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

Christophe Crespelle* Ioan Todinca†

Abstract

The minimal interval completion problem consists in adding edges to an arbitrary graph so that the resulting graph is an interval graph; the objective is to add an inclusion minimal set of edges, which means that no proper subset of the added edges can result in an interval graph when added to the original graph. 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 lower 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 , archeology , and clone fingerprinting . In addition, interval completion has been studied for its connection with another fundamental problem of computer science known as *chordal completion* (see [6] for a survey). A *chordal graph* is a graph that contains no chordless induced cycle on four or more vertices. The class of

*LIP6, Université Paris 6, christophe.crespelle@lip6.fr

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

chordal graphs properly includes all interval graphs. 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* [16]. Another chordal completion problem involves minimizing the maximum clique size of the completed graph. Indeed, this parameter is nothing else but *treewidth*, which is extensively studied, notably because many NP-complete problems become polynomial on graphs having this parameter bounded. Interval completion is similarly related to another famous graph parameters 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 (see [1] for references). But both kinds of solution are obtained on 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.

1.0.1 Related works.

The first algorithm solving the Minimal Chordal Completion problem in polynomial time is due to Rose, Tarjan and Lueker [15], 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 [10], 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. [13], 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 [9]. Using a completely different technique, Suchan and Todinca gave an $\mathcal{O}(nm)$-time algorithm in [17]. Note that several recent articles consider different types of minimal completion problems, e.g. into split graphs [7], comparability graphs [8] or proper interval graphs [14].

1.0.2 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 [13] and the $\mathcal{O}(nm)$ algorithm of [17]. 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 [10] which uses matrix multiplication techniques, ours is purely graph-theoretic. Like in [13], 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 [13] and simpler; and we rely on the results of [3] for efficiently updating, at each step, the PQ-tree of the new completion.

After giving, in next sections, some basic definitions and preliminary results, we present in Section 4 our main combinatorial tool for the minimal interval completion, 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 pairwise distinct 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 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 ([4]) *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 associating with 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.

Lemma 1 (see e.g. [5]) *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 [2] 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*). 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 in a plane drawing of the solidified rooted tree. The main property of the PQ-tree of G is that the set of orderings obtained this way is precisely the set of clique paths of G (see [2]).

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 $parent(u)$.

3 The vertex incremental approach

Let us observe that a minimal interval completion can be obtained incrementally. This result is due to [13].

Lemma 2 ([13]) *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, in 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.

3.0.3 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 induce a subpath P_x of CP_H. Now, let us get back to G. Delete x from every bag (clique) in CP_H, and possibly remove the bags that do not correspond to maximal cliques of G. This yields 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 induce 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 bag and possibly transforming the bordering separators into new bags of H (with x contained), we obtain a minimal interval completion of G'.

Definition 1 *A clique path CP_G is called* nice *if there exists a minimal interval completion H such that CP_G is obtained by pruning x from some clique path of H. In this case, we say that H* respects *CP_G.*

For obtaining a nice clique path we have to distinguish between two cases. For lack of space, we do not detail the case where the neighborhood of x in G' is a clique. We concentrate on the more general case where the neighborhood of x is not a clique.

4 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 [9], any clique path of G is associated with a canonical interval completion respecting it (Lemma 3 below). We need the following definition.

Definition 2 *Let CP_G be a clique path of G ordered left-to-right. If $N(x)$ is not a clique, we denote by K_L (resp. K_R) the leftmost (resp. rightmost) clique of CP_G such that x has a neighbor in $K_L \setminus K_{L+1}$ (resp. $K_R \setminus K_{R-1}$), in graph G'.*

Lemma 3 ([9]) *For any clique path CP_G of G, there is a unique interval completion H respecting CP_G which is inclusion-minimal. 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 any, 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 H is necessarily a minimal interval completion of G', but it may not be otherwise. Also note that any clique path obtained from a nice one CP_G by rearranging the maximal cliques of G in an order such that the cliques of the interval $[\![K_L, K_R]\!]$ of CP_G still form an interval whose endpoints are K_L and K_R (not necessarily in this order) is a nice clique path.

Before proving our main theorem (Theorem 3) which gives a way to obtain a nice clique path using 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. We point out that for all internal nodes u, the block $B[u]$ has a non-empty interior.

In the sequel, we denote by r the lowest node of the PQ-tree such that $N(x) \subseteq B[r]$. Since $N(x)$ is not a clique, node r is uniquely defined and is such that the interior of $B[r]$ contains at least two non-adjacent vertices both linked to x.

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 2, the difference being that here we consider blocks instead of cliques.

Definition 3 *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 element of σ if u has no such child v.*

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

By definition of node r, and since $N(x)$ is not a clique, for all valid orders σ of the children of r, $L_\sigma(r) <_\sigma R_\sigma(r)$. Thanks to the definition 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 4 *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 2). 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 4 below).

Definition 5 *A* forced *node is defined inductively 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 4 (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'.*

Consider a minimal interval completion of G'. It respects some clique path of G, obtained from a solidification π of the PQ-tree T. The proof can be made by induction from the leaves to the root of T. Basically, it relies on the fact that a forced node u has, by definition, a full descendant and therefore is hit. Moreover, its first and last children in π are forced, and so they are hit too. It implies that all the other children of u must be filled. The fact that the first (or last) child v of u is also filled can be obtained either from the induction hypothesis, if v is an internal node, or from the base case of the induction if v is a leaf. Finally, u which has all its children filled is filled itself.

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 suits for nodes in $\{r\} \cup LB(\pi)$: nodes in $RB(\pi)$ will be in fact assigned the reverse order of a nice ordering.

Definition 6 *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, and $R_\sigma(r)$ 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 the first element v of σ is such that $(N(x) \cap B[u]) \setminus B[v] \neq \varnothing$.*

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 7 *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.*

Idea of the 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 that is minimal for this property (recall that it is unique, by Lemma 3). Assume by contradiction that 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'. This will give us a contradiction completing our proof. From the remarks following Definition 4, our node u is necessarily in the set $\{r\} \cup LB(\pi) \cup RB(\pi)$. Moreover, observe that, by definition, 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 $\{r\} \cup LB(\pi') \cup RB(\pi')$.

For lack of space we cannot consider all cases: instead, we give as an example the case where $u \neq r$ is in the left-branch of the two solidifications and is a prime node. Then $\pi'(u) = \bar{\pi}(u)$. Let v be the leftmost child of u in $\pi(u)$. 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. It is possible to show that any non-forced node in $\{r\} \cup LB(\pi) \cup RB(\pi)$ is not filled in H. Then, v, which is on the left branch of π and is filled, is necessarily 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 4, $B[u]$ 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'.

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 [3]: 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 [11]. However, we formalize it in a different way that fits better our purposes.

Recall that T is the PQ-tree of G. We denote e_x for the least common ancestor of the leaves of T corresponding to a maximal clique of G containing x.

Lemma 5 ([12]) *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 5 (but Condition 2). These pointers encode which maximal cliques of G (i.e. leaves of T) contain x.

Notation 1 *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 $O(n)$.

5.2 Computing a nice solidification of the PQ-tree

We first collect information about the nodes of T. For each node, we determine whether it is hit or clean by a bottom-up marking process of the tree in which each node forwards its type to its parent, which is then able to determine its own type. In the same way, we can determine whether the nodes are forced or not. Both routines run in $O(n)$ time.

Before computing a nice solidification, we need to check whether $N(x)$ is a clique, and in the negative, we need to identify node r. These goals are achieved by the following routine. 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. At this point, it is easy to test whether $N(x)$ is a clique, and in the negative (which is the only case we treat in the following), the node on which the routine stopped is nothing but node r. This preliminary step takes $O(n)$ overall time. For sake of clearness, we describe the computation of a nice solidification in two steps, but they can be merged into a single top-down search from r to the leaves of T.

5.2.1 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 $u \in T_r$. For sake of simplicity of the presentation, we compute a nice ordering for all the nodes of T_r while it is necessary only for nodes of $\{r\} \cup LB(\pi) \cup RB(\pi)$, where π is the nice solidification we intend to build. 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 $e_y^1 = u_f$. This can be done in $O(|\mathcal{C}(u)| + |Y_u|)$ time. Thus the total running time of the first step is $O(\sum_{u \in T_r} |\mathcal{C}(u)| + |Y_u|) = O(n)$.

5.2.2 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. This is achieved 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. Similarly, we can identify the leaf l_L corresponding to K_L in the clique path defined by the solidification π, with the difference that, doing so, we don't need to change solidification π. This step takes $O(n)$ 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.

5.3 Overview of the algorithm

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.

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 search of the tree. 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.

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 $N(x) \cup F$. Thanks to the algorithm of [3], 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 $O(n^2)$.

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 [15] or $o(n^{2.376})$ in [10]. 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.

6.0.1 Acknowledgment.

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] 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.

[3] C. Crespelle. Dynamic representations of interval graphs. In *WG*, LNCS, 2009. To appear. http://www-npa.lip6.fr/∼crespell/publications/DynInt.pdf.

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

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

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

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

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

[9] P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In 13^{th} *Annual European Symposium on Algorithms (ESA'05)*, number 3669 in LNCS, pages 403–414. Springer Verlag, 2005.

[10] 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.

[11] N. Korte and R. H. Möhring. Transitive orientation of graphs with side constraints. In Linz Trauner, editor, 11^{th} *International Workshop on Graph-Theoretic Concepts in Computer Science (WG'85)*, pages 143–160, 1986.

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

[13] 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.

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

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

[16] 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.

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