Extending Partial Representations of Subclasses of Chordal Graphs

Pavel Klavík · Jan Kratochvíl · Yota Otachi · Toshiki Saitoh

Abstract Chordal graphs are intersection graphs of subtrees of a tree T. We investigate the complexity of the partial representation extension problem for chordal graphs. A partial representation specifies a tree T' and some pre-drawn subtrees of T'. It asks whether it is possible to construct a representation inside a modified tree T which extends the partial representation (i.e, keeps the pre-drawn subtrees unchanged).

We consider four modifications of T' and get vastly different problems. In some cases, it is interesting to consider the complexity even if just T' is given and no subtree is pre-drawn. Also, we consider three well-known subclasses of chordal graphs: Proper interval graphs, interval graphs and path graphs. We give an almost complete complexity characterization.

We further study the parametrized complexity of the problems when parametrized by the number of pre-drawn subtrees, the number of components and the size of the tree T'. We describe an interesting relation with integer partition problems. The problem 3-Partition is used for all NP-completeness reductions. The extension of interval graphs when the space in T' is limited is "equivalent" to the BinPacking problem.

Pavel Klavík
Computer Science Institute, Faculty of Mathematics and Physics, Charles University in Prague, Malostranské náměstí 25, 118 00 Prague, Czech Republic. E-mail: klavik@iuuk.mff.cuni.cz

Jan Kratochvíl
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University in Prague, Malostranské náměstí 25, 118 00 Prague, Czech Republic. E-mail: honza@kam.mff.cuni.cz

Yota Otachi
School of Information Science, Japan Advanced Institute of Science and Technology. Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. E-mail: otachi@jaist.ac.jp

Toshiki Saitoh
Graduate School of Engineering, Kobe University, Rokkodai 1-1, Nada, Kobe, 657-8501, Japan. E-mail: saitoh@eedept.kobe-u.ac.jp

1 Introduction

Geometric representations of graphs and graph drawing are important topics of graph theory. We study *intersection representations* of graphs where the goal is to assign geometrical objects to the vertices of the graph and encode edges by intersections of these objects. An intersection-defined class restricts the geometrical objects and contains all graphs representable by these restricted objects; for example, interval graphs are intersection graphs of closed intervals of the real line. Intersection-defined classes have many interesting properties and appear naturally in numerous applications; for details see for example [12,25,22].

For a fixed class, its recognition problem asks whether an input graph belongs to this class; in other words, whether it has an intersection representation of this class. The complexity of recognition is well-understood for many classes; for example interval graphs can be recognized in linear-time [2,5].

We study a recently introduced generalization of the recognition problem called the *partial representation extension* [20]. Given a graph and a partial representation (a representation of an induced subgraph), it asks whether it is possible to extend this partial representation to a representation of the entire graph. This problems falls into the paradigm of extending partial solutions, an approach that has been studied frequently in other circumstances. Often it proves to be much harder than building a solution from scratch, for example for graph coloring [13,7]. Surprisingly, a very natural problem of extending partially represented graphs was only considered recently.

The paper [20] gives an $\mathcal{O}(n^2)$-algorithm for interval graphs and an $\mathcal{O}(nm)$-algorithm for proper interval graphs. Also, several other papers consider this problem. Interval representations can be extended in time $\mathcal{O}(n+m)$ [1,19]. Proper interval representations can be extended in time $\mathcal{O}(n+m)$ and unit interval representations in time $\mathcal{O}(n^2)$ [17]. Polynomial time algorithms are also described for function and permutation graphs [16], and for circle graphs [3].

In this paper, we follow this recent trend and investigate the complexity of partial representation extension of chordal graphs. Our mostly negative NP-completeness results are very interesting since chordal graphs are the first class for which the partial representation problem is proved to be strictly harder than the original recognition problem. Also, we investigate three well-known subclasses – proper interval graphs, interval graphs and path graphs, for which the complexity results are richer. We believe that better understanding of these simpler cases will provide tools to attack chordal graphs and beyond (for example, from the point of the parameterized complexity). For the conference version of this paper see [18].

1.1 Chordal Graphs and Their Subclasses

A graph is *chordal* if it does not contain an induced cycle of length four or more, i.e., each "long" cycle is triangulated. The class of chordal graphs, denoted by CHOR, is well-studied and has many wonderful properties. Chordal graphs are closed under induced subgraphs and possess the so called *perfect elimination schemes* which describe perfect reorderings of sparse matrices for the Gaussian elimination. Chordal graphs are perfect and many hard combinatorial problems are easy to

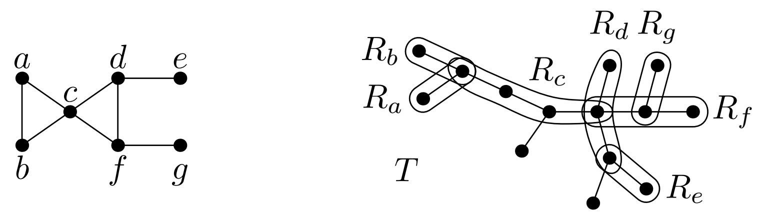

Fig. 1 An example of a chordal graph with one of its representations.

solve on chordal graphs: maximum clique, maximum independent set, k-coloring, etc. Chordal graphs can be recognized in time $\mathcal{O}(n+m)$ [23].

Chordal graphs have the following intersection representations [10]. For every chordal graph G there exists a tree T and a collection $\{R_v \mid v \in V(G)\}$ of subtrees of T such that $R_u \cap R_v \neq \emptyset$ if and only if $uv \in E(G)$. For an example of a chordal graph and one of its intersection representations, see Fig. 1.

When chordal graphs are viewed as *subtrees-in-tree* graphs, it is natural to consider two other possibilities: *subpaths-in-path* which gives *interval graphs* (INT), and *subpaths-in-tree* which gives *path graphs* (PATH). For example the graph in Fig. 1 is a path graph but not an interval one. Subpaths-in-path representations of interval graphs can be viewed as discretizations of the real line representations. Interval graphs can be recognized in $\mathcal{O}(n+m)$ [2,5] and path graphs in time $\mathcal{O}(nm)$ [11,24].

In addition, we consider proper interval graphs (PINT). An interval graph is a *proper interval graph* if it has a representation $\mathcal{R}$ for which $R_u \subseteq R_v$ implies $R_u = R_v$; so no interval is a proper subset of another one.[1] Proper interval graphs can be recognized in time $\mathcal{O}(n+m)$ [21,4]. From the point of our results, PINT behaves very similar to INT but there are subtle differences which we consider interesting. Also, partial representation extension of PINT is surprisingly very closely related to partial representation extension of unit interval graphs considered in [17]; see Section 1.4 for details.

1.2 Partial Representation Extension

For a class $\mathcal{C}$, we denote the recognition problem by RECOG($\mathcal{C}$). For an input graph G, it asks whether it belongs to $\mathcal{C}$, and moreover we may certify it by a representation. The partial representation extension problem denoted by REPEXT($\mathcal{C}$) asks whether a part of the representation given by the input can be extended to a representation of the whole graph.

A partial representation $\mathcal{R}'$ of G is a representation of an induced subgraph G'. The vertices of G' are called *pre-drawn*. A representation $\mathcal{R}$ extends $\mathcal{R}'$ if $R_v = R'_v$ for every $v \in V(G')$. The meta-problem we deal with is the following.

Problem:	REPEXT($\mathcal{C}$) (Partial Representation Extension of $\mathcal{C}$)
Input:	A graph G with a partial representation $\mathcal{R}'$.
Output:	Does G have a representation $\mathcal{R}$ that extends $\mathcal{R}'$?

[1]It is possible to define proper interval graphs differently: If $R_u \subseteq R_v$, then $R_v \setminus R_u$ is empty or a connected subpath of T. In other words, no interval can be placed in the middle of another interval. Our results can be easily modified for this alternative definition.

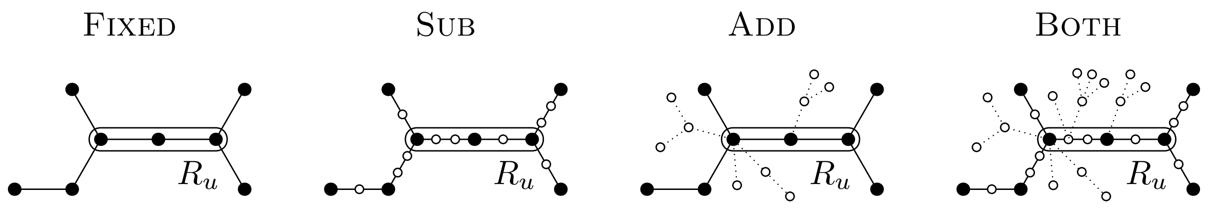

Fig. 2 The four possible modifications of T' with a single pre-drawn vertex u. The added branches in T are denoted by dots and new vertices of T are denoted by small circles.

In this paper, we study complexity of the partial representation extension problems for the classes CHOR, PATH, INT, and PINT in the setting of subtrees-in-tree representations. Here a partial representation $\mathcal{R}'$ fixes subtrees belonging to G' and also specifies some tree T' in which these subtrees are placed. A representation $\mathcal{R}$ is placed in a tree T which is created by some modification of T'. We consider four possible modifications and get different extension problems:

- Fixed – the tree cannot be modified at all, i.e, $T = T'$.
- Sub – the tree can only be subdivided, i.e., T is a subdivision of T'.[2]
- Add – we can add branches to the tree, i.e., T' is a subgraph of T.
- Both – we can both add branches and subdivide, i.e, a subgraph of T is a subdivision of T'. In other words T' is a topological minor of T.

We denote the problems by RepExt($\mathcal{C}, \mathfrak{T}$) where $\mathfrak{T}$ denotes the type. See Fig. 2.

Constructing a representation in a specified tree T' is interesting even if no subtree is pre-drawn, i.e., G' is empty; this problem is denoted by Recog*($\mathcal{C}, \mathfrak{T}$). Clearly, the hardness of the Recog* problem implies the hardness of the corresponding RepExt problem.

For PINT and INT classes, the types Add and Sub behave as follows. The type Add allows to extend the ends of the paths. The type Sub allows to expand the middle of the path. The difference is that if an endpoint of the path is contained in some pre-drawn subpath, it remains contained in it after the subdivision. The type Both makes the problems equivalent to the Recog and RepExt problems for the real line.

1.3 Our Results

We study the complexity of the Recog* and RepExt problems for all four classes and all four types. Our results are displayed in Fig. 3.

- All NP-complete results are reduced from the 3-Partition problem. The reductions are very similar and the basic case is Theorem 3 for RepExt(INT, Fixed) and RepExt(PINT, Fixed).
- The polynomial cases for INT and PINT are based on the known algorithm for recognition and extension. But since the space in T is limited, we adapt the algorithm for the specific problems.

[2]Let an edge $xy \in E(T')$ be subdivided (with a vertex z added in the middle). Then also pre-drawn subtrees containing both x and y are modified and contain z as well. So technically in the case of subdivision, it is not true that $R'_u = R_u$ for every pre-drawn interval, but from the topological point of view the partial representation is extended.

		PINT	INT	PATH	CHOR
Fixed	Recog*	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	NP-complete	NP-complete
	RepExt	NP-complete	NP-complete	NP-complete	NP-complete
Sub	Recog*	$\mathcal{O}(n+m)$ [21, 4]	$\mathcal{O}(n+m)$ [2, 5]	NP-complete	NP-complete
	RepExt	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	NP-complete	NP-complete
Add	Recog*	$\mathcal{O}(n+m)$ [21, 4]	$\mathcal{O}(n+m)$ [2, 5]	$\mathcal{O}(nm)$ [11, 24]	$\mathcal{O}(n+m)$ [23]
	RepExt	$\mathcal{O}(n+m)$	NP-complete	NP-complete	NP-complete
Both	Recog*	$\mathcal{O}(n+m)$ [21, 4]	$\mathcal{O}(n+m)$ [2, 5]	$\mathcal{O}(nm)$ [11, 24]	$\mathcal{O}(n+m)$ [23]
	RepExt	$\mathcal{O}(n+m)$ [17]	$\mathcal{O}(n+m)$ [1, 19]	**open**	NP-complete

Fig. 3 The table of the complexity of different problems for the four considered classes. The results without references are new results of this paper.

Every interval graph has a real-line representation in which all endpoints are at integer positions. But the result that RepExt(INT, Add) is NP-complete can be interpreted in the way that extending such representations is NP-complete. (Here, we require that also the non-pre-drawn intervals have endpoints placed at integer positions.) On the other hand, our linear-time algorithm for RepExt(PINT, Add) shows that integer-position proper interval representations can be extended in linear time.

For a subpaths-in-path partial representation, we assume that an input gives the endpoints of the pre-drawn subpaths sorted by the input from left to right. This allows us to construct algorithms in time $\mathcal{O}(n+m)$ which do not depend on the size of the path T'.

Parameterized Complexity. We study the parameterized complexity of these problems with respect to three parameters: The number k of pre-drawn subtrees, the number c of components and the size t of the tree T'. In some cases, the parametrization does not help and the problem is NP-complete even if the value of the parameter is zero or one. In other cases, the problems are fixed-parameter tractable (FPT), W[1]-hard or in XP.

The main result concerning parametrization is the following. The BinPacking problem is a well-known problem concerning integer partitions; more details in Section 3.4. For two problems A and B, we denote by $A \le B$ polynomial reducibility and by $A \le_{\rm wtt} B$ weak truth-table reducibility. (Roughly speaking, to solve A we may use a number of B-oraculum questions which is bounded by a computable function.)

Theorem 1 *For the number k of bins and pre-drawn subtrees, we get*

$$\textsc{BinPacking} \le \textsc{RepExt}(\mathsf{PINT}, \textsc{Fixed}) \le_{\rm wtt} \textsc{BinPacking}.$$

The weak truth-table reduction needs to solve 2^k instances of BinPacking.

1.4 Two Related Problems

We describe two problems which are closely related to our results.

The problem of *simultaneous representations* [14] asks whether there exist representations $\mathcal{R}_1, \dots, \mathcal{R}_k$ of graphs $G_1, \dots, G_k$ which are the same on the common part of the vertex set $I = V(G_i) \cap V(G_j)$ for all $i \neq j$. It is noted in [20] that the partial representation extension is closely related to the simultaneous representations. For instance, using simultaneous representations of interval graphs, we can solve their partial representation extension [1]. As we show in this paper, this is not the case for chordal graphs since RepExt of chordal graphs is NP-complete but their simultaneous representations are solvable in polynomial-time [14].

The partial representation extension problem of proper interval graphs described here is closely related to partial representations and the *bounded representation problem* of unit interval graphs [17]. In all these problems, one deals with interval representations in a limited space. So the techniques initially developed for unit interval graphs are easily used here for proper interval graphs. We note that the problems concerning unit interval graphs are more difficult since they involve computations with rational number positions.

2 Preliminaries

In this section, we describe the notation used in this paper. Also, we deal with two common concepts of the partial representation extension problems: Located and unlocated components, and groups of indistinguishable vertices.

Notation. We consider finite undirected simple graphs, i.e., graphs without loops and multiedges. As usual, we reserve n for the number of the vertices and m for the number of the edges of the main considered graph G. The set of its vertices is denoted by $V(G)$ and the set of its edges by $E(G)$. For a vertex $v \in V(G)$, we let $N(v) = \{x \mid vx \in E(G)\}$ denote the *open neighborhood* of v, and $N[v] = N(v) \cup \{v\}$ the *closed neighborhood* of v.

By P_n, we denote the path of the length n with $n+1$ vertices. For a tree, we call the vertices of degree larger than two *branch vertices* and the vertices of degree at most two *non-branch vertices*, and of course the vertices of degree one are called *leaves*.

The Type Lattice. The four types Fixed, Sub, Add, and Both form the lattice depicted in Fig. 4. For a type $\mathfrak{T}$, we denote by $\mathrm{Gen}(\mathfrak{T}, T')$ the set of all trees T which we can generate from T' using the modifications of the type $\mathfrak{T}$. In addition, if T' contains pre-drawn subtrees, the trees in $\mathrm{Gen}(\mathfrak{T}, T')$ contain these (possibly subdivided) pre-drawn subtrees as well. The ordering of the types given by the lattice has this property: If $\mathfrak{T} \le \mathfrak{T}'$, then $\mathrm{Gen}(\mathfrak{T}, T') \subseteq \mathrm{Gen}(\mathfrak{T}', T')$.

Whether a given instance is solvable depends on the set $\mathrm{Gen}(\mathfrak{T}, T')$; so if this set contains more trees, it only helps in solving the problem. Let $\mathfrak{T} \le \mathfrak{T}'$. If an

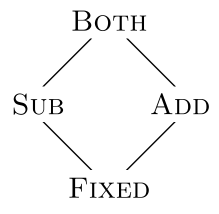

Fig. 4 The lattice formed by four types Fixed, Sub, Add, and Both.

instance of RECOG* or REPEXT is solvable for the type $\mathfrak{T}$, then it is solvable for the type $\mathfrak{T}'$ as well. Equivalently, if it is not solvable for $\mathfrak{T}'$, it is also not solvable for $\mathfrak{T}$.

For the types ADD and BOTH (and SUB for PINT and INT), the set $\mathrm{Gen}(\mathfrak{T}, T')$ contains a tree having an arbitrary tree T as a subtree. Therefore, the RECOG* problem for these types is equivalent to the standard RECOG problem, and we can use the known polynomial-time algorithms.

Topology of Components. The following property works quite generally for many intersection-defined classes of graphs, and works for all classes studied in this paper. The only required condition is that the sets R_v are connected subsets of some topological space, for example $\mathbb{R}^k$. (As a negative example, this property does not hold for *2-interval graphs*. A graph is a 2-interval graph if each R_v is a union of two closed intervals.) Let C be a connected component of G. Then the property is that for each representation $\mathcal{R}$, the set $\bigcup_{v \in C} R_v$ is a connected subset of the space, and we call this subset the *area of* C. Clearly, the areas of the components are pairwise disjoint.

For the classes PINT and INT, the areas of the components have to be ordered from left to right. Let us denote this ordering by $\blacktriangleleft$, so we have $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$. For different representations $\mathcal{R}$, we can have different orderings $\blacktriangleleft$. When no restriction is posed on $\mathcal{R}$, it is possible to create a representation in every of the $c!$ possible orderings.

Types of Components. For the partial representation extension problem, the graph G contains two types of components. A component C is called a *located component* if it has at least one vertex pre-drawn, i.e., $C \cap G'$ is non-empty. A component C is called an *unlocated component* if no interval is pre-drawn, i.e., $C \cap G' = \emptyset$. For located components, we have a partial information about their position. For unlocated components, we are much freer in their placement.

For the classes of interval graphs, the located components are ordered from left to right. An obvious necessary condition for an extendible partial representation is that the pre-drawn intervals of each component appear consecutively in $\mathcal{R}'$. Indeed, if C and C' are two distinct components, $u, v \in C$, $w \in C'$ and R_w is between R_u and R_v, then the partial representation is clearly not extendible. For every representation $\mathcal{R}$ extending $\mathcal{R}'$, the ordering $\blacktriangleleft$ has to extend the ordering $\blacktriangleleft'$ of the located components in $\mathcal{R}'$.

For many of the considered problems the unlocated components are irrelevant. For instance for REPEXT(INT, BOTH), we can extend the path T far enough to the right and place the unlocated components there, without interfering with the partial representation at all. On the other hand, for problems involving the types FIXED and SUB, the space in T is limited and the unlocated components have to be placed somewhere. In many cases, the existence of unlocated components is not only used for NP-completeness proofs but also necessary for the problems to be NP-complete.

Indistinguishable Vertices. Let u and v be two vertices of G such that $N[u] = N[v]$. These two vertices are called *indistinguishable* since they can be represented exactly the same, i.e., $R_u = R_v$. (This is a common property of indistinguishable vertices for all intersection representations). From the structural point of view, *groups of indistinguishable* vertices are not very interesting. The goal is to construct a pruned graph where each group is represented by a single vertex. For that, we need to be little careful since we cannot prune pre-drawn vertices.

For an arbitrary graph, its groups of indistinguishable vertices can be located in time $\mathcal{O}(n+m)$ [23]. We prune the graph in the following way. If u and v are indistinguishable and u is not pre-drawn, we eliminate u from the graph (and for the representation, we can put $R_u = R_v$). In addition, if two pre-drawn intervals are the same, we eliminate one of them. The resulting pruned graph has the following property: If two vertices u and v indistinguishable, they are both pre-drawn and represented by distinct intervals. For the rest of the paper, we expect that all input graphs are pruned.

Maximal Cliques. It is well known that subtrees of a tree possess the Helly property, i.e., every pairwise intersecting collection of subtrees has a non-empty intersection (which is again a subtree). Hence the following holds true for all classes of graphs considered. If K is a maximal clique of G, the common intersection $R_K = \cap_{u \in K} R_u$ is a subtree of T. This subtree R_K is not intersected by any other R_v for $v \notin K$ (otherwise K would not be a maximal clique). Thus the subtrees R_K corresponding to different maximal cliques are pairwise disjoint. For example, if $|T|$ is smaller than the number of maximal cliques of G, the graph is clearly not representable in T.

3 Interval Graphs

In this section, we deal with the classes PINT and INT. The results obtained here are used as tools for PATH and CHOR in Section 4.

Let $p_1, \dots, p_t$ be the vertices of the path T'. For a located component C, we say that a vertex p_i is *taken* by C if there exists a pre-drawn subpath of C containing p_i.

3.1 Structural Results

We describe two types orderings: Endpoint orderings for proper interval graphs and clique orderings for interval graphs. Also, we introduce an important concept called the minimum span of a component.

Endpoint Orderings of PINT. Each proper interval representation gives some ordering $\lhd$ of the intervals from left to right. This is the ordering of the left endpoints from left to right, and at the same time the ordering of the the right endpoints. The following lemma of [6] states that $\lhd$ is well determined:

Lemma 1 (Deng et al.) *For a component of a proper interval graph, the ordering $\lhd$ is uniquely determined up to a local reordering of the groups of indistinguishable vertices and the complete reversal.*

So for a connected graph, we have a partial ordering $<$ in which exactly the indistinguishable vertices are incomparable, and each $\lhd$ is a linear extension of either $<$, or its reversal. Corneil et al. [4] describes how this ordering can be constructed in time $\mathcal{O}(n+m)$. Since the graphs we consider are pruned, all incomparable vertices in $<$ are ordered by their positions in the partial representation. Thus we have at most two possibilities for $\lhd$ for each component C. (And two possibilities only if all pre-drawn vertices are indistinguishable.)

Minimum Spans of PINT. For the types FIXED and ADD, the space on the path T' is limited. So it is important to minimize the space taken by each component C. We call the minimum space required by C the *minimum span of* C, denoted by $\mathrm{minspan}(C)$. Let $\mathcal{R}$ be a proper interval representation of C extending $\mathcal{R}'$, and let p_i be the left-most vertex of T' taken by C and p_j the right-most one. Then

$$\mathrm{minspan}(C) = \begin{cases} \min_{\forall \mathcal{R}}\{j-i+1\} & \text{if some representation of } C \text{ exists,} \\ +\infty & \text{otherwise.} \end{cases}$$

A representation of C is called *smallest* if it realizes the minimum span of C.

Lemma 2 *For every component C, the value* $\mathrm{minspan}(C)$ *can be computed in time $\mathcal{O}(n+m)$, together with a smallest representation of C.*

Proof First, we deal with unlocated components, and later modify the approach for the located ones.

Case 1: An Unlocated Component. Since there are no indistinguishable vertices, we compute in time $\mathcal{O}(n+m)$ using the algorithm of [4] any ordering $\lhd$ for which we want to produce a representation as small as possible.

Let ℓ_i denote the left endpoint and r_i the right endpoint of the interval v_i. From the ordering $v_1 \lhd \cdots \lhd v_n$, we want to compute the common ordering $\lessdot$ of both the left and the right endpoints from left to right. The starting point is the ordering of just the left endpoints $\ell_1 \lessdot \cdots \lessdot \ell_n$. Into this ordering, we insert the right endpoints $r_1, \ldots, r_n$ one-by-one. A right endpoint r_i is inserted right before ℓ_j where v_j is the left-most non-neighbor of v_i on the right in $\lhd$; if such v_j does not exist, we append r_i to the end. For an example of $\lessdot$, see Fig. 5.

We build a smallest representation using $\lessdot$ as follows. Let $p_1, \ldots, p_k$ be the vertices of the tree T. We construct an assignment f which maps the endpoints of the intervals of C into T. Then for a vertex v_i we put

$$R_{v_i} = \{p_j \mid f(\ell_i) \le p_j \le f(r_i)\}.$$

The mapping f is constructed for the endpoints one-by-one, according to $\lessdot$. Suppose that the previous endpoint in $\lessdot$ has assigned a vertex p_i. If the current endpoint is a right endpoint and the previous endpoint is a left endpoint, we assign p_i to the current endpoint. Otherwise we assign p_{i+1} to it. For an example, see Fig. 5.

In total, the component needs $2n - \ell$ vertices of T where ℓ denotes the number of changes from a left endpoint to a right endpoint in the ordering $\lessdot$; in other words, $2n - \ell$ is the value of $\mathrm{minspan}(C)$. The total complexity of the algorithm is clearly $\mathcal{O}(n+m)$.

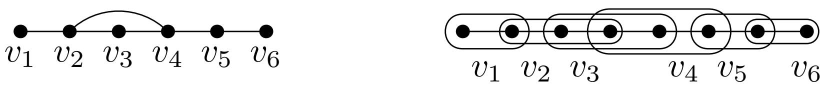
v_1 v_2 v_3 v_4 v_5 v_6
v_1 v_2 v_3 v_4 v_5 v_6

Fig. 5 The ordering $\lessdot$ is $\ell_1 \lessdot \ell_2 \lessdot r_1 \lessdot \ell_3 \lessdot \ell_4 \lessdot r_2 \lessdot r_3 \lessdot \ell_5 \lessdot r_4 \lessdot \ell_6 \lessdot r_5 \lessdot r_6$ for the component C on the left. The constructed smallest possible representation of the component C on the right, with $\mathrm{minspan}(C) = 8$.

To conclude the proof, we need to show that we construct a correct smallest representation of C. A property of $\lhd$ is that the closed neighborhood $N[v]$ of every vertex $v \in V(G)$ is consecutive in $\lhd$. If $v_iv_j \in E(G)$ and $v_i \lhd v_j$, then $\ell_i \lessdot \ell_j \lessdot r_i$, and so R_{v_i} intersects R_{v_j} (between $f(\ell_j)$ and $f(r_i)$). If $v_iv_j \notin E(G)$, then $r_i \lessdot \ell_j$. Thus r_i is placed on the left of ℓ_j in $\lessdot$, and $R_{v_i} \cap R_{v_j} = \emptyset$ as required.

Concerning the minimality notice that in a pruned graph, $\ell_i \neq \ell_j$ and $r_i \neq r_j$ hold for every $i \neq j$. We argue that we use gaps as small as possible. Only a right endpoint r_i following a left endpoint ℓ_j can be placed at the same position. The other case of a right endpoint r_i followed by a left endpoint ℓ_j requires a gap of size one; otherwise R_{v_i} would intersect R_{v_j} but $v_iv_j \notin E(G)$. So the gaps are minimal, we construct a smallest representation, and give the value $\mathrm{minspan}(C)$ correctly.

Case 2: A Located Component. We modify the above approach slightly to deal with located components. We already argued that there are at most two possible orderings $\lhd$ (since the indistinguishable vertices are ordered by the partial representation), and we just test both of them. Both orderings can be used if and only if all pre-drawn vertices belong to one group of indistinguishable vertices. Then these two orderings give the same $\mathrm{minspan}(C)$ but the minimum representations might be differently shifted, and we are able to construct both of them. If the pre-drawn intervals do not belong to one group, the ordering $\lhd$ is uniquely determined. (If it is compatible with the ordering of the pre-drawn intervals at all.)

We compute the common ordering $\lessdot$ exactly as before and place the endpoints in this ordering. The only difference is that the endpoints of the pre-drawn intervals are prescribed. So we start at the position of the left-most pre-drawn endpoint ℓ_i. We place the endpoints smaller in $\lessdot$ than ℓ_i on the left of ℓ_i as far to the right as possible. (We approach them in the reverse order exactly as above.) Then we proceed with the remaining endpoints in the order given by $\lessdot$. If the current endpoint is pre-drawn, we keep it as it is. Otherwise, we place it in the same way as above. The constructed representation is smallest and gives $\mathrm{minspan}(C)$. □

Clique Orderings of INT. Recall the properties of maximal cliques from Section 2. For a component C, we denote by $\mathrm{cl}(C)$ the number of maximal cliques of C. Let $\mathcal{R}$ be a representation of C. Since the subtrees R_K corresponding to the maximal cliques are pairwise disjoint, they have to be ordered from left to right. This ordering has the following well-known property [8]:

Lemma 3 (Fulkerson and Gross) *A graph is an interval graph if and only if there exists an ordering of the maximal cliques $K_1 < \cdots < K_{\mathrm{cl}(C)}$ such that for each vertex v the cliques containing v appear consecutively in this ordering.*

We quickly argue about the correctness of the lemma. Clearly, in an interval representation, all maximal cliques containing one vertex v appear consecutively. (Otherwise the clique in between would be intersected by R_v in addition.) On the other hand, having an ordering $<$ of the maximal cliques from the statement, we can construct a representation as follows. Assign a vertex p_i of T to each clique K_i, respecting the ordering $<$. For each vertex v, we assign $R_v = \{p_i \mid v \in K_i\}$. Since the maximal cliques containing v appear consecutively, each R_v is a subpath.

Minimum Spans of INT. We again consider the minimum span defined exactly as for proper interval graphs above. Clearly, $\mathrm{minspan}(C) \geq \mathrm{cl}(C)$. We show:

Lemma 4 *For an unlocated component C of an interval graph,* $\mathrm{minspan}(C) = \mathrm{cl}(C)$*. We can find a smallest representation in time $\mathcal{O}(n + m)$.*

Proof We start by identifying maximal cliques in time $\mathcal{O}(n + m)$, using the algorithm of Rose et al. [23]. To construct a smallest representation, we find an ordering from Lemma 3, using the PQ-tree algorithm [2] in time $\mathcal{O}(n+m)$. If such an ordering does not exist, the graph G is not an interval graph and no representation exists. If the ordering exists, we can construct a representation using exactly $\mathrm{cl}(C)$ vertices of the path as described above, by putting $R_v = \{p_i \mid v \in K_i\}$. $\square$

We note that this approach does not translate to located components, as in Lemma 2 for proper interval graphs. We prove in Corollary 1 that finding the minimum span for a located component is an NP-complete problem. (We prove this in the setting that the problem RepExt(INT, Add) is NP-complete. In the reduction, we ask whether a connected interval graph has the minimum span at most $(M + 1)k + 1$ for some integers k and M.)

3.2 The Polynomial Cases

First we deal with all polynomial cases.

Fixed Type Recognition. We just need to use the values of minimum spans we already know how to compute.

Proposition 1 *Both* Recog*(PINT, Fixed) *and* Recog*(INT, Fixed) *can be solved in time $\mathcal{O}(n + m)$.*

Proof We process the components $C_1, \ldots, C_c$ one-by-one and place them on T' from left to right. If $\sum_{i=1}^{c} \mathrm{minspan}(C_i) \le |T'|$, we can place the components using smallest representations from Lemma 2 for PINT, resp. Lemma 4 for INT. Otherwise, the path is too small and a representation cannot be constructed. $\square$

Add Type Extension, PINT. Again, we approach this problem using minimum spans and Lemma 2.

Proposition 2 *The problem* RepExt(PINT, Add) *can be solved in time $\mathcal{O}(n + m)$.*

Proof Since the path can be expanded to the left and to the right as much as necessary, we can place unlocated components far to the left. So we only need to deal with located components, ordered $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$ from left to right. We process the components from left to right. When we place C_i, it has to be placed on the right of C_{i-1}. We have (at most) two possible smallest representations corresponding to two different orderings of C_i. We test whether at least one of them can be placed on the right of C_{i-1}, and pick the one minimizing the right-most vertex of T taken by C_i (leaving the maximum possible space for $C_{i+1}, \ldots, C_c$). If neither of the smallest representations can be placed, the extension algorithm outputs “no”.

If the algorithm finishes, it constructs a correct representation. On the other hand, we place each component as far to the left as possible (while restricted by the previous components on the left). So if C_i cannot be placed, there exists no representation extending the partial representation. $\square$

Non-fixed Type Recognition. The only limitation for recognition of interval graphs inside a given path is the length of the path. In the three types SUB, ADD and BOTH, we can produce a path as long as necessary. (With the trivial exception $T' = P_0$ for SUB for which the instance is solvable if and only if $G = K_n$.) For a subpaths-in-path representation, the order of the endpoints of the subpaths from left to right is the only thing that matters, not the exact positions. In a tree T with at least $2n$ vertices, every possible ordering is realizable.

Thus the problems are equivalent to the standard recognition of interval graphs on the real line. The recognition can be solved in time $\mathcal{O}(n+m)$; see [21,4] for PINT, and [2,5] for INT.

Both Type Extension. This extension type is equivalent with the partial representation extension problems of interval graphs on the real line. Again only the ordering of the endpoints is important. The only change here is that some of the endpoints are already placed. By subdividing, we can place any amount of the endpoints between any two endpoints (not sharing the same position). Also, the path can be extended to the left and to the right which allows to place any amount of endpoints to the left of the left-most pre-drawn endpoint and to the right of the right-most pre-drawn endpoint. So any extending ordering can be realized in the BOTH type.

The partial representation extension problem for interval graphs on the real line was first considered in [20]. The paper gives algorithms for both classes INT and PINT, and does not explicitly deal with representations sharing endpoints but the algorithms are easy to modify. The results [1,19,17] show that both extension problems are solvable in time $\mathcal{O}(n+m)$.

Sub Type Extension. It is possible to modify the above algorithms for partial representation extension of INT and PINT. Instead of describing details of these algorithms, we simply reduce the problems to the type BOTH which we can solve in time $\mathcal{O}(n+m)$ (as discussed above):

Theorem 2 *The problems* REPEXT(PINT, SUB) *and* REPEXT(INT, SUB) *can be solved in time* $\mathcal{O}(n+m)$.

The general idea is as follows. The difference between between SUB and BOTH is that for the SUB type, we cannot extend the path T' at the ends. Suppose that some pre-drawn subpath R'_v contains say the left endpoint of T'. Then R'_v contains this endpoint also in T. So we are going to modify the graph G in such a way, that every representation of the BOTH type has to place everything on the right of R'_v.

Suppose first that the graph contains some unlocated components, and we show how to deal with them. We want to find one edge p_ip_{i+1} of T' which we can subdivide many times and place all unlocated components in between of p_i and p_{i+1} in T. We call an edge p_ip_{i+1} *expandable* if no located component C takes p_j and p_k such that $j \le i < i+1 \le k$.

Lemma 5 *Let G have at least one unlocated component, and let $\widetilde{G}$ be the graph constructed from G by removing all unlocated components. Then $\mathcal{R}'$ is extendible to $\mathcal{R}$ if and only if T' contains at least one expandable edge p_ip_{i+1} and $\mathcal{R}'$ is extendible to $\widetilde{\mathcal{R}}$ of $\widetilde{G}$.*

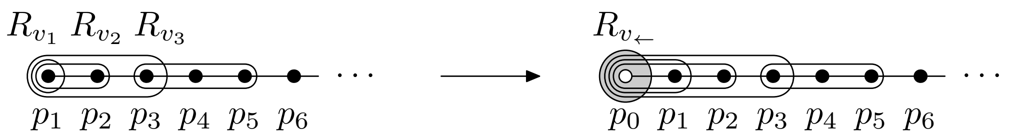
R_{v_1} R_{v_2} R_{v_3}
p_1 p_2 p_3 p_4 p_5 p_6
$R_{v_\leftarrow}$
p_0 p_1 p_2 p_3 p_4 p_5 p_6

Fig. 6 The three pre-drawn subpaths containing p_1 are $R_{v_1} = \{p_1\}$, $R_{v_2} = \{p_1, p_2\}$ and $R_{v_3} = \{p_1, p_2, p_3\}$. We add p_0 to T' and to $R_{v_1}, \dots, R_{v_3}$, and we introduce additional pre-drawn subpaths $R_{v_\leftarrow} = \{p_0\}$.

Proof Let $\mathcal{R}'$ be extendible to $\mathcal{R}$ and let C be one unlocated component placed in T such that it takes a vertex in between of p_i and p_{i+1} of T'. Clearly $\mathcal{R}'$ is extendible to $\widetilde{\mathcal{R}}$. And $p_i p_{i+1}$ is expandable since if there would be a located component $\widetilde{C}$ taking p_j and p_k, then C would split $\widetilde{C}$, contradicting existence of $\blacktriangleleft$ in $\mathcal{R}$; recall the definition of $\blacktriangleleft$ in Section 2.

For the other implication, we subdivide the expandable edge $p_i p_{i+1}$ many times such that we can place all unlocated components in this area. For located components, some of them have to be placed on the left of the unlocated components, and some on the right. We can subdivide all edges of T' enough to place the endpoints in the same order as in $\widetilde{\mathcal{R}}$. Thus we get $\mathcal{R}$ extending $\mathcal{R}'$. $\square$

Proof (Theorem 2) We describe the reduction for INT, and then we slightly modify it in the last paragraph for PINT. We deal with unlocated components using Lemma 5. We just need to check existence of an expandable edge for which we first compute the ordering $\blacktriangleleft$ of the located components (if it doesn't exist, the partial representation is clearly not extendible). If there is exactly one located component C, then at least one of p_1 and p_t is not taken by C, and say for p_1 we obtain an expandable edge $p_1 p_2$. And if there are at least two located components $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$, let p_i be the right-most vertex taken by C_1. Then $p_i p_{i+1}$ is clearly expandable. It remains to deal with located components.

Let us consider the endpoint p_1 of T'. In BOTH, we can attach in T a path P of any length on the left of p_1. If p_1 is not taken by C_1, we can create in T the same path P by subdividing $p_1 p_2$. But if p_1 is taken by C_1, we have to forbid P to be used in the construction of $\mathcal{R}$. We modify both the path T' and the graphs G, and we show that any representation $\mathcal{R}$ extending $\mathcal{R}'$ is realized in T in between of p_1 and p_t.

The modification is as follows. Let $v_1, \dots, v_k \in C_1$ be all pre-drawn subpaths such that $p_1 \in R'_{v_1}, \dots, R'_{v_k}$. First, we extend the path by one by adding p_0 attached to p_1. We introduce an additional pre-drawn vertex $v_\leftarrow$ adjacent exactly to $v_1, \dots, v_k$ in G. We put $R'_{v_\leftarrow} = \{p_0\}$ and we modify $R'_{v_i} = R'_{v_i} \cup \{p_0\}$. See Fig. 6. Indeed, we proceed exactly the same on the other side of T'; if p_t is taken by C_c, we introduce p_{t+1} and $v_\rightarrow$.

We use the described algorithm for REPEXT(INT, BOTH) for the modified graph and the modified path, which runs in time $\mathcal{O}(n+m)$. We obtain a representation $\mathcal{R}$ extending $\mathcal{R}'$ if it exists. If $\mathcal{R}$ does not exist, then the original problem is clearly not solvable. It remains to argue that if $\mathcal{R}$ exists, then we can either construct a solution for the original SUB type problem, or we can prove that it is not solvable.

We deal only with the left side of T; for the right side the argument is symmetrical. If T' is not modified on the left side, then the edge $p_1 p_2$ can be subdivided as necessary and we are equivalent with the BOTH type. Suppose that p_0 is added. There are no unlocated components, and so everything with the exception of $v_1, \dots, v_k$ has to be represented on the right of $v_\leftarrow$ which is placed on p_0.

We need to argue the issue that the newly added edge p_0p_1 can be subdivided in T. There are the following two cases:

- *Case 1.* If $|R'_{v_i}| \geq 3$ for each i, i.e, p_1 and p_2 belong to each R'_{v_i}, the subdivision of p_0p_1 is equivalent to the subdivision of p_1p_2 which is correct in the original Sub type problem. So nothing needs to be done.
- *Case 2.* Let $|R'_{v_i}| = 2$ for some i, so $R'_{v_i} = \{p_0, p_1\}$. Then $N(v_i) \setminus v_\leftarrow$ has to form a complete subgraph of G, otherwise the starting partial representation having $R'_{v_i} = \{p_1\}$ would not be extendible. We revert the subdivision of p_0p_1 by modifying $\mathcal{R}$ as follows. Let $p'_1, \dots, p'_s$ be the new vertices of T created by the subdivision of p_0p_1. For each $v \in N(v_i) \setminus v_\leftarrow$, we set $R_v = R_v \setminus \{p'_1, \dots, p'_s\} \cup \{p_1\}$, and we remove $p'_1, \dots, p'_s$ by contractions. Clearly, the resulting representation is correct and still extends $\mathcal{R}'$.

By removing p_0 and the vertices attached to it on the left, p_{t+1} and the vertices attached to it on the right, $v_\leftarrow$ and $v_\rightarrow$ (of course, only if they are added), we obtain a correct representation of G inside a subdivision of T' extending the partial representation $\mathcal{R}'$.

Concerning PINT, we use almost the same approach. The only difference is that we append two vertices p_0 and $\bar{p}_0$ (resp. p_{t+1} and $\bar{p}_{t+1}$) to the end of T', and we put $R'_{v_\leftarrow} = \{\bar{p}_0, p_0\}$ (resp. $R'_{v_\rightarrow} = \{p_{t+1}, \bar{p}_{t+1}\}$), so the modified partial representation is proper. $\square$

3.3 The NP-complete Cases

The basic gadgets of the reductions are paths. They have the following minimum spans.

Lemma 6 *For* INT*,* $\mathrm{minspan}(P_n) = n$*. For* PINT *and* $n \geq 2$*,* $\mathrm{minspan}(P_n) = n + 2$*.*

Proof For INT, the number of the maximal cliques of P_n is n. For PINT, the ordering $\lessdot$ is

$$\ell_0 \lessdot \ell_1 \lessdot r_0 \lessdot \ell_2 \lessdot r_1 \lessdot \cdots \lessdot \ell_i \lessdot r_{i-1} \lessdot \cdots \lessdot \ell_n \lessdot r_{n-1} \lessdot r_n.$$

There are n changes from ℓ_i to r_{i-1} and P_n has $n+1$ vertices. So the minimum span equals $2(n+1) - n = n + 2$. $\square$

We reduce the problems from 3-Partition. An input of 3-Partition consists of positive integers k, M and $A_1, \dots, A_{3k}$ such that $\frac{M}{4} < A_i < \frac{M}{2}$ for each A_i and $\sum A_i = kM$. It asks whether it is possible to partition A_i's into k triples such that the sets A_i of each triple sum to exactly M.[3] This problem is strongly NP-complete [9] which means that it is NP-complete even when the input is coded in unary, i.e., all integers are of polynomial sizes.

Theorem 3 *The problems* RepExt(PINT, Fixed) *and* RepExt(INT, Fixed) *are* NP*-complete.*

[3]Notice that if a subset of A_i's sums to exactly M it has to be a triple due to the size constraints.

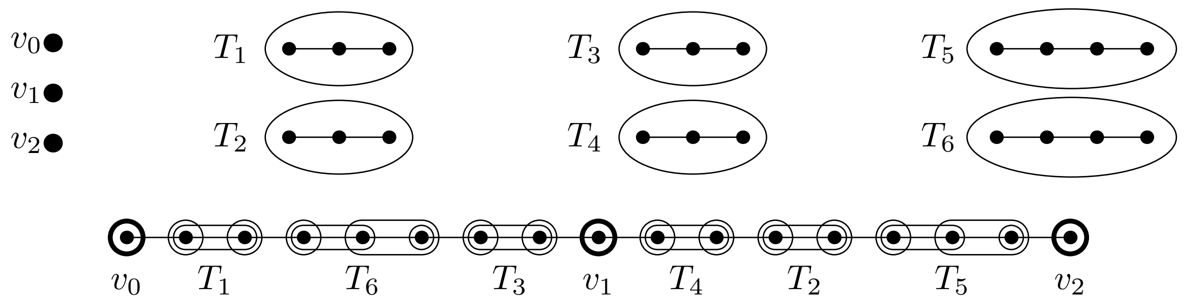

Fig. 7 An example of the reduction for the following input of 3-Partition: $k = 2$, $M = 7$, $A_1 = A_2 = A_3 = A_4 = 2$ and $A_5 = A_6 = 3$. On top, the constructed interval graph is depicted. On bottom, the partial representation (depicted in bold) is extended.

Proof We use almost the same reductions for both PINT and INT. For a given input of 3-Partition (with $M \geq 4$), we construct a graph G and its partial representation as follows.

As the fixed tree we choose $T' = P_{(M+1)k}$, with the vertices $p_0, \ldots, p_{(M+1)k}$. The graph G contains two types of gadgets as separate components. First, it contains $k + 1$ *split gadgets* $S_0, \ldots, S_k$ which split the path into k gaps of the size M. Then it contains $3k$ *take gadgets* $T_1, \ldots, T_{3k}$. A take gadget T_i takes in each representation at least A_i vertices of one of the k gaps.

For these reductions, the gadgets are particularly simple. The split gadget S_i is just a single pre-drawn vertex v_i with $R_{v_i} = \{p_{(M+1)i}\}$. The split gadgets clearly split the path into the k gaps of the size M. The take gadget T_i is P_{A_i} for INT, resp. P_{A_i-2} for PINT. According to Lemma 6, $\text{minspan}(T_i) = A_i$. The representation is extendible if and only if it is possible to place the take gadgets into the k gaps. For an example, see Fig. 7. The reduction is clearly polynomial.

To conclude the proof, we show that the partial representation is extendible if and only if the corresponding 3-Partition input has a solution. If the partial representation is extendible, the take gadgets T_i are divided into the k gaps on the path which gives a partition. Based on the constraints for the sizes of A_i's, each gap contains exactly three take gadgets of the total minimum span M; thus the partition solves the 3-Partition problem. On the other hand, a solution of 3-Partition describes how to place the take gadgets into the k gaps and construct an extending representation. □

Corollary 1 *The problem* RepExt(INT, Add) *is* NP*-complete.*

Proof We use the above reduction for INT with one additional pre-drawn interval v attached to everything in G. We put $R_v = \{p_0, \ldots, p_{(M+1)k}\}$, so it contains the whole tree T'. Since a representation of each take gadget T_i has to intersect R_v, it has to be placed inside of the k gaps as before. □

We note that the above modification does not work for proper interval graphs. Indeed, this is not very surprising since Proposition 2 states that the problem RepExt(PINT, Add) can be solved in time $\mathcal{O}(n + m)$.

3.4 The Parameterized Complexity

In this subsection, we study the parameterized complexity. The parameters are the number c of components, the number k of pre-drawn intervals and the size t of the path T'.

By the Number of Components. In the reduction of Theorem 3, one might ask whether it is possible to make the reduction graph G connected. For INT, it is indeed possible to add a universal vertex adjacent to everything in G, and thus make G connected as in the proof of Corollary 1. The following result answers this question for PINT negatively (unless $\mathsf{P} = \mathsf{NP}$):

Proposition 3 *The problem* $\textsc{RepExt}(\mathsf{PINT}, \textsc{Fixed})$ *is fixed-parameter tractable in the number c of components, solvable in time $\mathcal{O}((n+m)c!)$.*

Proof There are $c!$ possible orderings $\blacktriangleleft$ of the components from left to right, and we test each of them. (The located components force some partial ordering $\blacktriangleleft$ so we need to test less then $c!$ orderings; see below the proof for details.) We show that for a prescribed ordering $\blacktriangleleft$ of the components, we can solve the problem in time $\mathcal{O}(n+m)$; thus gaining the total time $\mathcal{O}((n+m)c!)$. We solve the problem almost the same as in the proof of Proposition 2. The only difference is that we deal with all components instead of only the located ones.

We process the components from left to right. When we process C_i, we place it on the right of C_{i-1} as far to the left as possible. For the unlocated C_i, we can take any smallest representation. For the located C_i, we test both smallest representations and take the one placing the right-most endpoint of C_i further to the left. We construct the representation in time $\mathcal{O}(n+m)$. For the correctness of the algorithm see the proof of Proposition 2 for more details. $\square$

We note that for NP-hardness of the problem $\textsc{RepExt}(\mathsf{PINT}, \textsc{Fixed})$ it is necessary to have some pre-drawn subpaths. On the other hand, also some unlocated components are necessary. If all the components were located, there would be a unique ordering $\blacktriangleleft$ and we could test it in time $\mathcal{O}(n+m)$ as described above. In general, for c components and c' located components, we need to test only $\frac{c!}{c'!}$ different orderings.

By the Number of Pre-drawn Intervals. In the reduction in Theorem 3, we need to have k pre-drawn intervals. One could ask, whether the problems become simpler with a small number of pre-drawn intervals. We answer this negatively. For PINT, the problem is in XP and $\mathsf{W[1]}$-hard with respect to k. For INT, we only show that it is $\mathsf{W[1]}$-hard.

There are two closely related problems $\textsc{BinPacking}$ and $\textsc{GenBinPacking}$. In both problems, we have k bins and n items of positive integer sizes. The question is whether we can pack (partition) these items into the k bins when the volumes of the bins are limited. For $\textsc{BinPacking}$, all the bins have the same volume. For $\textsc{GenBinPacking}$, the bins have different volumes. Formally:

Problem:	$\textsc{BinPacking}$
Input:	Positive integers k, ℓ, V, and $A_1, \dots, A_\ell$.
Output:	Does there exist a k-partition $\mathcal{P}_1, \dots, \mathcal{P}_k$ of $A_1, \dots, A_\ell$ such that $\sum_{A_i \in \mathcal{P}_j} A_i \le V$ for every $\mathcal{P}_j$.

Problem:	GenBinPacking
Input:	Positive integers k, ℓ, $V_1, \dots, V_k$, and $A_1, \dots, A_\ell$.
Output:	Does there exist a k-partition $\mathcal{P}_1, \dots, \mathcal{P}_k$ of $A_1, \dots, A_\ell$ such that $\sum_{A_i \in \mathcal{P}_j} A_i \le V_j$ for every $\mathcal{P}_j$.

Lemma 7 *The problems* BinPacking *and* GenBinPacking *are polynomially equivalent.*

Proof Obviously BinPacking is a special case of GenBinPacking. On the other hand, let k, ℓ, $V_1, \dots, V_k$, and $A_1, \dots, A_\ell$ be an instance of GenBinPacking. We construct an instance k', ℓ', V', and $A'_1, \dots, A'_{\ell'}$ of BinPacking as follows. We put $k' = k$, $\ell' = \ell + k$ and $V' = 2 \cdot \max V_i + 1$. The sizes of the first ℓ items are the same, i.e, $A'_i = A_i$ for $i = 1, \dots, \ell$. The additional items $A'_{\ell+1}, \dots, A'_{\ell+k}$ are called *large* and we put $A'_{\ell+i} = V' - V_i$ for $i = 1, \dots, k$.

Each bin has to contain exactly one large item since two large items take more space than V'. After placing large items into the bins, we obtain the bins of the remaining volumes $V_1, \dots, V_k$ in which we have to place the remaining items. This corresponds exactly to the original GenBinPacking instance. □

If the sizes of items are encoded in binary, the problem is NP-complete even for $k = 2$. The more interesting version which we use here is that the sizes are encoded in unary so all sizes are polynomial. In such a case, the BinPacking problem is known to be solvable in time $t^{\mathcal{O}(k)}$ using dynamic programming where t is the total size of all items. And it is W[1]-hard with respect to the parameter k [15]. The similar holds for RepExt(PINT, Fixed):

Proof (Theorem 1) For a given instance of the BinPacking problem, we can solve it by RepExt(PINT, Fixed) in a similar manner as in the reduction in Theorem 3. As T', take a path $P_{(V+1)k}$. As G, take $P_{A_i - 2}$ for each A_i and the pre-drawn vertices $v_0, \dots, v_k$ such that $R_{v_i} = \{p_{(V+1)i}\}$. The rest of the argument is exactly as in the proof of Theorem 3.

Now, we want to solve RepExt(PINT, Fixed) using 2^k instances of GenBinPacking (which is polynomially equivalent to BinPacking), where k is the number of pre-drawn intervals.

First we deal with located components $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$. For each component, we have two possible orderings $\lhd$ and using Lemma 2 we get (at most) two possible smallest representations which might be differently shifted. In total, we have at most $2^c \le 2^k$ possible representations keeping $C_1, \dots, C_c$ as small as possible leaving maximal gaps for unlocated components. We test each of these 2^c representations.

Let $C'_1, \dots, C'_{c'}$ be the unlocated components. For each C'_i, we compute $\mathrm{minspan}(C'_i)$ using Lemma 2. The goal is to place the unlocated components into the $c + 1$ gaps between representations of the located components $C_1, \dots, C_c$. We can solve this problem using GenBinPacking as follows. We have $k + 1$ bins of the volumes equal to the sizes of the gaps between the representations of $C_1, \dots, C_c$. We have c' items of the sizes $A_i = \mathrm{minspan}(C'_i)$.

A solution of GenBinPacking tells how to place the unlocated components into the k gaps. If there exists no solution, this specific representation of the located components cannot be used. We can test all 2^c possible representations of the located components. Thus we get the required weak truth-table reduction. □

Corollary 2 *The problem* $\textsc{RepExt}(\mathsf{PINT}, \textsc{Fixed})$ *is* $\mathsf{W[1]}$*-hard and belongs to* XP*, solvable in time* $n^{\mathcal{O}(k)}$ *where* k *is the number of pre-drawn intervals.*

Proof Both claims follow from Theorem 1. □

Proposition 4 *The problems* $\textsc{RepExt}(\mathsf{INT}, \textsc{Fixed})$ *and* $\textsc{RepExt}(\mathsf{INT}, \textsc{Add})$ *are* $\mathsf{W[1]}$*-hard when parameterized by the number* k *of pre-drawn intervals.*

Proof We modify the reductions of Theorem 3 and Corollary 1 exactly as in the proof of Theorem 1. □

By the Size of the Path. We show that the $\textsc{Fixed}$ type problems are fixed-parameter tractable with respect to the size of the path t. It is easy to find a solution by a brute-force algorithm:

Proposition 5 *For the size* t *of a path* T'*, the problems* $\textsc{RepExt}(\mathsf{PINT}, \textsc{Fixed})$ *and* $\textsc{RepExt}(\mathsf{INT}, \textsc{Fixed})$ *are fixed-parameter tractable with the respect to the parameter* t*. They can be solved in time* $\mathcal{O}(n + m + f(t))$ *where*

$$f(t) = t^{2t^2}.$$

Proof In a pruned graph, the vertices have to be represented by pairwise different intervals. There are at most t^2 possible different subpaths of a path with t vertices so the pruned graph can contain at most t^2 vertices; otherwise the extension is clearly not possible. We can test every possible assignment of the non-pre-drawn vertices to the t^2 subpaths, and for each assignment we test whether we get a correct representation extending $\mathcal{R}'$. □

4 Path and Chordal Graphs

We present and prove the results concerning the classes PATH and CHOR.

4.1 The Polynomial Cases

The recognition problems for the types $\textsc{Add}$ and $\textsc{Both}$ are equivalent to standard recognition without any specified tree T'. Indeed, we can modify T' by adding an arbitrary tree to it. If the input graph is PATH or CHOR, there exists a tree T'' in which the graph can be represented. We produce T by attaching T'' to T' in any way. Then the input graph can be represented in T as well, completely ignoring the part T'.

For path graphs, the original recognition algorithm is due to Gavril [11] in time $\mathcal{O}(n^4)$. The current fastest algorithm is by Schäffer [24] in time $\mathcal{O}(nm)$. For chordal graphs, there is a beautiful simple algorithm by Rose et al. [23] in time $\mathcal{O}(n + m)$.

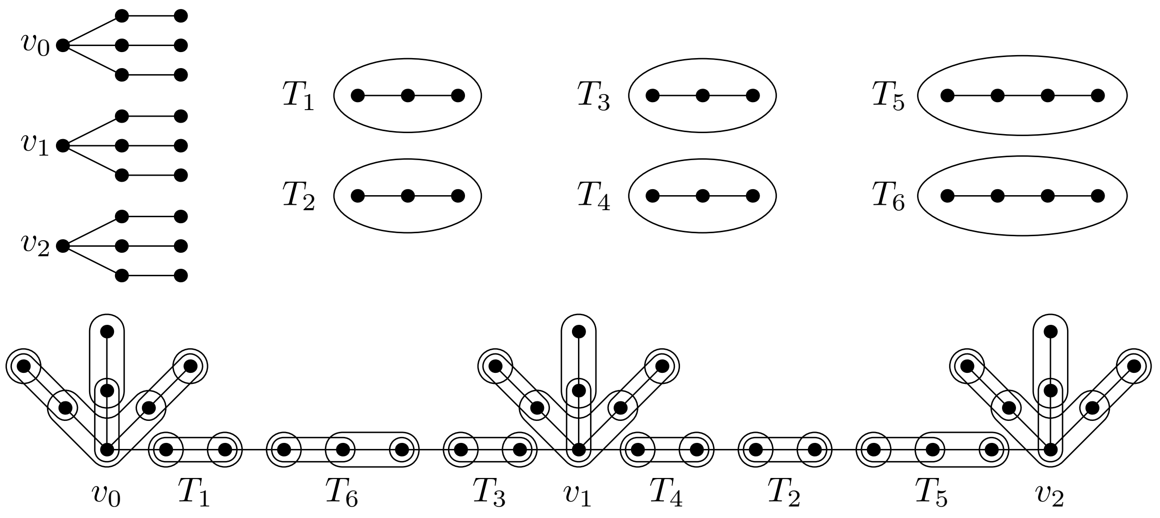

Fig. 8 An example for the same input of 3-Partition as in Fig. 7. On top the graph G is depicted. On bottom, a representation of G is constructed, giving the solution $\{A_1, A_3, A_6\}$ and $\{A_2, A_4, A_5\}$.

4.2 The NP-complete Cases

All the remaining cases from the table of Fig. 3 are NP-complete. We modify the reduction for INT of Theorem 3. We start with the simplest reduction for the Fixed type and then modify it for the other types.

Fixed Type Recognition. For the Fixed type, we can avoid pre-drawn subtrees, using an additional structure of the tree.

Proposition 6 *The problems* Recog*(PATH, Fixed) *and* Recog*(CHOR, Fixed) *are* NP*-complete.*

Proof We again reduce from 3-Partition with an input k and M. For technical purposes, let $M \geq 8$ and so $|A_i| > 2$ for each A_i. We construct a graph G and a tree T' as follows.

The tree T' is a path $P_{(M+1)k}$ (its vertices being denoted by $p_0, \dots, p_{(M+1)k}$) with three paths of length two attached to every vertex $p_{(M+1)i}$, for each $i = 0, \dots, k$; see Fig. 8. Each split gadget S_i is a star, depicted on the left of Fig. 8. When the split gadgets are placed as in T', they split the tree into k gaps exactly as the pre-drawn vertices in the proof of Theorem 3. Each take gadget T_i is the path P_{A_i} exactly as before. The reduction is obviously polynomial.

What remains to argue is the correctness of the reduction. Observe that given a solution of 3-Partition, we can construct a subpaths-in-tree representation of G as in Fig. 8. For the other direction, let $v_0, \dots, v_k$ be the central vertices of the split gadgets $S_0, \dots, S_k$. We claim that each R_{v_i} contains at least one branch vertex. (Actually, exactly one since there are only $n + 1$ branch vertices in T'.) If some R_{v_i} contained only non-branch vertices, then it would not be possible to represent three disjoint neighbors u_1, u_2 and u_3 of this v_i having each $R_{u_j} \setminus R_{v_i}$ non-empty.

Since each branch vertex is taken by one R_{v_i}, the path $P_{(M+1)k}$ is split into k gaps as before. Since $|A_i| > 2$, each T_i can be represented only inside of these gaps. Notice that the total number of the vertices in the gaps has to be equal kM, and therefore the split gadgets have to be represented entirely in the attached stars as in Fig. 8. The rest of the reduction works exactly as in Theorem 3. □

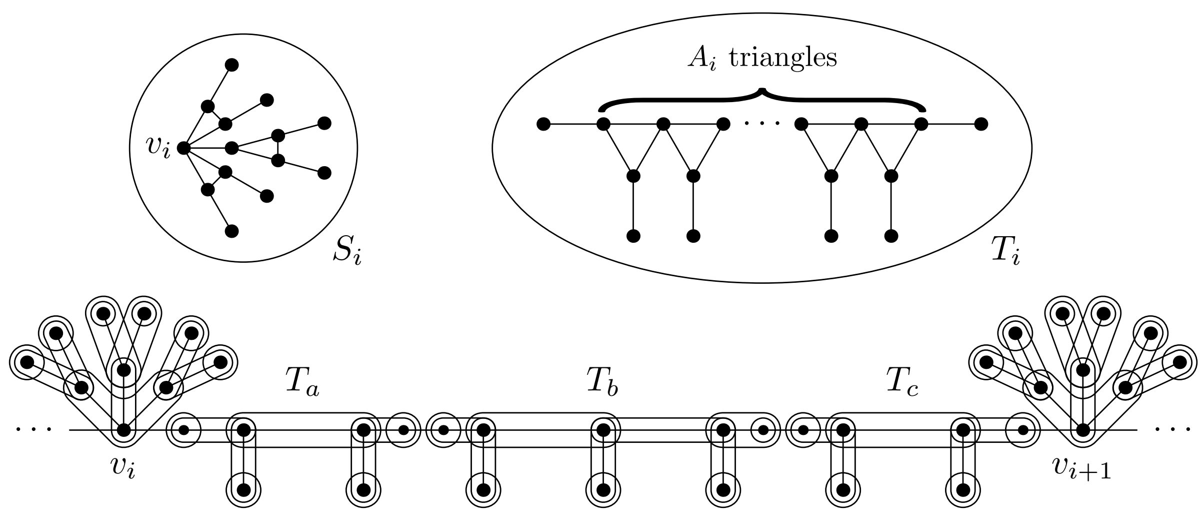

Fig. 9 On top, the split gadget S_i is on the left and the take gadget T_i is on the right. On bottom, a part of the tree T is depicted with the small vertices added by subdivisions. The gap between two split gadgets contains three take gadgets T_a, T_b and T_c giving one triple $\{A_a, A_b, A_c\}$ with $A_a + A_b + A_c = M$.

Sub Type Recognition. By modifying the above reduction, we get:

Theorem 4 *The problems* RECOG*(PATH, SUB) *and* RECOG*(CHOR, SUB) *are* NP-*complete.*

Proof We need to modify the two gadgets from the reduction of Theorem 6 in such a way that a subdivision of the tree T' does not help in placing them. Subdivision only increases the number of non-branch vertices. Thus a take gadget T_i requires A_i branch vertices. Similarly, the split gadget S_i is more complicated. See Fig. 9 on top.

The tree T is constructed as follows. We start with a path $P_{(M+1)k}$ with vertices $p_0, \dots, p_{(M+1)k}$. To each vertex $p_{(M+1)i}$ we attach a subtree isomorphic to the trees in Fig. 9 on bottom. To the remaining vertices of the path, we attach one leaf per vertex. The reduction is again polynomial.

Straightforwardly, for a given solution of 3-PARTITION, we can construct a correct subpaths-in-tree representation in a subdivided tree. On the other hand, we are going to show how to construct a solution of 3-PARTITION from a given tree representation.

Recall the properties of maximal cliques from Section 2. Note that each triangle $u_1u_2u_3$ in each split or take gadget is a maximal clique K. Since $N[u_i] \neq N[u_j]$ for each $i \neq j$, there has to be a branch vertex in $R_{u_i} \cap R_{u_j}$ for some i and j. The gadget S_i contains three triangles, each taking one branch vertex of T. In addition, R_{v_i} connecting them has to contain another branch vertex. So in total, S_i contains at least four branch vertices. Each gadget T_i contains A_i triangles, and so it requires at least A_i branch vertices. Since the number of branch vertices of T is limited, each S_i takes exactly four branch vertices and each T_i takes exactly A_i branch vertices.

Now, if some T_i contained a branch vertex of the subtrees attached to $p_{(M+1)j}$, at least one of its branch vertices would not be used. (Either not taken by T_i, or T_i would require at least $A_i + 1$ branch vertices.) So each S_i has to take the branch vertices of the subtrees attached to $p_{(M+1)j}$ for some j, and the take gadgets have to be placed inside the gaps exactly as before. $\square$

Proposition 7 *Even with a single pre-drawn subtree, i.e,* $|G'| = 1$*, the problems* $\textsc{RepExt}(\mathsf{CHOR}, \textsc{Add})$ *and* $\textsc{RepExt}(\mathsf{CHOR}, \textsc{Both})$ *are* NP*-complete.*

Proof We easily modify the above reductions; for the Add type, the reduction of Proposition 6, for the Both type, the reduction of Theorem 4. The modification adds into G one pre-drawn vertex v adjacent to everything such that $R_v = T'$. Since R_v spans the whole tree, it forces the entire representation $\mathcal{R}$ into T'.

We just deal with the Add type, for Both the argument is exactly the same. Let T' be the partial tree and let T be the tree in which the representation is constructed, so T' is a subtree of T. We claim that we can restrict a representation of each vertex of G into T' and thus obtain a correct representation inside the subtree T'.

Let $x \in V$. Since $xv \in E(G)$, the intersection of R_x and T' is a non-empty subtree. We put $\widetilde{R}_x = R_x \cap T'$, and we claim that $\widetilde{\mathcal{R}}$ is a representation of G in T'. To argue the correctness, let x and y be two different vertices from v (otherwise trivial). If $xy \notin E(G)$, then $R_x \cap R_y = \emptyset$, and so $\widetilde{R}_x \cap \widetilde{R}_y = \emptyset$ as well. Otherwise xyv is a triangle in G, and thus by the Helly property the subtrees R_x, R_y and $R_v = T'$ have a non-empty common intersection, giving that $\widetilde{R}_x \cap \widetilde{R}_y$ is non-empty. □

For path graphs, one can use a similar technique of a pre-drawn universal vertex attached to everything. But there is the following difficulty: To do so, the input partial tree T' has to be a path. For the type Both, the complexity of $\textsc{RepExt}(\mathsf{PATH}, \textsc{Both})$ remains open. For the type Add, we get the following weaker result:

Proposition 8 *The problem* $\textsc{RepExt}(\mathsf{PATH}, \textsc{Add})$ *is* NP*-complete.*

Proof Similarly as in Proposition 7, add a pre-drawn universal vertex v on the path T' constructed in the reduction of Theorem 3 such that $R_v = T'$. The rest is exactly as above. □

4.3 The Parameterized Complexity

We deal with parameterized complexity of the problems and we give only minor and partial results in this direction. Unlike in Section 3, parameterization by the number k of pre-drawn subtrees is mostly not helpful. We show that every problem with exception of $\textsc{RepExt}(\mathsf{PATH}, \textsc{Add})$ is already NP-complete for $k = 0$ or $k = 1$. For $\textsc{RepExt}(\mathsf{PATH}, \textsc{Add})$, we have only a weaker result that it is W[1]-hard with respect to the parameter k since Proposition 4 straightforwardly generalizes.

Similarly, a low number c of components does not make the problem any easier. We can easily insert a universal vertex attached to everything. So the above reductions can be modified and the problems remain NP-complete even if the graph G is connected.

Concerning the size t of the tree, Proposition 5 straightforwardly generalizes:

Proposition 9 *Let t be the size of T'. The problems* $\textsc{RepExt}(\mathsf{PATH}, \textsc{Fixed})$ *and* $\textsc{RepExt}(\mathsf{CHOR}, \textsc{Fixed})$ *are fixed-parameter tractable with respect to t. They can be solved in time $\mathcal{O}(n + m + g(t))$ where*

$$g(t) = 2^{t2^t}.$$

Proof Proceed exactly as in the proof of Proposition 5, test all possible assignments of all vertices of a pruned graph. The only difference is that T' has at most 2^t different subtrees. □

We note that a more precise bound for the number of subtrees could be use but we did not try to better estimate the function g.

5 Conclusions

In this paper, we have considered different problems concerning extending partial representations of chordal graphs and their three subclasses. One of the main goals of this paper is to stimulate a future research in this area. Therefore, we conclude with three open problems.

The first problem concerns the only open case in the table in Fig. 3.

Problem 1 What is the complexity of RepExt(PATH, Both)?

Concerning the parameterized complexity, we believe it is useful to first attack problems related to interval graphs. This allows to develop tools for more complicated chordal graphs. A generalization of Theorem 1 and Corollary 2 for INT seems to be particularly interesting. The PQ-tree approach seems to be a good starting point.

Problem 2 Does RepExt(INT, Fixed) belong to XP with respect to k where k is the number of pre-drawn intervals?

We present only basic results concerning the parameterized complexity with respect to the parameter t where t is the size of the tree T'. We deal with the type Fixed for which the solution is straightforward. The complexity for the other types Sub, Add, and Both remains open.

Problem 3 What is the parameterized complexity of the remaining problems with respect to the parameter t where t is the size of the tree T'?

Acknowledgements The first two authors are supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023. The first author is also supported by Charles University as GAUK 196213.

References

1. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. In: SODA '13 (2013)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. Journal of Computational Systems Science **13**, 335–379 (1976)
3. Chaplick, S., Fulek, R., Klavík, P.: Extending partial representations of circle graphs. In preparation. (2013)
4. Corneil, D.G., Kim, H., Natarajan, S., Olariu, S., Sprague, A.P.: Simple linear time recognition of unit interval graphs. Information Processing Letters **55**(2), 99–104 (1995)
5. Corneil, D.G., Olariu, S., Stewart, L.: The LBFS structure and recognition of interval graphs. SIAM Journal on Discrete Mathematics **23**(4), 1905–1953 (2009)

6. Deng, X., Hell, P., Huang, J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. SIAM J. Comput. **25**(2), 390–403 (1996)
7. Fiala, J.: NP completeness of the edge precoloring extension problem on bipartite graphs. J. Graph Theory **43**(2), 156–160 (2003)
8. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**, 835–855 (1965)
9. Garey, M.R., Johnson, D.S.: Complexity results for multiprocessor scheduling under resource constraints. SIAM Journal on Computing **4**(4), 397–411 (1975)
10. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. Journal of Combinatorial Theory, Series B **16**(1), 47–56 (1974)
11. Gavril, F.: A recognition algorithm for the intersection of graphs of paths in trees. Discrete Mathematics **23**, 211–227 (1978)
12. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. North-Holland Publishing Co. (2004)
13. Hujter, M., Tuza, Z.: Precoloring extension. II. Graph classes related to bipartite graphs. Acta Mathematica Universitatis Comenianae **62**(1), 1–11 (1993)
14. Jampani, K., Lubiw, A.: The simultaneous representation problem for chordal, comparability and permutation graphs. In: Algorithms and Data Structures, *Lecture Notes in Computer Science*, vol. 5664, pp. 387–398 (2009)
15. Jansen, K., Kratsch, S., Marx, D., Schlotter, I.: Bin packing with fixed number of bins revisited. In: Algorithm Theory - SWAT 2010, *Lecture Notes in Computer Science*, vol. 6139, pp. 260–272 (2010)
16. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Algorithms – ESA 2012, *Lecture Notes in Computer Science*, vol. 7501, pp. 671–682 (2012)
17. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. In preparation. (2012)
18. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T.: Extending partial representations of subclasses of chordal graphs. In: Algorithms and Computation – ISAAC, *Lecture Notes in Computer Science*, vol. 7676, pp. 444–454 (2012)
19. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T., Vyskočil, T.: Linear-time algorithm for partial representation extension of interval graphs. Journal version of the TAMC paper, in preparation. (2012)
20. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: Theory and Applications of Models of Computation - 8th Annual Conference, TAMC 2011, *Lecture Notes in Computer Science*, vol. 6648, pp. 276–285 (2011)
21. Looges, P.J., Olariu, S.: Optimal greedy algorithms for indifference graphs. Comput. Math. Appl. **25**, 15–25 (1993)
22. McKee, T.A., McMorris, F.R.: Topics in Intersection Graph Theory. SIAM Monographs on Discrete Mathematics and Applications (1999)
23. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM Journal on Computing **5**(2), 266–283 (1976)
24. Schäffer, A.A.: A faster algorithm to recognize undirected path graphs. Discrete Appl. Math **43**, 261–295 (1993)
25. Spinrad, J.P.: Efficient Graph Representations. Field Institute Monographs (2003)