On the Classes of Interval Graphs of Limited Nesting and Count of Lengths*†

Pavel Klavík[1], Yota Otachi[2], and Jiří Šejnoha[3]

1 **Computer Science Institute, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic**
klavik@iuuk.mff.cuni.cz

2 **School of Information Science, Japan Advanced Institute of Science and Technology, Japan**
otachi@jaist.ac.jp

3 **Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic**
jirka.sejnoha@gmail.com

Abstract

In 1969, Roberts introduced *proper* and *unit interval graphs* and proved that these classes are equal. Natural generalizations of unit interval graphs called *k-length interval graphs* were considered in which the number of different lengths of intervals is limited by k. Even after decades of research, no insight into their structure is known and the complexity of recognition is open even for $k = 2$. We propose generalizations of proper interval graphs called *k-nested interval graphs* in which there are no chains of $k + 1$ intervals nested in each other. It is easy to see that k-nested interval graphs are a superclass of k-length interval graphs.

We give a linear-time recognition algorithm for k-nested interval graphs. This algorithm adds a missing piece to Gajarský et al. [FOCS 2015] to show that testing FO properties on interval graphs is FPT with respect to the nesting k and the length of the formula, while the problem is W[2]-hard when parameterized just by the length of the formula. Further, we show that a generalization of recognition called *partial representation extension* is polynomial-time solvable for k-nested interval graphs, while it is NP-hard for k-length interval graphs, even when $k = 2$.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases interval graphs, proper and unit interval graphs, recognition, partial representation extension

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.45

1 Introduction

An *interval representation* $\mathcal{R}$ of a graph G is a collection $\{\langle u\rangle : u \in V(G)\}$ of intervals of the real line such that $uv \in E(G)$ if and only if $\langle u\rangle \cap \langle v\rangle \neq \emptyset$. A graph is an *interval graph* if it has an interval representation, and we denote this class by INT.

An interval representation is called *proper* if $\langle u\rangle \subseteq \langle v\rangle$ implies $\langle u\rangle = \langle v\rangle$, and *unit* if the length of all intervals $\langle u\rangle$ is one. The classes of *proper* and *unit interval graphs* (denoted PROPER INT and UNIT INT) consist of all interval graphs which have proper and unit interval representations, respectively. Roberts [27] proved that PROPER INT = UNIT INT.

* We omit many details and proofs, for the full version, see [20].

† The first author is supported by CE-ITI (GAČR P202/12/G061) and Charles University as GAUK 196213.

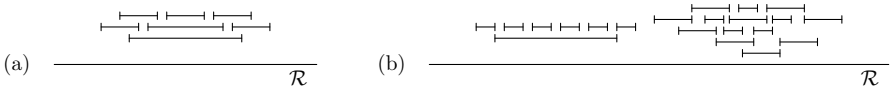

Figure 1 (a) An interval representation with the nesting three. (b) The disjoint union of two components with the minimum nesting two requiring three different lengths of intervals. On the left, the shorter intervals are shorter than $\frac{1}{4}$ of the longer ones. On the right, they are longer than $\frac{1}{3}$.

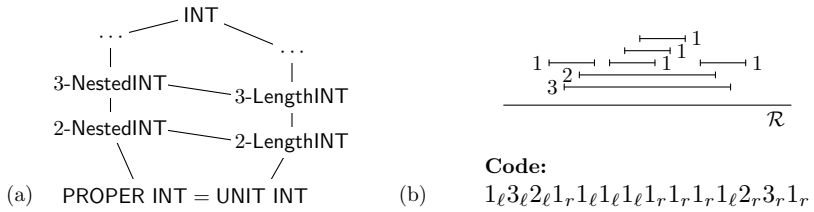

Figure 2 (a) The Hasse diagram of proper inclusions of the considered classes. (b) We can label each interval by the length of a maximal chain of nested intervals ending in it. We code the graph by the left-to-right sequence of left endpoints ℓ and right endpoints r together with their labels.

The Studied Classes. In this paper, we consider two hierarchies of subclasses of interval graphs which generalize proper and unit interval graphs. The class k-NestedINT consists of all interval graphs which have representations with no $k+1$ intervals $\langle u_0\rangle, \ldots, \langle u_k\rangle$ such that $\langle u_0\rangle \subsetneq \langle u_1\rangle \subsetneq \cdots \subsetneq \langle u_k\rangle$; see Fig. 1a. The class k-LengthINT consists of all interval graphs which have representations having at most k different lengths of intervals; see Fig. 1b. We know by [27] that 1-NestedINT = PROPER INT = UNIT INT = 1-LengthINT.

For an interval graph G, we denote the minimum nesting over all interval representations by $\nu(G)$, and the minimum number of interval lengths by $\lambda(G)$. Since nested intervals have different lengths, we know that $\nu(G) \leq \lambda(G)$ and this inequality may be strict (as in Fig. 1b). For each $k \geq 2$, $(k-1)$-LengthINT $\subsetneq$ k-LengthINT $\subsetneq$ k-NestedINT $\subsetneq$ $(k+1)$-NestedINT. Fishburn [10, Theorem 5, p. 177] shows that graphs G in 2-NestedINT have unbounded $\lambda(G)$, so 2-NestedINT $\not\subseteq$ k-LengthINT for each k. See Figure 2a.

Partial Representation Extension. This problem was introduced by Klavík et al. [19]. A *partial representation* $\mathcal{R}'$ of G is an interval representation $\{\langle x\rangle' : x \in V(G')\}$ of an induced subgraph G' of G. The vertices of G' and the intervals of $\mathcal{R}'$ are called *pre-drawn*. A representation $\mathcal{R}$ of G *extends* $\mathcal{R}'$ if and only if it assigns the same intervals to the vertices of G': $\langle x\rangle = \langle x\rangle'$ for every $x \in V(G')$.

Problem:	Partial Representation Extension – RepExt($\mathcal{C}$)
Input:	A graph G and a partial representation $\mathcal{R}'$ of an induced subgraph G'.
Output:	Is there an interval representation of $\mathcal{C}$ of G extending $\mathcal{R}'$?

An $\mathcal{O}(n^2)$-time algorithm for RepExt(INT) was given in [19]. There are two different linear-time algorithms [3, 18] for this problem. Minimal obstructions making partial representations non-extendible are described in [21]. A linear-time algorithm for proper interval graphs [16] and a quadratic-time algorithm for unit interval graphs [30] are known.

The partial representation extension problems were considered for other graph classes. Polynomial-time algorithms are known for circle graphs [7], and permutation and function graphs [15]. The problems are NP-hard for chordal graphs [17] and contact representations of planar graphs [6]. The complexity is open for circular-arc and trapezoid graphs.

Previous Results and Motivation. The classes k-LengthINT were introduced by Graham as a natural hierarchy between unit interval graphs and interval graphs; see Fig. 2a. Unfortunately, even after decades, the only results known are curiosities that illustrate the incredibly complex structure of k-LengthINT, very different from the case of unit interval graphs. For instance, k-LengthINT is not closed under disjoint unions; see Fig. 1b.

Leibowitz et al. [23] show that the class 2-LengthINT contains caterpillars, threshold, and unit interval graphs with one additional vertex. Further, there exist graphs G with $\lambda(G) > 2$ such that $\lambda(G \setminus x) \leq \lambda(G) - 2$ for $x \in V(G)$ [23]. Fishburn [9] shows that there are infinitely many forbidden induced subgraphs for 2-LengthINT (where UNIT INT are interval graphs just without $K_{1,3}$). It is also known [8] that there are graphs in 2-LengthINT such that, when the shorter length is fixed to 1, the longer one can be one of the real numbers belonging to arbitrary many distinct intervals of the real line, arbitrary far from each other.

Not much is known about the computational complexity of problems involving k-LengthINT, even recognition is open for $k = 2$. In [5], a polynomial-time algorithm is given for computing $\lambda(G)$ for interval graphs G which are extended bull-free or almost threshold (which highly restricts them). Skrien [29] characterized 2-LengthINT which can be realized by lengths zero (points) and one (unit intervals), leading to a linear-time recognition algorithm. As most of the efficient algorithms for intersection graph classes require representations, very little is known how to algorithmically use that a given interval graph can be represented by k lengths. In this paper, we show that partial representation extension is NP-hard already for 2-LengthINT.

All these difficulties lead us to introduce another hierarchy of k-NestedINT which generalizes proper interval graphs; see Fig. 2. We illustrate the nice structure of k-NestedINT by describing a relatively simple linear-time recognition algorithm which we also generalize to a more involved polynomial-time algorithm for partial representation extension. To the best of our knowledge, the only reference is Fishburn's book [10] in which the parameter $\nu(G)$ called *depth* is considered and linked to k-LengthINT. There are some different generalizations of proper interval graphs [26], which are less rich and not linked to k-LengthINT.

Since k-NestedINT seem to share many properties with proper interval graphs, several future directions of research are immediately offered. It should be possible to describe minimal forbidden induced subgraphs. For the computational problems which are tractable for proper interval graphs and hard for interval graphs, the complexity of the intermediate problems for k-NestedINT can be studied. (One such problem is FO property checking, discussed below.)

Our Results. In [14], a polynomial-time algorithm is given for recognizing 2-LengthINT when intervals are partitioned into two subsets A and B, each of one length, and both $G[A]$ and $G[B]$ are connected. This approach might be generalized for partial representation extension, but we show that removing the connectedness condition makes it hard:

▶ **Theorem 1.** *The problem* RepExt(2-LengthINT) *is* NP*-hard when every pre-drawn interval is of one length a. It remains* NP*-hard even when* (i) *the input prescribes two lengths $a = 1$ and b, and* (ii) *for every interval, the input assigns one of the lengths a or b. Also, it is* W[1]*-hard when parameterized by the number of pre-drawn intervals.*

In comparison, we give a dynamic programming algorithm for recognizing k-NestedINT, based on a data structure called an MPQ-tree. We show that we can optimize nesting greedily from the bottom to the top. We compute three different representations for each subtree and we show how to combine them.

▶ **Theorem 2.** *The minimum nesting number $\nu(G)$ can be computed in time $\mathcal{O}(n+m)$ where n is the number of vertices and m is the number of edges. Therefore, the problem* Recog(k-NestedINT) *can be solved in linear time.*

This result has the following application in the computational complexity of deciding logic formulas over graphs. Let φ be the length of a first-order logic formula for graphs. By the locality, this formula can be decided in G in time $n^{\mathcal{O}(\varphi)}$. Since it is W[2]-hard to decide it for general graphs when parameterized by φ, it is natural to ask for which graph classes there exists an FPT algorithm running in time $\mathcal{O}(n^c \cdot f(\varphi))$.

In [13], it is shown that the problem above is W[2]-hard even for interval graphs. On the other hand, if an interval graph is given together with a k-length interval representation, [13] gives an FPT algorithm with respect to the parameters φ and the particular lengths of the intervals. It was not clear whether such an algorithm is inherently geometrical. Recently, Gajarský et al. [12] give a different FPT algorithm for FO property testing for interval graphs parameterized by φ and the nesting k, assuming that a k-nested interval representation is given by the input. By our result, this assumption can be removed since we can compute an interval representation of the optimal nesting in linear time.

The problem RepExt(k-NestedINT) is more involved since a straightforward greedy optimization from the bottom to the top does not work. We describe a complex dynamic programming algorithm which computes with exponentially many possibilities efficiently.

▶ **Theorem 3.** *The problem* RepExt(k-NestedINT) *can be solved in polynomial time.*

The last result contrasts with Theorem 1. Partial representation extension of k-NestedINT and k-LengthINT are problems for which the geometrical version (at most k lengths) is much harder than the corresponding topological problem (the left-to-right ordering of endpoints of intervals). A generalization of partial representation extension called bounded representations is NP-complete for unit interval graphs [16], but polynomial-time solvable for proper interval graphs [2]. Partially embedded planar graphs can be extended in linear time [1], but extension of geometric straight-line embeddings is NP-hard [25].

2 Extending Partial Representations with Two Lengths

The complexity of recognizing k-LengthINT is a long-standing open problem, even for $k = 2$. In this section, we show that RepExt is NP-complete even when $k = 2$.

Proof of Theorem 1. Assume (i) and (ii). We adapt the reduction from 3-Partition used in [17, 16]. Let $A_1, \ldots, A_{3s}$ and M be input of 3-Partition, with $\frac{M}{2} < A_i < \frac{M}{4}$ and $\sum A_i = Ms$. The task is to split A_i's into s triples, each summing to exactly M.

We solve this problem by constructing an interval graph G and a partial representation $\mathcal{R}'$ as depicted in Fig. 3. We claim that $\mathcal{R}'$ can be extended using two lengths of intervals if and only if the original instance of 3-Partition is solvable. We set $a = 1$ and $b = s \cdot (M+1)$. The partial representation $\mathcal{R}'$ consists of $s+1$ disjoint pre-drawn intervals $v_0, \ldots, v_s$ of length a, with their pre-drawn positions $\ell'(v_i) = i \cdot (M+2)$. So they split the real line into s equal gaps of size $M+1$ and two infinite regions.

Aside $v_0, \ldots, v_s$, the graph G contains exactly one vertex w of the length b, adjacent to every vertex in G. Further, for each A_i, the graph $G \setminus w$ contains P_{2A_i} (a path with $2A_i$ vertices) as one component, with each vertex of the length a.

This reduction is clearly polynomial. It remains to show that $\mathcal{R}'$ is extendible if and only if the instance of 3-Partition is solvable. First, because of the length constraint, in

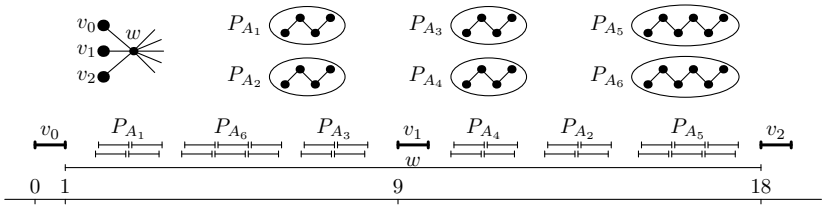

Figure 3 We consider the following input for 3-Partition: $s = 2$, $M = 7$, $A_1 = A_2 = A_3 = A_4 = 2$ and $A_5 = A_6 = 3$. The associated graph G is depicted on top, and at the bottom we find one of its extending representations, giving the 3-partitioning $\{A_1, A_3, A_6\}$ and $\{A_2, A_4, A_5\}$.

every extending representation has $\ell(w) = 1$; otherwise it would not intersect either v_0, or v_s. Therefore, each of the paths P_{2A_i} has to be placed in exactly one of the s gaps. In every representation of P_{2A_i}, it requires the space at least $A_i + \varepsilon$ for some $\varepsilon > 0$. Therefore, three paths can be packed into the same gap if and only if their three integers sum to at most M. Therefore, an extending representation $\mathcal{R}'$ gives a solution to 3-Partition, and vice versa. We get W[1]-hardness by a similar reduction from BinPacking, as in [17].

The above reduction can be easily modified when (i) and (ii) are avoided. We add two extra vertices: w_0 adjacent to v_0 and w_s adjacent to v_s such that both are non-adjacent to w. It forces the length of w to be in the interval $[s \cdot (M+1), s \cdot (M+1) + 2)$, so the length b does not have to be prescribed. Also, this reduction works even when each interval does not have a length assigned, aside the pre-drawn intervals in $\mathcal{R}'$ indeed. ◀

3 Basic Properties of k-Nested Interval Graphs

The nesting defines a partial ordering $\subsetneq$ of intervals and we denote by $\nu(u)$ the length of the longest chain of nested intervals ending with $\langle u \rangle$. By $\mathrm{Pred}(u)$, we denote the set of all direct predecessors of $\langle u \rangle$ in $\subsetneq$.

▶ **Lemma 4.** *An interval graph belongs to k-NestedINT if and only if it has an interval representation which can be partitioned into k proper interval representations.*

Proof. Let $\mathcal{R}$ be an interval representation partitioned into proper interval representations $\mathcal{R}_1, \ldots, \mathcal{R}_k$. No chain of nested intervals contains two intervals from some $\mathcal{R}_i$, so the nesting is at most k. On the other hand, let $\mathcal{R}$ be a k-nested interval representation. We label each interval by the length of the longest chain of nested intervals ending in it; see Fig. 2b. Notice that the intervals of each label $i \in \{1, \ldots, k\}$ form a proper interval representation $\mathcal{R}_i$. ◀

Interval graphs and the subclasses k-NestedINT and k-LengthINT are closed under induced subgraphs, so they can be characterized by minimal forbidden induced subgraphs. Lekkerkerker and Boland [24] describe them for interval graphs, and Roberts [27] proved that 1-NestedINT=1-LengthINT are claw-free interval graphs. On the other hand, 2-LengthINT have infinitely many minimal forbidden induced subgraphs [9] which are interval graphs.

4 Maximal Cliques and MPQ-trees

▶ **Lemma 5** (Fulkerson and Gross [11]). *A graph is an interval graph if and only if there exists a linear ordering $<$ of its maximal cliques such that, for each vertex, the maximal cliques containing this vertex appear consecutively.*

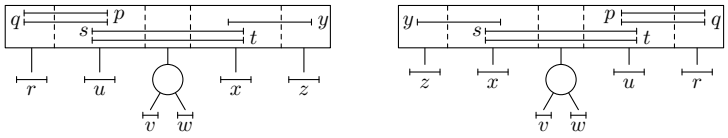

Figure 4 Two equivalent MPQ-trees with denoted sections. In all figures, we denote P-nodes by circles and Q-nodes by rectangles.

An ordering of the maximal cliques satisfying the statement of Lemma 5 is called a *consecutive ordering*. Each interval graph has $\mathcal{O}(n)$ maximal cliques of total size $\mathcal{O}(n+m)$ which can be found in linear time [28].

PQ-trees. A *PQ-tree* T is a rooted tree, introduced by Booth and Lueker [4]. Its leaves are in one-to-one correspondence with the maximal cliques. Its inner nodes are of two types: *P-nodes* and *Q-nodes*. Each P-node has at least two children, each Q-node at least three. The orderings of the children of inner nodes are given. The PQ-tree T represents one consecutive ordering $<_T$ called the *frontier* of T which is the ordering of the leaves from left to right.

The PQ-tree T represents all consecutive orderings of G as frontiers of equivalent PQ-trees which can be constructed from T by sequences of *equivalent transformations* of two types: (i) an arbitrary reordering of the children of a P-node, and (ii) a reversal of the order of the children of a Q-node; see Fig. 4. A subtree of T consists of a node and all its descendants. For a node N, its subtrees are the subtrees which have the children of N as the roots.

MPQ-trees. The *MPQ-tree* [22] is an augmentation of the PQ-tree in which the nodes of T have assigned subsets of $V(G)$ called *sections*. To a leaf representing a clique C, we assign one section $s(C)$. Similarly, to each P-node P, we assign one section $s(P)$. For a Q-node Q with subtrees $T_1, \ldots, T_q$, we have q sections $s_1(Q), \ldots, s_q(Q)$ ordered from left to right, each corresponding to one subtree. Examples of sections are depicted in Fig. 4.

The section $s(C)$ has all vertices contained in the maximal clique C and no other maximal clique. The section $s(P)$ of a P-node P has all vertices that are contained in all maximal cliques of the subtree rooted at P and in no other maximal clique. Let Q be a Q-node with subtrees $T_1, \ldots, T_q$. Let x be a vertex contained only in maximal cliques of the subtree rooted at Q, contained in maximal cliques of at least two subtrees. Then x is contained in every section $s_i(Q)$ such that some maximal clique of T_i contains x.

Every vertex x is in sections of exactly one node of T. In the case of a Q-node, it is placed in consecutive sections of this node. For a Q-node Q, if x is placed in a section $s_i(Q)$, then x is contained in all cliques of T_i. Every section of a Q-node is non-empty, and two consecutive sections have a non-empty intersection.

5 Recognizing k-nested Interval Graphs

Our linear-time algorithm is a dynamic programming on the MPQ-tree. We process from the bottom to the top, and we optimize the minimal nesting.

Two vertices x and y are twins if and only if $N[x] = N[y]$. The standard observation is that twins can be ignored since they can be represented by identical intervals, and notice that this does not increase nesting and the number of lengths. We can locate all twins in time $\mathcal{O}(n+m)$ [28] and we can prune the graph by keeping one vertex per equivalence class of twins. So no two vertices belong to the exactly same sections of the MPQ-tree.

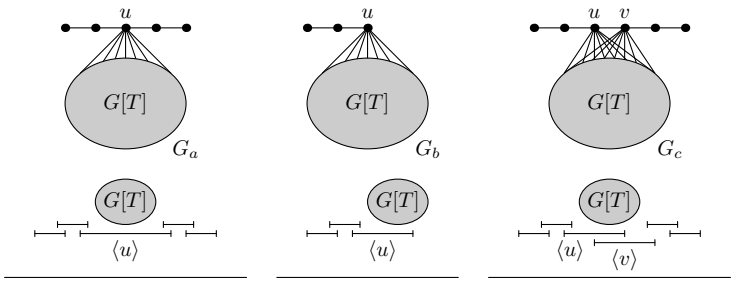

Figure 5 The graphs G_a, G_b and G_c with representations, defining the triple (a, b, c) of T. The vertices of $G[T]$ are adjacent to the added vertices u and v and not to the others.

Triples. We compute triples (a, b, c) for each subtree of the MPQ-tree: a is the optimal nesting of the entire subtree, b is the nesting when the subtree is placed on a side of its parent, and c is used for Q-nodes. We define for a subtree T a triple (a, b, c) as follows. Let $G[T]$ be the interval graph induced by the vertices of the sections of T. Let G_a, G_b and G_c be graphs as in Fig. 5. We define $a = \nu(G_a) - 1 = \nu(G[T])$, $b = \nu(G_b) - 1$, and $c = \nu(G_c) - 1$.

The triple can be interpreted as increase in the nesting, depending how $G[T]$ is represented with respect to the rest of the graph. We compute these triples from the leaves to the root, and output a of the root as $\nu(G)$.

▶ **Lemma 6.** *For any subtree T, its triple (a, b, c) satisfies $a - 1 \leq b \leq c \leq a$.*

Proof. We prove equivalently that $\nu(G_a) - 1 \leq \nu(G_b) \leq \nu(G_c) \leq \nu(G_a)$. We trivially know that $\nu(G_b) \leq \nu(G_c)$ since G_b is an induced subgraph of G_c.

Our definition of G_a implies that $\nu(G_a) = \nu(G[T])+1$, since in every interval representation of G_a, both endpoints of $\langle u \rangle$ are covered by attached paths, and there the representation of $G[T]$ is nested in $\langle u \rangle$. Since $\nu(G_b) \geq \nu(G[T])$, the inequality $\nu(G_a) - 1 \leq \nu(G_b)$ follows. Alternatively, for a representation of G_b optimizing nesting, we stretch $\langle u \rangle$ (which increases nesting by at most one) and add the second attached path, to get a representation of G_a.

It remains to show the last inequality that $\nu(G_c) \leq \nu(G_a)$. Consider the representation of G_a with optimal nesting, in which $G[T]$ is strictly contained inside $\langle u \rangle$. By shifting $r(u)$ to the left and adding $\langle v \rangle$, we get a representation of G_c with the same nesting. ◀

Leaves. We initiate (a, b, c) for every leaf L of the MPQ-tree as follows. If $s(L) = \emptyset$, we put $(0, 0, 0)$; otherwise we put $(1, 0, 0)$. The reason is that the vertices of $s(L)$ form a clique.

P-nodes. Let $T_1, \ldots, T_p$ be the children of a P-node P, with $p \geq 2$, with the computed triple (a_i, b_i, c_i) for each subtree T_i. We want to compute (a, b, c) for P. We first assume that $s(P) \neq \emptyset$. Let $w \in s(P)$, then the vertices of every subtree except two outer subtrees T_s and T_t are nested inside $\langle w \rangle$, so their minimal nesting number is increased by one. We can choose optimally two outer subtrees T_s and T_t, and their outer maximal cliques. Only the intervals contained in these two outer maximal cliques are not nested inside $\langle w \rangle$.

We get the following formulas (see Fig. 6):

$$\begin{aligned} a &= \begin{cases} \max\{a_1, \ldots, a_p\}, & s(P) = \emptyset, \\ \min_{s \neq t} \max\{b_s, b_t, a_i : i \neq s, t\} + 1, & s(P) \neq \emptyset. \end{cases} \\ b &= \min_s \max\{b_s, a_i : i \neq s\}, \\ c &= \max\{a_1, \ldots, a_p\}. \end{aligned}$$

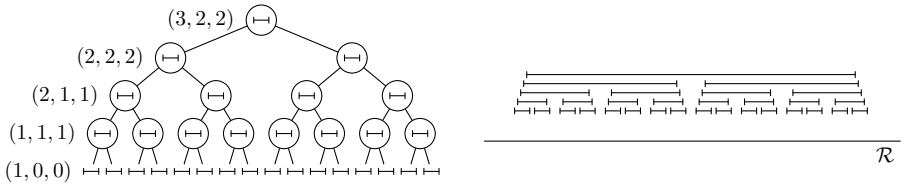

Figure 6 An MPQ-tree representing G with the computed triples (a, b, c) (equal on each level) and a representation minimizing nesting. We have that $\nu(G) = 3$.

▶ **Lemma 7.** *The formulas compute the triple (a, b, c) of a P-node P correctly.*

Proof. Let T be the subtree with P as the root. We know that $a = \nu(G[T])$. If $s(P) = \emptyset$, then $G[T]$ is the disjoint union of $G[T_1], \ldots, G[T_p]$ with $a_i = \nu(G[T_i])$, so $a = \max\{a_1, \ldots, a_p\}$. Otherwise, let $\{w\} = s(P)$. Vertices of every subtree except for two outer T_s and T_t, which we are free to choose, are completely nested inside; so we minimize over all possible choices of $T_s \neq T_t$. For every $i \neq s, t$, we have the optimal nesting a_i increased by one because of $\langle w \rangle$. Now, T_s can be placed on one side of $\langle w \rangle$, while the other side is blocked by the remaining vertices. Therefore, we can simulate this by G_b of $G[T_s]$, where u corresponds to w, so the optimal nesting of $G[T]$ is at least $b_s + 1$, and similarly for T_t and $b_t + 1$.

Concerning b, we have in every representation $\langle u \rangle$ with one side covered by the attached path. Therefore all vertices of $G[T]$ are either nested in $\langle u \rangle$, or intersect the other side. We can always place all vertices of $s(P)$ in such a way that they are not nested in $\langle u \rangle$, and so we can ignore them. All vertices of $G[T]$ except for one subtree T_s, which we can arbitrarily choose, have to be nested in $\langle u \rangle$. Therefore, their optimal nesting a_i is increased by one by $\langle u \rangle$. For T_s, the nesting again corresponds to the optimal nesting of G_b of $G[T_s]$, which is $b_s + 1$. Therefore we get $\nu(G_b) = \min_s \max\big(\{a_i + 1 : i \neq s\} \cup \{b_s + 1\}\big)$, which gives the formula for b since $b = \nu(G_b) - 1$. Concerning c, we have also $\langle v \rangle$. We can choose T_s and T_t so that T_s has optimized nesting with respect to $\langle u \rangle$ and T_t with respect to $\langle v \rangle$. But anyway all intervals of $G[T_s]$ are nested in $\langle v \rangle$ and all intervals of $G[T_t]$ are nested in $\langle u \rangle$. Therefore this optimization is useless and we just get $\nu(G_c) = \max\{a_1, \ldots, a_p\} + 1$. ◀

▶ **Lemma 8.** *For a P-node with p children, the triple (a, b, c) can be computed in $\mathcal{O}(p)$.*

Proof. By Lemma 6, we always have either $b_i = a_i - 1$, or $b_i = a_i$. Only in the former case, we can optimize the nesting by choosing s or t equal i. We call these T_i *savable*. Concerning c, we just find the maximum a_i which can be done in time $\mathcal{O}(p)$. Concerning a and b, we first locate all T_i which maximize a_i. If at least one of them is not savable, say T_j, then $a = a_j + 1$ and $b = a_j$. Otherwise if all are savable, then the values depend on the number of T_i's maximizing a_i. If there are at most two, then $a = a_i$, otherwise $a = a_i + 1$. If there is at most one, then $b = b_i$, otherwise $b = b_i + 1$. ◀

Q-nodes. To optimize Q-nodes, the values c are also required. Let Q be a Q-node with children $T_1, \ldots, T_q$, where $q \geq 3$, each with a triple (a_i, b_i, c_i). We first work with each section $s_i(Q)$ separately. For intervals of sections of Q, we define their *nesting* $\subsetneq$ as follows: $u \subsetneq v$ if v is contained in a section more to the left than all sections of u and in a section more to the right than all sections of u. (In every representation, $\langle u \rangle$ is contained in $\langle v \rangle$.)

For each subtree T_i, we choose whether to optimize its left side, or its right side. For the optimized side, the nesting is increased by b_i, while for the other side it is increased by c_i. We denote the increase in the nesting on the left side by $\bigcirc_i^{\leftarrow}$ and on the right side by $\bigcirc_i^{\rightarrow}$. So we choose either $\bigcirc_i^{\leftarrow} = b_i$ and $\bigcirc_i^{\rightarrow} = c_i$, or $\bigcirc_i^{\leftarrow} = c_i$ and $\bigcirc_i^{\rightarrow} = b_i$.

Suppose that we have some choice of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ for each subtree T_i. Let $\nu(x)$ be the length of the longest chain of nested intervals ending with $\langle x \rangle$, minimized over all possible interval representations of G. We compute the nesting $\nu(x)$ for each x belonging to sections of Q, from bottom to the top of $\subsetneq$. Suppose that we process some x and we know $\nu(y)$ for every direct predecessor of x, and let $s_s(Q)$ be the leftmost section of x and $s_t(Q)$ be the rightmost section. We get the following formula:

$$\nu(x) = \max\{\bigcirc_s^{\leftarrow}, \bigcirc_t^{\rightarrow}, a_i, \nu(y) : s < i < t \text{ and } y \in \mathrm{Pred}(x)\} + 1.$$

Then, we compute (a, b, c) as follows:

$$\begin{aligned} a &= \max\{a_1, \ldots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ b^{\leftarrow} &= \max\{b_1, a_2, \ldots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ b^{\rightarrow} &= \max\{a_1, \ldots, a_{q-1}, b_q, \nu(y) : y \in \mathrm{Pred}(u)\}, \\ c &= \max\{a_1, \ldots, a_q, \nu(y) : y \in \mathrm{Pred}(u) \text{ or } y \in \mathrm{Pred}(v)\}. \end{aligned}$$

We put $b = \min\{b^{\leftarrow}, b^{\rightarrow}\}$. For each of numbers a, b and c, we choose the minimum over all possible choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$.

▶ **Lemma 9.** *The formulas compute the triple (a, b, c) of a Q-node Q correctly.*

Proof. Notice that there exists a representation such that two intervals of sections of Q are nested as in the relation $\subsetneq$ defined above. For each subtree with some choice of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$, we have precomputed what is the optimal nesting and what is the length of a longest chain ending with a vertex not contained in the leftmost clique ($\bigcirc_i^{\leftarrow}$), and of a longest chain not ending with a vertex in the rightmost clique ($\bigcirc_i^{\rightarrow}$). The formulas extends these chains according to the vertices of sections of Q and of added vertices u and v in G_a, G_b, G_c. The length of a longest chain ending with u or v is used. ◀

▶ **Lemma 10.** *For each of a, $b^{\leftarrow}$, $b^{\rightarrow}$ and c, the choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ are done greedily.*

Proof. We argue for a, the argument is similar for the others. Notice that values $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ describe the starting length of some chains. We can easily compute how much are these chains extended by the intervals of sections of Q and by u. We find out which side needs to save more, for which we choose b_i, and for the other side we choose c_i. These values can be chosen for different subtrees independently since their chains start differently. ◀

▶ **Lemma 11.** *For a Q-node Q with q children, the triple (a, b, c) can be computed in time $\mathcal{O}(q + m_Q)$, where m_Q is the number of edges of $G[Q]$.*

Proof. For every interval u in sections of Q, we know its leftmost section and its rightmost section. We compute the DAG of nesting for all vertices of the sections of Q. This can be done by considering all edges, and testing for each whether the pair is nested.

For each of a, $b^{\leftarrow}$, $b^{\rightarrow}$, and c, we add vertex u (and possibly v) to this DAG, together with correct edges according to their nesting. Then for each vertex of this DAG, we compute the length of the longest chain starting in this vertex. This can be done in linear time by processing the DAG from top to the bottom. Next, we process the subtrees T_i. For each, we compute how much chains of length a_i, $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ are prolonged. Then we greedily choose $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ from b_i and c_i. The total consumed time is $\mathcal{O}(q + m_Q)$. ◀

Proof of Theorem 2. If a graph is an interval graph, we can compute its MPQ-tree in time $\mathcal{O}(n + m)$. Then we process the tree from the bottom to the root and compute triples (a, b, c)

for every node, as described above. We output a of the root which is the minimal nesting number $\nu(G)$. By Lemmas 7 and 9, this value is computed correctly. By Lemmas 8 and 11, the running time of the algorithm is $\mathcal{O}(n+m)$. ◀

6 Extending Partial Representations of Minimal Nesting

In this section, we sketch a polynomial-time algorithm of Theorem 3; see the full version [20].

Partial Ordering $\lhd$. The paper [18] characterizes all extendible partial representations in terms of consecutive orderings. A certain partial ordering $\lhd$ of maximal cliques is derived from the partial representation $\mathcal{R}'$.

▶ **Lemma 12** ([18]). *A partial representation $\mathcal{R}'$ is extendible if and only if there exists a consecutive ordering $<$ of the maximal cliques extending $\lhd$. The consecutive orderings of extending representations are exactly the consecutive ordering extending $\lhd$.*

General Idea. We want to minimize the nesting similarly as in Section 5. A partial representation $\mathcal{R}'$ poses three restrictions: (i) Some pre-drawn intervals can be nested in each other which increases the nesting. (ii) The consecutive ordering has to extend $\lhd$ which restricts the possible shuffling of subtrees. (iii) Some subtrees can be optimized differently depending whether they are placed on the left or on the right of its parent.

Concerning (iii), instead of a triple, we compute for every subtree T a tuple $(a, b^{\leftarrow}, c^{\rightarrow}, b^{\rightarrow}, c^{\leftarrow})$. These number are equal to minimum nestings of extending representations of partial representations of G_a, G_b and G_c, in which the pre-drawn vertices of $G[T]$ are pre-drawn the same and we have pre-drawn $\langle u\rangle'$ (and possibly $\langle v\rangle'$) with their added neighbors. The difference between $b^{\leftarrow}$ and $b^{\rightarrow}$ is whether $\langle u\rangle'$ covers the right side of $G[T]$, or covers the left side of $G[T]$; similarly for $c^{\leftarrow}$ and $c^{\rightarrow}$.

For (i), we compute for pre-drawn intervals $\langle x\rangle'$ their nesting $\nu(x)$ in an optimal extending representation of $\mathcal{R}'$ constructed by the algorithm, from bottom to the top as they are encountered in the MPQ-tree. We show that for each *variable* $\nu(x)$ and each $b^{\leftarrow}$, $c^{\rightarrow}$, $b^{\rightarrow}$ and $c^{\leftarrow}$, their values are determined up to one and we can force some subtrees to optimize them. Unfortunately, some of these values cannot be optimized simultaneously, so we introduce the concept of an optimization graph H. The variables are represented by vertices and we have an edge $xy \in E(H)$ if and only if they cannot be optimized at the same time. The optimizable subsets of variables are the independent sets in H.

P-nodes. We have a P-node P with subtrees $T_1, \ldots, T_p$ with the computed tuple $(a_i, b_i^{\leftarrow}, c_i^{\rightarrow}, b_i^{\rightarrow}, c_i^{\leftarrow})$ and the optimization graph H_i for each subtree T_i and $\nu(x)$. Because of (ii), we get that T_s has to be one of the minimal elements $\min(\lhd)$ of $\lhd$ and T_t one of the maximal elements $\max(\lhd)$. Suppose that $T_s \in \min(\lhd)$ and $T_t \in \max(\lhd)$ is fixed.

We first compute $\nu(x)$ for intervals of $s(P)$, according to $\subsetneq$. Assuming that we know $\nu(y)$ for every $y \in \mathrm{Pred}(x)$, we compute $\nu(x)$ as follows:

$$\nu(x) = \max\left\{b_s^{\leftarrow}, b_t^{\rightarrow}, a_i, \nu(y) : i \neq s, t \text{ and } y \in \mathrm{Pred}(x)\right\} + 1.$$

Then, we compute $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$:

$$\begin{aligned} a &= \max\{a_1, \dots, a_p, \nu(y) : y \in \mathrm{Pred}(u)\}. \\ b^{\leftarrow} &= \max\{b_s^{\leftarrow}, a_i, \nu(y) : i \neq s \text{ and } y \in \mathrm{Pred}(u)\}. \\ c^{\rightarrow} &= \max\{b_t^{\rightarrow}, a_i, \nu(y) : i \neq t \text{ and } y \in \mathrm{Pred}(v)\}. \\ b^{\rightarrow} &= \max\{b_t^{\rightarrow}, a_i, \nu(y) : i \neq t \text{ and } y \in \mathrm{Pred}(u)\}. \\ c^{\leftarrow} &= \max\{b_s^{\leftarrow}, a_i, \nu(y) : i \neq s \text{ and } y \in \mathrm{Pred}(v)\}. \end{aligned}$$

The value of a can be always obtained greedily. Since the values given by $T_1, \dots, T_n$ are not fully determined, we create a new optimization subgraph H of T as the disjoint union of $H_1, \dots, H_p$ together with new pre-drawn vertices of $s(P)$ and $b^{\leftarrow}$, $b^{\rightarrow}$, $c^{\leftarrow}$ and $c^{\rightarrow}$, and we add suitable edges.

We run the above computation over all choices of $T_s \in \min(\lhd)$ and $T_t \in \max(\lhd)$. Since $b^{\leftarrow}$ and $c^{\rightarrow}$ might be optimized using different subtrees T_s and T_t than $b^{\rightarrow}$ and $c^{\leftarrow}$, we argue that the situation is not very limited, so we can combine optimizations together in one H.

Q-node. We have a Q-node Q with subtrees $T_1, \dots, T_q$ with the computed tuple $(a_i, b_i^{\leftarrow}, c_i^{\rightarrow}, b_i^{\rightarrow}, c_i^{\leftarrow})$ and the optimization graph H_i for each subtree T_i and $\nu(x)$. Because of (ii), we get that the Q-node might be flippable (which greatly restricts it), or not. For a subtree T_i, we may optimize it with respect to its left side, which is encoded by $b_i^{\leftarrow}$ and $c_i^{\rightarrow}$, or with respect to the right side, which is encoded by $c_i^{\leftarrow}$ and $b_i^{\rightarrow}$. In the former case, we put $\bigcirc_i^{\leftarrow} = b_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow} = c_i^{\rightarrow}$. In the latter case, we put $\bigcirc_i^{\leftarrow} = c_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow} = b_i^{\rightarrow}$. We have trade-offs leading to exponentially many possibilities how $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$ can be chosen, which we capture by the optimization graph H.

Suppose that we fix some choices of $\bigcirc_i^{\leftarrow}$ and $\bigcirc_i^{\rightarrow}$. Unlike for P-nodes, we compute $\nu(x)$ for all intervals of sections of Q, not just for the pre-drawn ones. For two such vertices, we define $\subsetneq$ by nesting of sections, similarly as in Section 5. We process them from the bottom to the top according to $\subsetneq$. Suppose that we process some x and we know $\nu(y)$ for every direct predecessor of x, and let $s_s(Q)$ be the leftmost section of x and $s_t(Q)$ be the rightmost section. We get the following formula:

$$\nu(x) = \max\{\bigcirc_s^{\leftarrow}, \bigcirc_t^{\rightarrow}, a_i, \nu(y) : s < i < t \text{ and } y \in \mathrm{Pred}(x)\} + 1.$$

Then, we compute $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$ as follows:

$$\begin{aligned} a &= \max\{a_1, \dots, a_q, \nu(y) : y \in \mathrm{Pred}(u)\} \\ b^{\leftarrow} &= \max\{b_1^{\leftarrow}, a_i, \nu(y) : i > 1 \text{ and } y \in \mathrm{Pred}(u)\}. \\ c^{\rightarrow} &= \max\{b_n^{\rightarrow}, a_i, \nu(y) : i < n \text{ and } y \in \mathrm{Pred}(v)\}. \\ b^{\rightarrow} &= \max\{b_n^{\rightarrow}, a_i, \nu(y) : i < n \text{ and } y \in \mathrm{Pred}(u)\}. \\ c^{\leftarrow} &= \max\{b_1^{\leftarrow}, a_i, \nu(y) : i > 1 \text{ and } y \in \mathrm{Pred}(v)\}. \end{aligned}$$

By some involved structural properties, each variable is determined up to one. We encode all possibilities into the optimization graph H, similarly as in the case of P-nodes.

Putting Together. We construct a polynomial-time algorithm for RepExt(k-NestedINT) as follows. We process the MPQ-tree from the bottom to the top. We have to argue that each step can be computed in polynomial time (easy) and that it computes the tuples $(a, b^{\leftarrow}, b^{\rightarrow}, c^{\rightarrow}, c^{\leftarrow})$ and the optimization graph H correctly (involved).

Acknowledgment. We want to thank Takehiro Ito and Hirotaka Ono for discussions.

References

1 P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algor.*, 11(4):32:1–32:42, 2015.

2 M. Balko, P. Klavík, and Y. Otachi. Bounded representations of interval and proper interval graphs. In *ISAAC*, volume 8283 of *LNCS*, pages 535–546. Springer, 2013.

3 T. Bläsius and I. Rutter. Simultaneous pq-ordering with applications to constrained embedding problems. *ACM Trans. Algor.*, 12(2):16, 2016.

4 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.

5 M. R. Cerioli, F. de S. Oliveira, and J. L. Szwarcfiter. On counting interval lengths of interval graphs. *Discrete Applied Mathematics*, 159(7):532–543, 2011.

6 S. Chaplick, P. Dorbec, J. Kratochvíl, M. Montassier, and J. Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In *WG'14*, volume 8747 of *LNCS*, pages 139–151. Springer, 2014.

7 S. Chaplick, R. Fulek, and P. Klavík. Extending partial representations of circle graphs. In *Graph Drawing*, volume 8242 of *LNCS*, pages 131–142. Springer, 2013.

8 P. C. Fishburn. Paradoxes of two-length interval orders. *Discrete Math.*, 52(2):165–175, 1984.

9 P. C. Fishburn. Interval graphs and interval orders. *Discrete Math.*, 55(2):135–149, 1985.

10 P. C. Fishburn. *Interval orders and interval graphs: A study of partially ordered sets*. John Wiley & Sons, 1985.

11 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.

12 J. Gajarský, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M. S. Ramanujan, and S. Saurabh. FO model checking on posets of bounded width. In *FOCS 2015*, pages 963–974, 2015.

13 R. Ganian, P. Hlinený, D. Král, J. Obdržálek, J. Schwartz, and J. Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4:11):1–20, 2015.

14 F. Joos, C. Löwenstein, F. de S. Oliveira, D. Rautenbach, and J. L. Szwarcfiter. Graphs of interval count two with a given partition. *Inform. Process. Lett.*, 114(10):542–546, 2014.

15 P. Klavík, J. Kratochvíl, T. Krawczyk, and B. Walczak. Extending partial representations of function graphs and permutation graphs. In *ESA*, volume 7501 of *LNCS*, pages 671–682. Springer, 2012.

16 P. Klavík, J. Kratochvíl, Y. Otachi, I. Rutter, T. Saitoh, M. Saumell, and T. Vyskočil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 2016.

17 P. Klavík, J. Kratochvíl, Y. Otachi, and T. Saitoh. Extending partial representations of subclasses of chordal graphs. *Theoretical Computer Science*, 576:85–101, 2015.

18 P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh, and T. Vyskočil. Extending partial representations of interval graphs. *Algorithmica*, 2016.

19 P. Klavík, J. Kratochvíl, and T. Vyskočil. Extending partial representations of interval graphs. In *TAMC*, volume 6648 of *LNCS*, pages 276–285. Springer, 2011.

20 P. Klavík, Y. Otachi, and J. Šejnoha. On the classes of interval graphs of limited nesting and count of lengths. *CoRR*, abs/1510.03998, 2015.

21 P. Klavík and M. Saumell. Minimal obstructions for partial representation extension of interval graphs. In *ISAAC*, volume 8889 of *LNCS*, pages 401–413, 2014.

22 N. Korte and R. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.

23 R. Leibowitz, S. F. Assmann, and G. W. Peck. The interval count of a graph. *SIAM J. Algebr. Discrete Methods*, 3:485–494, 1982.

24 C. Lekkerkerker and D. Boland. Representation of finite graphs by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962.

25 M. Patrignani. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science*, 17(05):1061–1069, 2006.

26 A. Proskurowski and J. A. Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4):167–176, 1999.

27 F. S. Roberts. Indifference graphs. *Proof techniques in graph theory*, pages 139–146, 1969.

28 D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SICOMP*, 5(2):266–283, 1976.

29 D. Skrien. Chronological orderings of interval graphs. *Discrete Appl. Math.*, 8(1):69–83, 1984.

30 F. J. Soulignac. Bounded, minimal, and short representations of unit interval and unit circular-arc graphs. *CoRR*, abs/1408.3443, 2014.