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

Christophe Crespelle[1] and Ioan Todinca[2]

[1] LIP6, Université Paris 6
christophe.crespelle@lip6.fr
[2] LIFO, Université d'Orleans, BP 6759, F-45067 Orleans Cedex 2, France
ioan.todinca@univ-orleans.fr

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.

1 Introduction

Arbitrary graphs are complex combinatorial objects, thus it is very common to tempt to "simplify" them using few transformations. Classically, we want to know if an input graph can be modified into a graph with a special property (e. g. into a planar or chordal graph) using only a few operations like edge and/or vertex additions or deletions. Several important graph parameters like *treewidth*, *pathwidth* or *bandwidth* can be defined in terms of *graph completion* problems. In graph completions we are only allowed to add edges to the input graph $G = (V, E)$, in order to obtain a new graph $H = (V, E \cup F)$ with the required property. For example the treewidth problem consists in computing a *chordal* completion H – also known as *triangulation* – of the input graph G, such that the maximum cliquesize of H is as small as possible. Recall that a graph is *chordal* if it has no induced cycle with four or more vertices, thus a *chordal completion* of $G = (V, E)$ is any chordal supergraph $H = (V, E \cup F)$. Similarly, the pathwidth (resp. bandwidth) problem consists in computing an *interval* (resp. *proper interval*) completion H of the input graph G, and again the goal is to minimize the largest cliquesize of H. A graph is an interval graph if its vertices can be put into a one-to-one correspondence with a family of intervals of the real line such that two vertices are adjacent in the graph if and only if the corresponding intervals intersect.

Computing any of the parameters cited above, as well as many other parameters related to graph completions (minimum fill-in, profile...) is NP-hard. Therefore researchers turned their attention toward a heuristic approach, which consists in computing *minimal* completions of the input graph. We say that $H = (V, E \cup F)$ is a *minimal triangulation* (*minimal interval completion*) if no proper subgraph of H is a triangulation (interval completion) of G.

Related work. The minimal triangulation problem has been intensively studied and the algorithms are used as heuristics for treewidth and minimum fill-in (see [5] for a survey). The first algorithm solving this problem in polynomial time is due to Rose, Tarjan and Leuker [14], 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 [9], one can compute a minimal triangulation 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. [12], running in $\mathcal{O}(nm')$ time; here m' denotes the number of edges of the resulting minimal interval completion. A similar approach has been rediscovered in [8]. Using a completely different technique, Suchan and Todinca gave an $\mathcal{O}(nm)$ algorithm in [15]. Note that several recent articles consider different types of minimal completion problems, e.g. into split graphs [6], comparability graphs [7] or proper interval graphs [13].

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

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

Note that our approach is faster than the previous $\mathcal{O}(nm')$ algorithm of [12] and the $\mathcal{O}(nm)$ algorithm of [15]. It is also faster than the best algorithms for the minimal triangulation problem – which is somehow natural if we consider that interval graphs are "simpler" than chordal graphs. Like in [12], 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 of v_{i+1}. The second common feature is that we use PQ-trees in order to 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 [12], and we rely on the results of [2] 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. 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.

A graph H is an *interval* graph if continuous intervals 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. 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 ([3]). *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 is equivalent to 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.

Lemma 1 (see e.g. [4]). *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 [1] 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. 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.

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 [12].

Lemma 2 ([12]). *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 vertex x together with the edges between x and $N_{G_{i+1}}(x)$, 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 $G = (V, E)$. A new vertex x is added to G, together with a set of edges incident to x. For the rest of this document, let G' denote the graph $G + x$, that is, the graph G augmented with vertex x and the edges defining its neighborhood. The neighborhood of x in G' is simply denoted $N(x)$. We want to compute a minimal interval completion H of G', obtained by adding edges incident to x only. (Actually, the graph G will be given together with its PQ-tree, and we will also compute the PQ-tree of H.)

Take any clique path CP_H of H. 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 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 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 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 particular case where the neighborhood of x in G' is a clique, which is treated in Appendix A. From now on, we rather 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, in which we have to compute a nice clique path. We show later how to do so using the PQ-tree. But for now, let us first note that, as stated in [8], any clique path of G naturally gives a minimal interval completion respecting it (Lemma 3 below). We will need the following definition.

Definition 2. *Fix a clique path CP_G of G. In the case where $N(x)$ 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 ([8]). *For 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 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 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. We point out that for all internal nodes u, the block $B[u]$ has a non-empty interior (see Remark 1 in Appendix B).

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.

Notation 1 *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 there is no such vertex 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, $L_\sigma(r) <_\sigma R_\sigma(r)$.

Definition 3. *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.

Definition 4. *A* forced *node is defined inductively by: a node u of the PQ-tree is forced iff:*

- *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.*

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 – in particular they 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 2) in the clique path given by π. Thus, Lemma 3 gives that the corresponding blocks $B[\tilde{u}]$ are filled in H. 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). 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, hence so is the block $B[u]$. □

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.

Definition 5. *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.

Definition 6. *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 $v \in RB(\pi)$, $\bar{\pi}(u)$ 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. 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 4. 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 maximal cliques in the subtree of v are outside the $[\![K_L, K_R]\!]$ interval (see Definition 2) 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. 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 3, 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 a private 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 manage so 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 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 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'.

This achieves the proof of our theorem. □

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

In the classic PQ-tree, the maximal cliques of G, which correspond to the leaves of T, are stored in extension, that is, using the list of their vertices. Consequently, the size of the structure is $O(n + m)$, while the number of nodes in the PQ-tree is only $O(n)$. In the PQ-representation, the vertices of G are stored in the internal nodes of T (thanks to the pointers defined below) instead of being stored in its leaves. This results in an $O(n)$ space representation having deeper structural properties. The PQ-representation is essentially the same structure as the MPQ-tree introduced in [10]. 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 ([11]). *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 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. the 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 $O(n)$.

5.2 Computing a nice solidification of the PQ-tree

We first collect some 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, named `BranchTree`. Start with the root as current node. At each step, if the current node has a unique hit child, then make it become the new current node, otherwise stop the process. The node u on which the routine stops may either has at least two hit children, or no hit children. In the former case, $N(x)$ is not a clique, that is we are in the general case. In the latter case, if u is prime, we compute $L_{\sigma_u}(u) = min_{y \in N(x) \cap Y_u} e_y^2$ and $R_{\sigma_u}(u) = max_{y \in N(x) \cap Y_u} e_y^1$. If $L_{\sigma_u}(u) <_{\sigma_u} R_{\sigma_u}(u)$, then $N(x)$ is not a clique and we are again in the general case. In all the other cases, $N(x)$ is a clique and it is shown in Appendix A how to treat this particular case. Let us now concentrate on the general case. Note that in that case, the node u on which Routine `BranchTree` stops is nothing but node r.

Let us now consider the general case, where $N(x)$ is not a clique. 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.

First step: computing nice orderings. Thanks to the information computed 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 $e_y^1 = u_f$. This can be done in $O(|\mathcal{C}(u)| + |Y_u|)$ time. Thus the total running time of the first step is $O(n)$.

Second step: computing a nice solidification. 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 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 step, and then 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. 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. This takes $O(|\mathcal{C}(u)| + |Y_u|)$ time, and 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.

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 of the PQ-tree thanks to the PQ-representation, as shown in Section 5. 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, as shown in Appendix B. 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 that 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 [2], 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.

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

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
2. C. Crespelle. Dynamic representations of interval graphs. In *WG*, LNCS, 2009. To appear. http://www-npa.lip6.fr/∼crespell/publications/DynInt.pdf.
3. P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian J. Math.*, 16:539–548, 1964.
4. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. Elsevier, second edition, 2004.
5. P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Math.*, 306(3):297–317, 2006.
6. P. Heggernes and F. Mancini. Minimal split completions. *Discrete Applied Mathematics*, 157(12):2659–2669, 2009.
7. P. Heggernes, F. Mancini, and C. Papadopoulos. Minimal comparability completions of arbitrary graphs. *Discrete Applied Mathematics*, 156(5):705–718, 2008.
8. P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger. Minimal interval completions. In *ESA*, number 3669 in LNCS, pages 403–414. Springer Verlag, 2005. Extended version: http://www.univ-orleans.fr/lifo/prodsci/rapports/RR/RR2005/RR-2005-04.ps.gz.
9. 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.
10. N. Korte and R. H. Möhring. Transitive orientation of graphs with side constraints. In *WG*, pages 143–160, 1985.
11. N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18:68–81, 1989.
12. 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.
13. I. Rapaport, K. Suchan, and I. Todinca. Minimal proper interval completions. *Information Processing Letters*, 5:195–202, 2008.
14. D. Rose, R. E. Tarjan, and G.Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:146–160, 1976.
15. K. Suchan and I. Todinca. Minimal interval completion through graph exploration. *Theoretical Computer Science*, 410(1):35–43, 2009.

A The case where $N(x)$ 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 [2]), is it sufficient to add edges from x to a well chosen minimal separator of G:

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 $CP_{H'}$. Since some maximal clique of G' contains $N(x)$, at least one of K'_1 and K'_2 must do, let say K'_1 without loss of generality. Since $N_{H'}(x) \subsetneq N_H(x)$ and $N_{H'}(x) = (K'_1 \cap K'_2) \cup N(x)$ 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 $(K_1 \cap K_2) \setminus N(x)$. □

Algorithm for the case where $N(x)$ is a clique. Let u be the final node of Routine `BranchTree`, described in Section 5.2. $N(x)$ is a clique iff u has no hit children and

1. u is a leaf, or
2. u is a degenerate node, or
3. u is prime and $R_{\sigma_u}(u) <_{\sigma_u} L_{\sigma_u}(u)$.

From Lemma 6, when $Y_u \cap N(x) = \varnothing$ and $parent(u)$ is degenerate, we fill the border of $B[u]$, that is the set of vertices y such that e_y is an ancestor of u and either (i) y has no secondary pointers, or (ii) y has secondary pointers and there is an ancestor of u in the interval $[\![e^1_y, e^2_y]\!]$ of the children of e_y. This is done by scanning the sets X_v and Y_v for each node v on the path from u to the root of T. It takes $O(n)$ time.

If $Y_u \cap N(x) = \varnothing$ and $w = parent(u)$ is prime, then we denote S_- for the set of vertices in Y_w that belong both to the block of u and to the block of its predecessor in σ_w, if it has one. Symmetrically, we denote S_+ for the set of vertices in Y_w that belong both to the block of u and to the block of its successor in σ_w, if it has one. More formally, $S_- = \{y \in Y_w \mid e^1_y <_{\sigma_w} u \text{ and } u \leq_{\sigma_w} e^2_y\}$ and $S_+ = \{y \in Y_w \mid e^1_y \leq_{\sigma_w} u \text{ and } u <_{\sigma_w} e^2_y\}$. We get a minimal interval completion by filling the border of $B[u]$,

X_u and an inclusion-minimal set among S_- and S_+. Set S_- and S_+ can be computed by a simple scan of Y_w. Thus, the filling operation can be done in $O(n)$ time.

If $Y_u \cap N(x) \neq \varnothing$, we denote $v_1, \ldots, v_k$ with $k \in \mathbb{N}$ for the interval of children of u between $R_{\sigma_u}(u) = v_1$ and $L_{\sigma_u}(u) = v_k$. From Lemma 6, when v_1 is the first element of σ_u or v_k is the last element of σ_u, then we simply fill the border of $B[u]$ and X_u. Otherwise, we denote v_0 and v_{k+1} for respectively the predecessor of v_1 and the successor of v_k in σ_u. For $i \in [\![1, k+1]\!]$, we denote S_i for the set of vertices of Y_u that belong to both $B[v_{i-1}]$ and $B[v_i]$, that is $S_i = \{y \in Y_u \mid e_y^1 \leq_{\sigma_u} v_{i-1} \text{ and } v_i \leq_{\sigma_u} e_y^2\}$. According to Lemma 6, 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.

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

Fig. 1. Routine `IncMinSep`.

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_u. Then, the routine goes on looking for some separator S_j which is after $Smin$ in σ_u 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 $O(|Y_u|)$ time. Thus, the total running time of the routine is $O(k + |Y_u|) = O(n)$. Once we identified the desired S_i, we fill the border of $B[u]$, X_u and S_i. Again, the filling operation can be performed in $O(n)$ time. Therefore, this is also the complexity of the whole treatment of the particular case where $N(x)$ is a clique.

B Supplementary technical material

Remark 1. We point out that, for any *internal* node u of the PQ-tree, its interior is not empty. Indeed, 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 $B[u]$.

Computing the interval model corresponding to a given solidification of the PQ-tree. In other words, we aim at computing the order σ on the maximal cliques of G corresponding to the given solidification 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.

A simple depth-first search of the solidified PQ-tree gives σ. In order to assign their pointers to the vertices of G, we first assign to each node u of T two pointers toward respectively the first and the last element in σ that corresponds to a leaf of T_u. This can be easily done by a bottom-up process :

- any leaf of T is assigned two pointers toward its corresponding maximal clique, and
- any internal node u of T is assigned a first pointer which is the first pointer of its first child in $\pi(u)$ and a second pointer which is the second pointer of its last child in $\pi(u)$.

Then, for each vertex y of G that has no secondary pointers toward T in the PQ-representation, we assign to y the same pointers as e_y toward σ. If y has secondary pointers, it is assigned the first pointer of $min_{\pi(u)}\{e_y^1, e_y^2\}$ and the second pointer of $max_{\pi(u)}\{e_y^1, e_y^2\}$. The whole process cost is that of two searches of the tree and one search of the vertices of the graph, that is $O(n)$ time.