marked forced. Once an internal node is marked full or forced, it forwards this information to its parent. Again, this process takes $O(n)$ time.

We need to check whether $N(x)$ is a clique, and in the negative, we need to identify node r as in Lemma 4. Both goals are achieved by the following routine, named `BranchTree`. Start with the root as current node. At each step, if the current node u has a unique child v such that $N(x) \subseteq B[v]$, then make it become the new current node, otherwise stop the process. A degenerate node u satisfies the condition if and only if u has a unique hit child, and a prime node u satisfies it if and only if $L_{\sigma_u}(u) = R_{\sigma_u}(u)$.

From the definition of Routine `BranchTree` we deduce:

Claim 2 *The nodes w on which routine `BranchTree` stops satisfy exactly one of these two conditions:*

1. *w is degenerate and has at least two hit children, or w is prime and is such that $L_{\sigma_w}(w) <_{\sigma_w} R_{\sigma_w}(w)$.*

2. *w is a leaf, or w is a degenerate node and has no hit children, or w is prime and $R_{\sigma_w}(w) <_{\sigma_w} L_{\sigma_w}(w)$.*

It is straightforward to see that if Condition 1 is satisfied, then $N(x)$ is not a clique, and if Condition 2 is satisfied, then $N(x)$ is a clique. It follows that the above conditions are not only sufficient but are also necessary for $N(x)$ not to be, and respectively to be, a clique.

Note that for any node u in T, $L_{\sigma_u}(u) = min_{\sigma}\{v \in \mathcal{C}(u) \mid v \text{ is hit or } \exists y \in N(x), v = e^2_y\}$, and can then be computed in $O(|\mathcal{C}(u)| + |Y_u|)$. Thus, `BranchTree` determines whether $N(x)$ is a clique or not in $O(n)$ time.

5.2 When the neighborhood of x is not a clique

The particular case where $N(x)$ is a clique will be treated at the end of this section. For now, we focuss on the (more general) case where $N(x)$ is not a clique. Note that in this case, node w on which Routine `BranchTree` stops is nothing but node r. For sake of clearness, we describe the computation of a nice solidification in two steps (see the following two paragraphs), but they can be merged into a single top-down search from r to the leaves of T.

First step: computing nice orderings Thanks to the information collected initially about the nodes of T, we compute, during an arbitrary traversal of T_r, a nice ordering π_u for every node $u \in T_r$. Note that we compute a nice ordering for all the nodes of T_r and not only for those that will belong to $\{r\} \cup LB(\pi) \cup RB(\pi)$, where π is the nice solidification we intend to build. Of course, no special ordering is necessary for the other nodes, but it will be convenient for us not to particularize their treatment. We have to distinguish two cases depending on the label of u:

1. u is degenerate; all the clean children of u are placed at the beginning of π_u, and if u has at least one non-forced hit child then we place it right after the clean nodes in π_u, and if u has another non-forced hit child, we place it at the end of π_u.

2. u is prime; if the first child u_f of u in σ_u is forced or is such that $B[u] \cap N(x) \subseteq B[u_f]$, then we set $\pi_u = \bar{\sigma}_u$, otherwise we set $\pi_u = \sigma_u$.

A degenerate node u can be treated in $O(|\mathcal{C}(u)|)$ time – recall that $\mathcal{C}(u)$ denotes the set of children of u. In the treatment of a prime node u, the difficult part is to test whether $B[u] \cap N(x) \subseteq B[u_f]$. To that purpose, we have to check whether all the children of u different from u_f are clean and whether all the vertices of $Y_u \cap N(x)$ are such that $e^1_y = u_f$. This can be done in $O(|\mathcal{C}(u)| + |Y_u|)$ time. Thus the total running time of the first step is $O(\sum_{u \in T_r} |\mathcal{C}(u)| + |Y_u|) = O(n)$.

Second step: reversing orderings of the right branch The only thing left to do in order to obtain a nice solidification π is to identify the nodes of the right branch $RB(\pi)$ and to reverse the nice ordering computed for them in the previous step. (We point out that the first step also provides a nice solidification on the nodes of $LB(\pi)$) . We achieve this goal by following the path defined by $RB(\pi)$, from r to the leaf corresponding to K_R, while, at the same time, we modify π along this path.

We start with the current node being r: we do not modify π_r and we change the current node for its child $R_{\pi_r}(r)$. Then, at each step of the routine, we first reverse the order π_u affected to the current node u in the first part of the algorithm. Then, using the new value of π_u, we make $R_{\pi_u}(u)$ become the current node. We stop when the current node is a leaf. As noted previously, this leaf l_R is nothing else but the one corresponding to K_R in the clique path defined by the solidification π we computed during the present step of the algorithm. Remark that, by a similar routine, we can also identify the leaf l_L corresponding to K_L in the clique path defined by the solidification π, the difference being that we don't need to change solidification π during this routine.

For a node u, computing $R_{\pi_u}(u)$ can be done by a simple parse of its children and a parse of the vertices of Y_u, which takes $O(|\mathcal{C}(u)| + |Y_u|)$ time. Thus, the total running time of the second step is $O(\sum_{v \in RB(\pi)} |\mathcal{C}(v)| + |Y_v|) = O(n)$ time.

Finally, the time needed to compute a nice solidification π of the PQ-tree is $O(n)$, and we can identify K_L and K_R in the clique path defined by π within the same complexity.

Choosing the set of fill edges We proceed by computing a nice solidification π of the PQ-tree thanks to the PQ-representation, as shown in Section 5.2. This takes $O(n)$ time. Moreover, within the same complexity we can get the cliques K_L and K_R in the corresponding clique path CP_G which is, from Theorem 3, a nice clique path. Then, we compute the set F of nodes that has to be filled according to Lemma 3. We proceed as follows.

First, from the PQ-representation solidified by π, we compute the interval model of G based on the clique path corresponding to π, that is, the order σ on the maximal cliques of G corresponding to π and, for each vertex y of G, two pointers from y to the first and the last maximal clique of G containing y, in σ, denoted respectively K_y^1 and K_y^2. This can be done in $O(n)$ time by a simple Depth First Search of T. Thanks to this interval model, we can compute the minimal interval completion H described in Lemma 3: we must fill the set of vertices $F = \{y \in V \mid K_y^1 <_\sigma K_R \text{ and } K_L <_\sigma K_y^2\}$. Set F can be easily computed thanks to a scan of the vertices of G which takes $O(n)$ time.

5.3 Particular case where $N(x)$ is a clique

In this case a minimal fill is given by the following lemma derived from Lemma 6. When w is a prime node, σ is a valid ordering of its children and u is one of the children, we denote by $pred(u)$ (resp. $succ(u)$) the predecessor (resp. successor) of u in σ, if it exists. We also denote by $border(B[v])$ the border of a block v.

Lemma 8 *Let G be an interval graph and x a new incoming vertex such that $N(x)$ is a clique and $G+x$ is not an interval graph. Let w be the node on which Routine* `BranchTree` *stops.*

1. *If w is prime and $R_{\sigma_w}(w)$ is not its first child and $L_{\sigma_w}(w)$ is not its last child, then a minimal interval competion is obtained by filling X_w, the border of $B[w]$ and an inclusion-minimal set among sets $(Y_w \cap B[u] \cap B[succ(u)]) \setminus N(x)$ for $u \in [\![pred(R_{\sigma_w}(w)), L_{\sigma_w}(w)]\!]$.*

2. *Otherwise, we denote w' the lowest strict ancestor of w that is prime and such that its child v being an ancestor of w is neither its first child nor its last one. A minimal interval completion is obtained by filling an inclusion minimal set among $S_- \setminus N(x)$ and $S_+ \setminus N(x)$, where $S_- = Y_{w'} \cap B[pred(v)] \cap B[v]$ and $S_+ = Y_{w'} \cap B[v] \cap B[succ(v)]$, and filling $X_{w'}$ and the border of $B[w']$.*

Proof. We point out that the valid order used for w or w' is not relevant. Due to the symmetry of the definitions, both σ and $\overline{\sigma}$ yield the same results.

In case 1, the maximal cliques containing $N(x)$ are exactly the leaves of T_u, for every $u \in [\![R_{\sigma_w}(w), L_{\sigma_w}(w)]\!]$. First note that, by adding a clique containing x right between T_u and $T_{succ(u)}$ (resp. T_u and $T_{pred(u)}$) we obtain an interval completion of $G+x$ where the neighborhood of x is precisely $Y_w \cap B[u] \cap B[succ(u)]$, X_w and the border of $B[w]$ (resp. $Y_w \cap B[pred(u)] \cap B[u]$, X_w and the border of $B[w]$). Conversely, note that for two cliques of some clique path belonging to a same subtree T_u with $u \in [\![R_{\sigma_w}(w), L_{\sigma_w}(w)]\!]$, their intersection contains $Y_w \cap B[u] \cap B[succ(u)]$, $Y_w \cap B[pred(u)] \cap B[u]$, X_w and the border of $B[w]$. If these consecutive cliques are respectvely in T_u and $T_{succ(u)}$, their intersection contains $Y_w \cap B[u] \cap B[succ(u)]$, X_w and the border of $B[w]$. The symmetric situation holds when a clique is in T_u and the other in $T_{pred(u)}$. Thus, Lemma 6 implies that the fill proposed in case 1 is minimal.

In case 2, first note that node w' always exists. Otherwise, it would possible to place the cliques containing $N(x)$ at the beginning of a clique path of G', and then by adding a clique $N(x) \cup \{x\}$ in first position, we would obtain a clique path of $G+x$, which would then be an interval graph. Now, note that, in case 2, the maximal cliques containing $N(x)$ are all in T_v. And the intersection between two cliques in T_v must contain vertices in S_-,

IncMinSep(w, v_1, k)
1. $Smin \leftarrow S_1$
2. $D \leftarrow \varnothing$
3. **For** i from 1 to k **do**
4. $\quad D \leftarrow (D \setminus \{y \in Y_w \setminus N(x) \mid e_y^2 = v_i\}) \cup \{z \in Y_w \setminus N(x) \mid e_z^1 = v_i\}$
5. $\quad$ **If** $D = \varnothing$ **Then** $Smin \leftarrow S_{i+1}$
6. **End for**
7. Return $Smin$

Figure 1: Routine `IncMinSep`.

vertices in S_+, as well as vertices in $X_{w'} \cup B[w']$. Clearly, in any clique path, the neighbors of the cliques of T_v may be either in T_v or in $T_{pred(v)}$ or in $T_{succ(v)}$. Moreover, by choosing appropriately orders of the children for node w and its ancestors, it is possible to place the interval of cliques containing $N(x)$ at the beginning (resp. at the end) of the interval of the cliques of T_v. In this case, the intersection between the first (resp. last) clique of T_v, which contains $N(x)$, and its neighboring clique in $T_{pred(v)}$ (resp. $T_{succ(v)}$) is precisely $S_- \cup X_{w'} \cup border(B[w'])$ (resp. $S_+ \cup X_{w'} \cup border(B[w'])$). Then, Lemma 6 implies that the fill proposed in case 2 is minimal. □

Let us first examine case 2 of Lemma 8. Finding node w' is done by going up in the tree from node w until we reach a prime node arriving by one of its non-extremal child. Then, the computation of S_- and S_+ is done by parsing the vertices of $Y_{w'}$, and the inclusion between the two sets can be tested at the same time. At last, we have to compute the border of $B[w]$, which can be done by parsing X_u and Y_u for all the ancestors u of w. As a consequence, finding the set of fill edges in case 2 takes $O(n)$ time.

In case 1 of Lemma 8, we denote $v_1 < \ldots < v_k$ with $k \in \mathbb{N}$ the interval of children of w between $R_{\sigma_w}(w) = v_1$ and $L_{\sigma_w}(w) = v_k$. And we denote by v_0 and v_{k+1} respectively the predecessor of v_1 and the successor of v_k in σ_w. For $i \in [\![1, k+1]\!]$, let us denote S_i for the set of vertices of Y_w that belong to both $B[v_{i-1}]$ and $B[v_i]$, that is $S_i = \{y \in Y_w \mid e_y^1 \leq_{\sigma_w} v_{i-1} \text{ and } v_i \leq_{\sigma_w} e_y^2\}$.

According to Lemma 8, we have to find a i such that $S_i \setminus N(x)$ is inclusion-minimal. To that purpose we use Routine `IncMinSep`, see Figure 1.

In Routine `IncMinSep`, $Smin$ stores a minimal separator such that $Smin \setminus N(x)$ is inclusion-minimal among the separators $S_1, \ldots S_i$ considered so far (before loop number i). D stores the difference $(S_i \setminus N(x)) \setminus (Smin \setminus N(x))$. During iteration number i of the loop, D is updated by removing the vertices that are in $S_i \setminus N(x)$ but not in $S_{i+1} \setminus N(x)$ and by adding those that are in $S_{i+1} \setminus N(x)$ but not in $S_i \setminus N(x)$. Thus, the test of Line 5 determines

whether $S_{i+1} \setminus N(x) \subseteq Smin \setminus N(x)$. In the positive, $Smin$ is set to S_{i+1}. This implies that $Smin \setminus N(x)$ is included in $S_j \setminus N(x)$ for every separator S_j which is before $Smin$ in σ_w. Then, the routine goes on looking for some separator S_j which is after $Smin$ in σ_w and such that $S_j \setminus N(x)$ is included in $Smin \setminus N(x)$. If no such separator is found, then $Smin \setminus N(x)$ is inclusion-minimal; otherwise, $Smin$ is replaced and the search continues.

Clearly, each of the k iterations of the loop in routine `IncMinSep` takes time proportional to $|\{y \in Y_w \setminus N(x) \mid e_y^2 = v_i\} \cup \{z \in Y_w \setminus N(x) \mid e_z^1 = v_i\}|$. Thus, the total running time of the routine is $O(k + |Y_w|) = O(n)$. Once we identified the desired S_i, we fill S_i, X_w and the border of $B[w]$, according to Lemma 8. Again, finding the set of fill edges can be performed in $O(n)$ time.

5.4 Putting everything together

From Lemma 2, we can compute a minimal interval completion of graph G incrementally. We start from the empty graph, and we add the vertices of G one by one. At each step, when a new vertex x is added, we compute a minimal interval completion of the augmented graph by adding only edges incident to x.

After distinguishing, through the routine `BranchTree`, whether $N(x)$ is a clique or not, we use the algorithms above to compute the set of fill edges incident to x. Recall this can be done in $O(n)$ time.

Thus, the only thing left to do is to update the PQ-representation in order to perform the next incremental step. This is done by inserting x in G along with the edges between x and $N(x)$, plus the inclusion-minimal set of filled vertices. Thanks to the algorithm of [4], we obtain the updated PQ-representation of H in $O(n)$ time. Since an incremental completion step can be performed in $O(n)$ time, including the update cost of the data-structure, the total running time of our completion algorithm is $O(n^2)$, as stated by Theorem 1.

6 Conclusions and perspectives

We obtained an $O(n^2)$-time algorithm for the Minimal Interval Completion problem. This complexity is lower than those of the best algorithms for minimal chordal completion: $O(nm)$ in [23] or $o(n^{2.376})$ in [14]. This is somehow natural as interval graphs are "simpler" than chordal graphs. Nevertheless, this sheds a new light on the question of whether it is possible to achieve such a complexity for chordal completion. In particular, one may ask whether it is possible to mimic the approach followed here by using the intersection model of chordal graphs.

Concerning interval completion, there are two degrees of freedom in the choice of the minimal interval completion produced by our algorithm: we

can choose the order of arrival of the vertices of the graph and we can choose the nice solidification we use at each incremental step. It would be interesting to exploit these possibilities in order to get new heuristics or new approximation algorithms for minimum interval completion and pathwidth.

Acknowledgments We would like to thank Karol Suchan and Christophe Paul for useful discussions on the subject.

References

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, and H.T. Wareham. Parameterized complexity analysis in computational biology. *Comput. Appl. Biosci.*, 11:49–57, 1995.

[3] 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.

[4] C. Crespelle. Dynamic representations of interval graphs. In *WG*, volume 5911 of *LNCS*, pages 77–87, 2009. Full version: http://perso.ens-lyon.fr/christophe.crespelle/publications/DynInterval_long.pdf.

[5] C. Crespelle and I. Todinca. An $O(n^2)$-time algorithm for the minimal interval completion problem. In *TAMC*, volume 6108 of *LNCS*, pages 175–186, 2010.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[7] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian J. Math.*, 16:539–548, 1964.

[8] P.W. Goldberg, M.C. Golumbic, H. Kaplan, and R. Shamir. Four strikes against physical mapping of DNA. *J. Comput. Biol.*, 2:139–152, 1995.

[9] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier, second edition, 2004.

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

[11] P. Heggernes and F. Mancini. Minimal split completions. *Discrete Applied Mathematics*, 157(12):2659–2669, 2009.

[12] P. Heggernes, F. Mancini, and C. Papadopoulos. Minimal comparability completions of arbitrary graphs. *Discrete Applied Mathematics*, 156(5):705–718, 2008.

[13] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *ESA*, number 3669 in LNCS, pages 403–414. Springer Verlag, 2005.

[14] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. *SIAM J. Discrete Math.*, 19(4):900–913, 2005.

[15] R.M. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *25th Annual ACM Symposium on the Theory of Computing*, page 278–285, 1993.

[16] D.G. Kendall. Incidence matrices, interval graphs, and seriation in archeology. *Pacific J. Math.*, 28:565–570, 1969.

[17] N. Korte and R. H. Möhring. Transitive orientation of graphs with side constraints. In *WG*, pages 143–160, 1985.

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

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

[20] 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.

[21] T. Ohtsuki, H. Mori, T. Kashiwabara, and T. Fujisawa. On minimal augmentation of a graph to obtain an interval graph. *Journal of Computer and System Sciences*, 22(1):60–97, 1981.

[22] I. Rapaport, K. Suchan, and I. Todinca. Minimal proper interval completions. *Information Processing Letters*, 5:195–202, 2008.

[23] D. Rose, R. E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:146–160, 1976.

[24] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, page 183–217, 1972.

[25] K. Suchan and I. Todinca. Minimal interval completion through graph exploration. *Theoretical Computer Science*, 410(1):35–43, 2009.

[26] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.