# Extending Partial Representations of Proper and Unit Interval Graphs☆

Pavel Klavík[a], Jan Kratochvíl[b], Yota Otachi[c], Ignaz Rutter[b,f], Toshiki Saitoh[d], Maria Saumell[e], Tomáš Vyskočil[b]

[a] *Computer Science Institute, Faculty of Mathematics and Physics, Charles University in Prague, Malostranské náměstí 25, 118 00 Prague, Czech Republic.*
[b] *Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University in Prague, Malostranské náměstí 25, 118 00 Prague, Czech Republic.*
[c] *School of Information Science, Japan Advanced Institute of Science and Technology. Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan.*
[d] *Graduate School of Engineering, Kobe University, Rokkodai 1-1, Nada, Kobe, 657-8501, Japan.*
[e] *Department of Mathematics and European Centre of Excellence NTIS (New Technologies for the Information Society), University of West Bohemia, Univerzitní 22, 306 14 Plzeň, Czech Republic.*
[f] *Faculty of Informatics, Karlsruhe Institute of Technology, Fasanengarten 5, 76128 Karlsruhe, Germany.*

**Abstract**

The recently introduced problem of extending partial interval representations asks, for an interval graph with some intervals pre-drawn by the input, whether the partial representation can be extended to a representation of the entire graph. In this paper, we give a linear-time algorithm for extending proper interval representations and an almost quadratic-time algorithm for extending unit interval representations.

We also introduce the more general problem of *bounded representations* of unit interval graphs, where the input constrains the positions of some intervals by lower and upper bounds. We show that this problem is NP-complete for disconnected input graphs and give a polynomial-time algorithm for the special class of instances, where the ordering of the connected components of the input graph along the real line is prescribed. This includes the case of partial representation extension.

The hardness result sharply contrasts the recent polynomial-time algorithm for bounded representations of proper interval graphs [Balko et al. ISAAC'13]. So unless P = NP, proper and unit interval representations have vastly different structure. This explains why partial representation extension problems for these different types of representations require substantially different techniques.

*Keywords:* intersection representation, partial representation extension, bounded representations, restricted representation, proper interval graph, unit interval graph, linear programming

## 1. Introduction

Geometric intersection graphs, and in particular intersection graphs of objects in the plane, have gained a lot of interest for their practical motivations, algorithmic applications, and interesting theoretical properties. Undoubtedly the oldest and the most studied among them are *interval graphs* (INT), i.e., intersection graphs

☆The conference version of this paper appeared in SWAT 2014 [1]. The first, second and sixth authors are supported by ESF Eurogiga project GraDR as GAČR GIG/11/E023, the first author also by GAČR 14-14179S and the first two authors by Charles University as GAUK 196213. The fourth author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD), the sixth author by projects NEXLIZ - CZ.1.07/2.3.00/30.0038, which is co-financed by the European Social Fund and the state budget of the Czech Republic, and ESF EuroGIGA project ComPoSe as F.R.S.-FNRS - EUROGIGA NR 13604.

*Email addresses:* klavik@iuuk.mff.cuni.cz (Pavel Klavík), honza@kam.mff.cuni.cz (Jan Kratochvíl), otachi@jaist.ac.jp (Yota Otachi), rutter@kit.edu (Ignaz Rutter), saitoh@eedept.kobe-u.ac.jp (Toshiki Saitoh), saumell@kma.zcu.cz (Maria Saumell), whisky@kam.mff.cuni.cz (Tomáš Vyskočil)

of intervals on the real line. They were introduced by Hájos [2] in the 1950's and the first polynomial-time recognition algorithm appeared already in the early 1960's [3]. Several linear-time algorithms are known, see [4, 5]. The popularity of this class of graphs is probably best documented by the fact that Web of Knowledge registers over 300 papers with the words "interval graph" in the title. For useful overviews of interval graphs and other intersection-defined classes, see textbooks [6, 7].

Only recently, the following natural generalization of the recognition problem has been considered [8]. The input of the *partial representation extension* problem consists of a graph and a part of the representation and it asks whether it is possible to extend this partial representation to a representation of the entire graph. Klavík et al. [8] give a quadratic-time algorithm for the class of interval graphs and a cubic-time algorithm for the class of proper interval graphs. Two different linear-time algorithms are given for interval graphs [9, 10]. There are also polynomial-time algorithms for function and permutation graphs [11] as well as for circle graphs [12]. Chordal graph representations as intersection graphs of subtrees of a tree [13] and intersection representations of planar graphs [14] are mostly hard to extend.

A related line of research is the complex of simultaneous representation problems, pioneered by Jampani and Lubiw [15, 16], where one seeks representations of two (or more) input graphs such that vertices shared by the input graphs are represented identically in each of the representations. Although in some cases the problem of finding simultaneous representations generalizes the partial representation extension problem, e.g., for interval graphs [9], this connection does not hold for all graph classes. For example, extending a partial representation of a chordal graph is NP-complete [13], whereas the corresponding simultaneous representation problem is polynomial-time solvable [16]. While a similar reduction as the one from [9] works for proper interval graphs, we are not aware of a direct relation between the corresponding problems for unit interval graphs.

In this paper, we extend the line of research on partial representation extension problems by studying the corresponding problems for proper interval graphs (PROPER INT) and unit interval graphs (UNIT INT). Roberts' Theorem [17] states PROPER INT = UNIT INT. It turns out that specific properties of unit interval representations were never investigated since it is easier to work with combinatorially equivalent proper interval representations. It is already noted in [8] that partial representation extension behaves differently for these two classes; see Figure 1a. This is due to the fact that for proper interval graphs, in whose representations no interval is a proper subset of another interval, the extension problem is essentially topological and can be treated in a purely combinatorial manner. On the other hand, unit interval representations, where all intervals have length one, are inherently geometric, and the corresponding algorithms have to take geometric constraints into account.

It has been observed in other contexts that geometric problems are sometimes more difficult than the corresponding topological problems. For example, the partial drawing extension of planar graphs is linear-time solvable [18] for topological drawing but NP-hard for straight-line drawings [19]. Together with Balko et al. [20], our results show that a generalization of partial representation extension exhibits this behavior already in 1-dimensional geometry. The bounded representation problem is polynomial-time solvable for

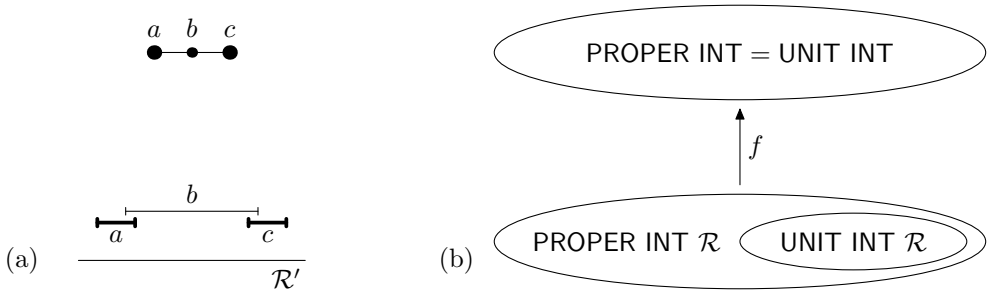


Figure 1: (a) A partial representation which is extendible as a proper interval representation, but not extendible as a unit interval representation. (b) The three structures studied in this paper. The class of proper/unit interval graphs, all proper interval representations and its substructure of all unit interval representations. The denoted mapping $f$ assigns to a representation the graph it represents. The Roberts' Theorem [17] just states that $f$ restricted to unit interval representations is surjective.

proper interval graphs [20] and NP-complete for unit interval graphs. From a perspective of representations, this result separates proper and unit interval graphs. We show that, unless P = NP, the structure of all proper interval representations is significantly different from the structure of all unit interval representations; see Figure 1b.

Next, we formally introduce the problems we study and describe our results.

### *1.1. Classes and Problems in Consideration*

For a graph $G$, an *intersection representation* $\mathcal{R}$ is a collection of sets $\{R_u : u \in V(G)\}$ such that $R_u \cap R_v \neq \emptyset$ if and only if $uv \in E(G)$; so the edges of $G$ are encoded by the intersections of the sets. An intersection-defined class $\mathcal{C}$ is the class of all graphs having intersecting representations with some specific type of sets $R_u$. For example, in an *interval representation* each $R_u$ is a closed interval of the real line. A graph is an *interval graph* if it has an interval representation.

**Studied Classes.** We consider two classes of graphs. An interval representation is called *proper* if no interval is a proper subset of another interval (meaning $R_u \subseteq R_v$ implies $R_u = R_v$). An interval representation is called *unit* if the length of each interval is one. The class of *proper interval graphs* (PROPER INT) consists of all interval graphs having proper interval representations, whereas the class of *unit interval graphs* (UNIT INT) consists of all interval graphs having unit interval representations. Clearly, every unit interval representation is also a proper interval representation.

In an interval representation $\mathcal{R} = \{R_v : v \in V\}$, we denote the left and right endpoint of the interval $R_v$ by $\ell_v$ and $r_v$, respectively. For numbered vertices $v_1, \ldots, v_n$, we denote these endpoints by $\ell_i$ and $r_i$. Note that several intervals may share an endpoint in a representation. When we work with multiple representations, we use $\mathcal{R}'$ and $\overline{\mathcal{R}}$ for them. Their intervals are denoted by $R'_v = [\ell'_v, r'_v]$ and $\bar{R}_v = [\bar{\ell}_v, \bar{r}_v]$.

**Studied Problems.** The *recognition* problem of a class $\mathcal{C}$ asks whether an input graph belongs to $\mathcal{C}$; that is, whether it has a representation by the specific type of sets $R_u$. We study two generalizations of this problem: The *partial representation extension problem*, introduced in [8], and a new problem called the *bounded representation problem*.

A *partial representation* $\mathcal{R}'$ of $G$ is a representation of an induced subgraph $G'$ of $G$. A vertex in $V(G')$ is called *pre-drawn*. A representation $\mathcal{R}$ *extends* $\mathcal{R}'$ if $R_u = R'_u$ for each $u \in V(G')$.

| | |
|---|---|
| **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 $\mathcal{R}'$? |

Suppose, that we are given two rational numbers lbound($v_i$) and ubound($v_i$) for each vertex $v_i$. A representation $\mathcal{R}$ is called a *bounded representation* if lbound($v_i$) $\leq \ell_i \leq$ ubound($v_i$).

| | |
|---|---|
| **Problem:** | BoundRep (Bounded Representation of UNIT INT) |
| **Input:** | A graph $G$ and two rational numbers lbound($v_i$) and ubound($v_i$) for each $v_i \in V(G)$. |
| **Output:** | Does $G$ have a bounded unit interval representation? |

It is easy to see that BoundRep generalizes RepExt(UNIT INT) since we can just put lbound($v_i$) = ubound($v_i$) = $\ell'_i$ for all pre-drawn vertices, and lbound($v_i$) = $-\infty$, ubound($v_i$) = $\infty$ for the remaining vertices.

The bounded representation problem can be considered also for interval graphs and proper interval graphs, where the left and right endpoints of the intervals can be restricted individually. A recent paper of Balko et al. [20] proves that this problem is polynomially solvable for these classes. Note that for unit intervals, it suffices to restrict the left endpoint since $r_i = \ell_i + 1$. The complexity for other classes, e.g. circle graphs, circular-arc graphs, permutation graphs, is open.

*1.2. Contribution and Outline.*

In this paper we present five results. The first is a simple linear-time algorithm for RepExt(PROPER INT), improving over a previous $O(nm)$-time algorithm [8]; it is based on known characterizations, and we present it in Section 3.

**Theorem 1.1.** RepExt(PROPER INT) *can be solved in time* $\mathcal{O}(n+m)$.

We note that this algorithm needs some minor and very natural assumption on the encoding of the input; see Conclusions for details.

Second, in Section 4, we give a reduction from 3-Partition to show that BoundRep is NP-complete for disconnected graphs. The main idea is that prescribed intervals partition the real line into gaps of a fixed width. Integers are encoded in connected components whose unit interval representations require a certain width. By suitably choosing the lower and upper bounds, we enforce that the connected components have to be placed inside the gaps such that they do not overlap.

**Theorem 1.2.** BoundRep *is* NP*-complete.*

Third, in Section 5.1, we give a relatively simple quadratic-time algorithm for the special case of BoundRep where the order of the connected components along the real line is fixed. We formulate this problem as a sequence of linear programs, and we show that each linear program reduces to a shortest-path problem which we solve with the Bellmann-Ford algorithm.

The running time is $O(n^2r + nD(r))$, where $r$ is the total encoding length of the bounds in the input, and $D(r)$ is the time required for multiplying or dividing two numbers whose binary representation has length $r$. This is due to the fact that the numbers specifying the upper and lower bounds for the intervals can be quite close to each other, requiring that the corresponding rationals have an encoding that is superpolynomial in $n$. Clearly, two binary numbers whose representations have length $r$ can be added in $O(r)$ time, explaining the term of $O(n^2r)$ in the running time. However, using Bellmann-Ford for solving the LP requires also the comparison of rational numbers. To be able to do this efficiently, we convert the rational numbers to a common denominator. Hence, the multiplication cost $D(r)$ enters the running time. The best known algorithm achieves $D(r) = \mathcal{O}(r \log r 2^{\log^* r})$ [22].

Fourth, in Sections 5.2–5.6, we show how to reduce the dependency on $r$ to obtain a running time of $O(n^2 + nD(r))$, which may be beneficial for instances with bounds that have a long encoding.

**Theorem 1.3.** BoundRep *with a prescribed ordering* $\blacktriangleleft$ *of the connected components can be solved in time* $\mathcal{O}(n^2 + nD(r))$*, where* $r$ *is the size of the input describing bound constraints.*

Our algorithm is based on shifting intervals. It starts with some initial representation and creates, by a series of transformations, the so-called *left-most representation* of the input graph. The algorithm performs $\mathcal{O}(n^2)$ combinatorial iterations, each taking time $\mathcal{O}(1)$. The additional time $\mathcal{O}(nD(r))$ is used for arithmetic operations with the bounds. The main idea for reducing the running time with respect to the previous approach is to work with short approximations of the involved rational numbers. We compute the precise position of intervals only once, when they reach their final position.

Further, we derive in Sections 4.1, 5.2, and 5.4 many structural results concerning unit interval representations. In particular, we show that all representation of one connected component form a semilattice. We believe that these results might be useful in designing a faster algorithm, attacking other problems, and getting overall better understanding of unit interval representations.

If the number of connected components is small, we can test all possible orderings $\blacktriangleleft$.

**Corollary 1.4.** *For* $c$ *connected components,* BoundRep *can be solved in* $O(c!(n^2 + nD(r)))$ *time.*

Finally, we note that every instance of RepExt(UNIT INT) is an instance of BoundRep. In Section 6, we show how to derive for these special instances a suitable ordering $\blacktriangleleft$ of the connected components, resulting in an efficient algorithm for RepExt(UNIT INT).

**Theorem 1.5.** RepExt(UNIT INT) *can be solved in time* $\mathcal{O}(n^2 + nD(r))$*, where* $r$ *is the size of the input describing positions of pre-drawn intervals.*

All the algorithms described in this paper are also able to certify the extendibility by constructing the required representations.

## 2. Notation, Preliminaries and Structure

As usual, we reserve $n$ for the number of vertices and $m$ for the number of edges of the graph $G$. We denote the set of vertices by $V(G)$ and the set of edges by $E(G)$. For a vertex $v$, we denote the closed neighborhood of $v$ by $N[v] = \{x : vx \in E(G)\} \cup \{v\}$. We also reserve $r$ for the size of the input describing either bound constraints (for the BoundRep problem) or positions of pre-drawn intervals (for RepExt(UNIT INT)). This value $r$ is for the entire graph $G$, and we use it even when we deal with a single component of $G$. We reserve $c$ for the number of components of $G$ (maximal connected subgraphs of $G$).

**(Un)located Components.** Unlike the recognition problem, RepExt cannot generally be solved independently for connected components. A connected component $C$ of $G$ is *located* if it contains at least one pre-drawn interval and *unlocated* if it contains no pre-drawn interval.

Let $\mathcal{R}$ be any interval representation. Then for each component $C$, the union $\bigcup_{u \in C} R_u$ is a connected segment of the real line, and for different components we get disjoint segments. These segments are ordered from left to right, which gives a linear ordering $\blacktriangleleft$ of the components. So we have $c$ components ordered $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$.

**Structure.** The main goal of this paper is to establish Theorem 1.3 and to apply it to solve RepExt(UNIT INT). Since this paper contains several other results, the structure might not be completely clear. Now, we try to sketch the story of this paper.

In Section 3, we describe a key structural lemma of Deng et al. [23]. Using this lemma, we give a simple characterization of extendible instances of RepExt(PROPER INT), which yields the linear-time algorithm of Theorem 1.1. Also, the reader gets more familiar with the basic difficulties we need to deal with in the case of unit interval graphs.

In Section 4, we show two results for the BoundRep problem. First, we give a polynomial bound on the required resolution of the drawing. So there exists a value $\varepsilon$, which is polynomial in the size of the input, such that there exists a representation where, for every $v_i$, the positions $\ell_i$ and $r_i$ belong to the $\varepsilon$-grid $\{k\varepsilon : k \in \mathbb{Z}\}$. Using this, the required representation can be constructed in this $\varepsilon$-grid. Also, we show that the BoundRep problem is in general NP-complete, which proves Theorem 1.2.

Section 5 is the main section of this paper and it deals with the BoundRep problem with a prescribed ordering $\blacktriangleleft$ of the components. First, we describe an LP-based algorithm for solving this problem that solves $2c$ linear programs. Then we derive some structural results concerning the partially ordered set $\mathfrak{Rep}$ of all $\varepsilon$-grid unit interval representations. Using this structure, we conclude the section with a fast combinatorial algorithm for the above linear programs, solving the BoundRep problem in time $\mathcal{O}(n^2 + nD(r))$.

In Section 6, we show using the main theorem that RepExt(UNIT INT) can be solved in time $\mathcal{O}(n^2 + nD(r))$. In Conclusions, we deal with the related problem of simultaneous representations and give some open problems.

## 3. Extending Proper Interval Representations

In this section, we describe how to extend partial representations of proper interval graphs in time $\mathcal{O}(m + n)$. We also give a simple characterization of all extendible instances.

**Indistinguishable Vertices.** Vertices $u$ and $v$ are called *indistinguishable* if $N[u] = N[v]$. The vertices of $G$ can be partitioned into *groups* of (pairwise) indistinguishable vertices. Note that indistinguishable vertices may be represented by the same intervals (and this is actually true for general intersection representations).


Figure 2: Two proper interval representations $\mathcal{R}_1$ and $\mathcal{R}_2$ with the left-to-right orderings $v_1 \lhd v_2 \lhd v_3 \lhd v_4 \lhd v_5 \lhd v_6 \lhd v_7 \lhd v_8$ and $v_2 \lhd v_1 \lhd v_3 \lhd v_4 \lhd v_5 \lhd v_7 \lhd v_6 \lhd v_8$.

Since indistinguishable vertices are not very interesting from the structural point of view, if the structure of the pre-drawn vertices allows it, we want to *prune* the graph to keep only one vertex per group.

Suppose that we are given an instance of RepExt(PROPER INT). We compute the groups of indistinguishable vertices in time $\mathcal{O}(n+m)$ using the algorithm of Rose et al. [24]. Let $u$ and $v$ be two indistinguishable vertices. If $u$ is not pre-drawn, or both vertices are pre-drawn with $R'_u = R'_v$, then we remove $u$ from the graph, and in the final constructed representation (if it exists) we put $R_u = R_v$. For the rest of the section, we shall assume that the input graph and partial representation are pruned. An important property is that for any representation of a pruned graph, it holds that all intervals are pairwise distinct. So if two intervals are pre-drawn in the same position and the corresponding vertices are not indistinguishable, then we stop the algorithm because the partial representation is clearly not extendible.

**Left-to-right ordering.** Roberts [25] gave the following characterization of proper interval graphs:

**Lemma 3.1 (Roberts).** *A graph is a proper interval graph if and only if there exists a linear ordering $v_1 \lhd v_2 \lhd \cdots \lhd v_n$ of its vertices such that the closed neighborhood of every vertex is consecutive.*

This linear order $\lhd$ corresponds to the left-to-right order of the intervals on the real line in some proper interval representation of the graph. In each representation, the order of the left endpoints is exactly the same as the order of the right endpoints, and this order satisfies the condition of Lemma 3.1. For an example of $\lhd$, see Figure 2.

How many different orderings $\lhd$ can a proper interval graph admit? In the case of a general unpruned graph possibly many, but all of them have a very simple structure. In Figure 2, the graph contains two groups $\{v_1, v_2, v_3\}$ and $\{v_6, v_7\}$. The vertices of each group have to appear consecutively in the ordering $\lhd$ and may be reordered arbitrarily. Deng et al. [23] proved the following:

**Lemma 3.2 (Deng et al.).** *For a connected (unpruned) proper interval graph, the ordering $\lhd$ satisfying the condition of Lemma 3.1 is uniquely determined up to local reordering of groups of indistinguishable vertices and complete reversal.*

This lemma is key for partial representation extension of proper interval graphs. Essentially, we just have to deal with a unique ordering (and its reversal) and match the partial representation on it. Notice that in a pruned graph, if two vertices are indistinguishable, then their order is prescribed by the partial representation.

We want to construct a partial ordering $<$ which is a simple representation of all orderings $\lhd$ from Lemma 3.1. There exists a proper interval representation with an ordering $\lhd$ if and only if $\lhd$ extends either $<$ or its reversal. According to Lemma 3.2, $<$ can be constructed by taking an arbitrary ordering $\lhd$ and making indistinguishable vertices incomparable. For the graph in Figure 2, we get

$$(v_1, v_2, v_3) < v_4 < v_5 < (v_6, v_7) < v_8,$$

where groups of indistinguishable vertices are put in brackets. This ordering is unique up to reversal and can be constructed in time $\mathcal{O}(n+m)$ [26].

**Characterization of Extendible Instances.** We give a simple characterization of the partial representation instances that are extendible. We start with connected instances. Let $G$ be a pruned proper interval graph and $\mathcal{R}'$ be a partial representation of its induced subgraph $G'$. Then intervals in $\mathcal{R}'$ are in some left-to-right ordering $<^{\mathcal{R}'}$. (Recall that the pre-drawn intervals are pairwise distinct.)

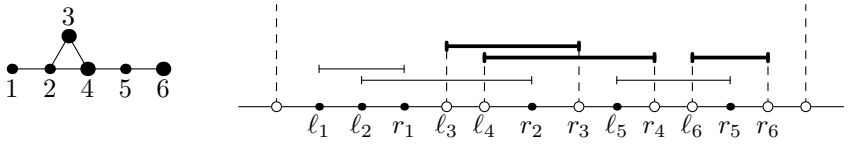


Figure 3: Representation of a component with order $1 \lhd 2 \lhd 3 \lhd 4 \lhd 5 \lhd 6$. First, we compute the common order of the left and right endpoints: $\ell_1 \lessdot \ell_2 \lessdot r_1 \lessdot \ell_3 \lessdot \ell_4 \lessdot r_2 \lessdot r_3 \lessdot \ell_5 \lessdot r_4 \lessdot \ell_6 \lessdot r_5 \lessdot r_6$. The endpoints of the pre-drawn intervals split the segment into several subsegments. We place the remaining endpoints in this order and, within every subsegment, distributed equidistantly.

**Lemma 3.3.** *The partial representation $\mathcal{R}'$ of a connected graph $G$ is extendible if and only if there exists a linear ordering $\lhd$ of $V(G)$ such that:*

(1) *The ordering $\lhd$ extends $<^{\mathcal{R}'}$, and either $<$ or its reversal.*
(2) *Let $R'_u$ and $R'_v$ be two pre-drawn touching intervals, i.e., $r_u = \ell_v$, and let $w$ be any vertex distinct from $u$ and $v$. If $uw \in E(G)$, then $w \lhd v$, and if $vw \in E(G)$, then $u \lhd w$.*

Proof. If there exists a representation $\mathcal{R}$ extending $\mathcal{R}'$, then it is in some left-to-right ordering $\lhd$. Clearly, the pre-drawn intervals are placed the same, so $\lhd$ has to extend $<^{\mathcal{R}'}$. According to Lemma 3.2, $\lhd$ extends $<$ or its reversal. As for (2), clearly $v$ has to be the right-most neighbor of $u$ in $\mathcal{R}$: If $R_w$ is on the right of $R_v$, it would not intersect $R_u$. Similarly, $u$ is the left-most neighbor of $v$.

Conversely, let $v_1 \lhd \cdots \lhd v_n$ be an ordering from the statement of the lemma. We construct a representation $\mathcal{R}$ extending $\mathcal{R}'$ as follows. We compute a common linear ordering $\lessdot$ of the left and right endpoints from left-to-right.[1] We start with the ordering $\ell_1 \lessdot \cdots \lessdot \ell_n$, into which we insert the right endpoints $r_1, \ldots, r_n$ one-by-one. For vertex $v_i$, let $v_j$ be its right-most neighbor in the ordering $\lhd$. Then, we place $r_i$ right before $\ell_{j+1}$ (if $j < n$, otherwise we append $r_i$ to the end of the ordering).

This left-to-right common order $\lessdot$ is uniquely determined by $\lhd$. Since $\lhd$ extends $<^{\mathcal{R}'}$, it is compatible with the partial representation (the pre-drawn endpoints are ordered as in $\lessdot$). To construct the representation, we just place the non-pre-drawn endpoints equidistantly into the gaps between neighboring pre-drawn endpoints (or to the left or right of $\mathcal{R}'$). It is important that, if two pre-drawn endpoints $\ell_i$ and $r_j$ share their position, then according to condition (2) there is no endpoint placed in between of $\ell_i$ and $r_j$ in $\lessdot$ (otherwise one of the two implications would not hold, depending whether a left endpoint is intersected in between, or a right one). See Figure 3 for an example.

We argue correctness of the constructed representation $\mathcal{R}$. First, it extends $\mathcal{R}'$, since the pre-drawn intervals are not modified. Second, it is a correct interval representation: Let $v_i$ and $v_j$ be two vertices with $v_i \lhd v_j$, and let $v_k$ be the right-most neighbor of $v_i$ in $\lhd$. If $v_iv_j \in E(G)$, then $\ell_i \lessdot \ell_k \lessdot r_i$ and, by consecutivity of $N[u]$ in $\lhd$, we have $\ell_j \lessdot \ell_k$. Therefore, $R_{v_i}$ and $R_{v_j}$ intersect. If $v_iv_j \notin E(G)$ and $v_j \neq v_{k+1}$, then $r_i \lessdot \ell_{k+1} \lessdot \ell_j$, so $R_{v_i}$ and $R_{v_j}$ do not intersect. If $v_iv_j \notin E(G)$ and $v_j = v_{k+1}$, then $r_i \lessdot \ell_{k+1}$ and $R_{v_i}$ and $R_{v_j}$ do not intersect. Finally, we argue that $\mathcal{R}$ is a proper interval representation. In $\lessdot$ the order of the left endpoints is the same as the order of the right-endpoints, since $r_{i+1}$ is always placed on the right of $r_i$ in $\lessdot$.

We conclude that the representation $\mathcal{R}$ can be made small enough to fit into any open segment of the real line that contains all pre-drawn intervals. □

Now, we are ready to characterize general solvable instances.

**Lemma 3.4.** *A partial representation $\mathcal{R}'$ of a graph $G$ is extendible if and only if*

(1) *for each component $C$, the partial representation $\mathcal{R}'_C$ consisting of the pre-drawn intervals in $C$ is extendible, and*

[1] Notice that, in the partial representation, some intervals may share position. But if two endpoints $\ell_i$ and $r_j$ share the position, then $v_iv_j \in E(G)$ and we break the tie by setting $\ell_i \lessdot r_j$.

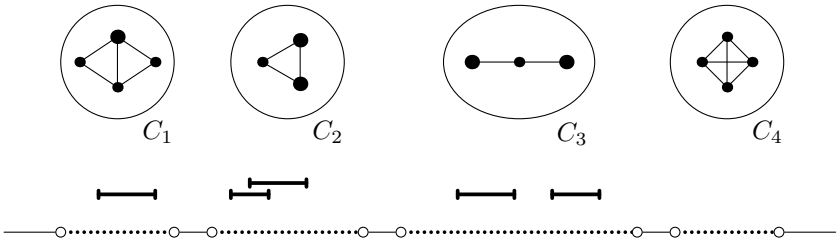


Figure 4: An example of a graph with four components $C_1, \ldots, C_4$. The pre-drawn intervals give the order of the located components $C_1 \blacktriangleleft C_2 \blacktriangleleft C_3$. The non-located component $C_4$ is placed to the right. For each component, we reserve some segment in which we construct the representation.

(2) *pre-drawn vertices of each component are consecutive in* $<^{\mathcal{R}'}$.

Proof. The necessity of (1) is clear. For (2), if some component $C$ would not have its pre-drawn vertices consecutive in $<^{\mathcal{R}'}$, then $\bigcup_{u \in C} R_u$ would not be a connected segment of the real line (contradicting existence of $\blacktriangleleft$ from Preliminaries).

Now, if the instance satisfies both conditions we can construct a correct representation $\mathcal{R}$ extending $\mathcal{R}'$ as follows. Using (2), the located components are ordered from left to right, and we assign pairwise disjoint *open segments* containing all their pre-drawn intervals (there is a non-empty gap between located components we can use). To unlocated components, we assign pairwise disjoint open segments to the right of the right-most located component. See Figure 4. For each component, we construct a representation in its open segment, using the construction in the proof of Lemma 3.3. □

We are ready to prove that RepExt(PROPER INT) can be solved in time $\mathcal{O}(n+m)$:

Proof (Theorem 1.1). We just use the characterization by Lemma 3.4, of which the conditions (1) and (2) can be easily checked in time $\mathcal{O}(n+m)$. For Lemma 3.3, we check for each component both constraints (1) and (2). To check (2), we compute for $<$ and its reversal the unique orderings $\lhd$. We test for each of them whether each touching pair of pre-drawn intervals is placed in $\lhd$ according to (2).

If necessary, a representation $\mathcal{R}$ can be constructed in the same running time since the proofs of Lemmas 3.3 and 3.4 are constructive. □

## 4. Bounded Representations of Unit Interval Graphs

In this section, we deal with bounded representations. An input of BoundRep consists of a graph $G$ and, for each vertex $v_i$, a *lower bound* $\text{lbound}(v_i)$ and an *upper bound* $\text{ubound}(v_i)$. (We allow $\text{lbound}(v_i) = -\infty$ and $\text{ubound}(v_i) = +\infty$.) The problem asks whether there exists a unit interval representation $\mathcal{R}$ of $G$ such that $\text{lbound}(v_i) \leq \ell_i \leq \text{ubound}(v_i)$ for each interval $v_i$. Such a representation is called a *bounded representation*.

Since unit interval representations are proper interval representations, all properties of proper interval representations described in Section 3 hold, in particular the properties of orderings $\lhd$ and $<$.

### *4.1. Representations in $\varepsilon$-grids*

Endpoints of intervals can be positioned at arbitrary real numbers. For the purpose of the algorithm, we want to work with representations drawn in limited resolution. For a given instance of the bounded representation problem, we want to find a lower bound for the required resolution such that this instance is solvable if and only if it is solvable in this limited resolution.

More precisely, we want to represent all intervals so that their endpoints correspond to points on some grid. For a value $\varepsilon = \frac{1}{K} > 0$, where $K$ is an integer, the $\varepsilon$-grid is the set of points $\{k\varepsilon : k \in \mathbb{Z}\}$.[2] For a given

[2] If $\varepsilon$ was not of the form $\frac{1}{K}$, then the grid could not contain both left and right endpoints of the intervals. We reserve $K$ for the value $\frac{1}{\varepsilon}$ in this paper.

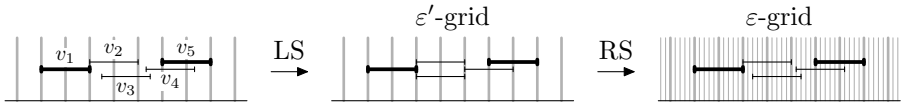


Figure 5: In the first step, we shift intervals to the left to the $\varepsilon'$-grid. The left shifts of $v_1, \ldots, v_5$ are $(0, 0, \frac{1}{2}\varepsilon', \frac{1}{3}\varepsilon', 0)$. In the second step, we shift to the right in the refined $\varepsilon$-grid. Right shifts have the same relative order as left shifts: $(0, 0, 2\varepsilon, \varepsilon, 0)$.

instance of BOUNDREP, we ask which value of $\varepsilon$ ensures that we can construct a representation having all endpoints on the $\varepsilon$-grid. So the value of $\varepsilon$ is the resolution of the drawing.

If there are no bounds, every unit interval graph has a representation in the grid of size $\frac{1}{n}$ [26]. In the case of BOUNDREP, the size of the grid has to depend on the values of the bounds. Consider all values $\text{lbound}(v_i)$ and $\text{ubound}(v_i)$ distinct from $\pm\infty$, and express them as irreducible fractions $\frac{p_1}{q_1}, \frac{p_2}{q_2}, \cdots, \frac{p_b}{q_b}$. Then we define:

$$\varepsilon' := \frac{1}{\text{lcm}(q_1, q_2, \ldots, q_b)}, \qquad \text{and} \qquad \varepsilon := \frac{\varepsilon'}{n}, \tag{1}$$

where $\text{lcm}(q_1, q_2, \ldots, q_b)$ denotes the *least common multiple* of $q_1, \ldots, q_b$. It is important that the size of this $\varepsilon$ written in binary is $\mathcal{O}(r)$. We show that the $\varepsilon$-grid is sufficient to construct a bounded representation:

**Lemma 4.1.** *If there exists a bounded representation $\mathcal{R}'$ for an input of the problem* BOUNDREP*, there exists a bounded representation $\mathcal{R}$ in which all intervals have endpoints on the $\varepsilon$-grid, where $\varepsilon$ is defined by (1).*

PROOF. We construct an $\varepsilon$-grid representation $\mathcal{R}$ from $\mathcal{R}'$ in two steps. First, we shift intervals to the left, and then we shift intervals slightly back to the right. For every interval $v_i$, the sizes of the left and right shifts are denoted by $\text{LS}(v_i)$ and $\text{RS}(v_i)$ respectively. The shifting process is shown in Figure 5.

In the first step, we consider the $\varepsilon'$-grid and shift all the intervals to the left to the closest grid-point (we do not shift an interval if its endpoints are already on the grid). Original intersections are kept by this shifting, since if $x$ and $y$ are two endpoints satisfying $x \leq y$ before the left-shift, then $x \leq y$ also holds after the left-shift. So if $v_iv_j \in E$ and $\ell_i \leq \ell_j \leq r_i$ before the shift, then these inequalities are preserved by the shifting. On the other hand, we may introduce additional intersections by shifting two non-intersecting intervals to each other. In this case, after the left-shift, the intervals only touch; for an example, see vertices $v_2$ and $v_4$ in Figure 5.

The second step shifts the intervals to the right in the refined $\varepsilon$-grid to remove the additional intersections created by the first step. The right-shift is a mapping

$$\text{RS} : \{v_1, \ldots, v_n\} \to \{0, \varepsilon, 2\varepsilon, \ldots, (n-1)\varepsilon\}$$

having the *right-shift property*: For all pairs $(v_i, v_j)$ with $r_i = \ell_j$, $\text{RS}(v_i) \geq \text{RS}(v_j)$ if and only if $v_iv_j \in E$. So the right-shift property ensures that RS fixes wrongly represented touching pairs created by LS.

To construct such a mapping RS, notice that if we relax the image of RS to $[0, \varepsilon')$, the reversal of LS would have the right-shift property, since it produces the original correct representation $\mathcal{R}'$. But the right-shift property depends only on the relative order of the shifts and not on the precise values. Therefore, we can construct RS from the reversal of LS by keeping the shifts in the same relative order. If $\text{LS}(v_i)$ is one of the $k$th smallest shifts, we set $\text{RS}(v_i) = (k-1)\varepsilon$.[3] See Figure 5.

We finally argue that these shifts produce a correct $\varepsilon$-grid representation. The right-shift does not create additional intersections: After LS non-intersecting pairs are at distance at least $\varepsilon' = n\varepsilon$, and by RS they can get closer by at most $(n-1)\varepsilon$. Also, if after LS two intervals overlap by at least $\varepsilon'$, their intersection is not removed by RS. The only intersections which are modified by RS are touching pairs of intervals $(v_i, v_j)$ having $r_i = \ell_j$ after LS. The mapping RS shifts these pairs correctly according to the edges of the graph.

[3]In other words, for the smallest shifts we assign the right-shift 0; for the second smallest shifts, we assign $\varepsilon$; for the third smallest shifts, $2\varepsilon$; and so on.

Next we look at the bound constraints. If, before the shifting, $v_i$ was satisfying $\ell_i \geq \text{lbound}(v_i)$, then this is also satisfied after $\text{LS}(v_i)$ since the $\varepsilon'$-grid contains the value $\text{lbound}(v_i)$. Obviously, the inequality is not broken after $\text{RS}(v_i)$. As for the upper bound, if $\text{LS}(v_i) = 0$ and $\text{RS}(v_i) = 0$, then the bound is trivially satisfied. Otherwise, after $\text{LS}(v_i)$ we have $\ell_i \leq \text{ubound}(v_i) - \varepsilon'$, so the upper bound still holds after $\text{RS}(v_i)$. □

Additionally, Lemma 4.1 shows that it is always possible to construct an $\varepsilon$-grid representation having the same topology as the original representation, in the sense that overlapping pairs of intervals keep overlapping, and touching pairs of intervals keep touching. Also notice that both representations $\mathcal{R}$ and $\mathcal{R}'$ have the same order of the intervals.

In the standard unit interval graph representation problem, no bounds on the positions of the intervals are given, and we get $\varepsilon' = 1$ and $\varepsilon = \frac{1}{n}$. Lemma 4.1 proves in a particularly clean way that the grid of size $\frac{1}{n}$ is sufficient to construct unrestricted representations of unit interval graphs. Corneil et al. [26] show how to construct this representation directly from the ordering $<$, whereas we use some given representation to construct an $\varepsilon$-grid representation.

### *4.2. Hardness of* BoundRep

In this subsection we focus on hardness of bounded representations of unit interval graphs. We prove Theorem 1.2 stating that BoundRep is NP-complete.

We reduce the problem from 3-Partition. An input of 3-Partition consists of natural numbers $k$, $M$, and $A_1, \ldots, A_{3k}$ such that $\frac{M}{4} < A_i < \frac{M}{2}$ for all $i$, and $\sum A_i = kM$. The question is whether it is possible to partition the numbers $A_i$ into $k$ triples such that each triple sums to exactly $M$. This problem is known to be strongly NP-complete (even if all numbers have polynomial sizes) [27].

Proof (Theorem 1.2). According to Lemma 4.1, if there exists a representation satisfying the bound constraints, then there also exists an $\varepsilon$-grid representation with this property. Since the length of $\varepsilon$ given by (1), written in binary, is polynomial in the size of the input, all endpoints can be placed in polynomially-long positions. Thus we can guess the bounded representation and the problem belongs to NP.

Let us next prove that the problem is NP-hard. For a given input of 3-Partition, we construct the following unit interval graph $G$. For each number $A_i$, we add a path $P_{2A_i}$ (of length $2A_i - 1$) into $G$ as a separate component. For all vertices $x$ in these paths, we set bounds

$$\text{lbound}(x) = 1 \qquad \text{and} \qquad \text{ubound}(x) = k \cdot (M + 2).$$

In addition, we add $k + 1$ independent vertices $v_0, v_1, \ldots, v_k$, and make their positions in the representation fixed:

$$\text{lbound}(v_i) = \text{ubound}(v_i) = i \cdot (M + 2).$$

See Figure 6 for an illustration of the reduction. Clearly, the reduction is polynomial.

We now argue that the bounded representation problem is solvable if and only if the given input of 3-Partition is solvable. Suppose first that the bounded representation problem admits a solution. There are $k$ gaps between the fixed intervals $v_0, \ldots, v_k$ each of which has space less than $M + 1$. (The length of the gap is $M + 1$ but the endpoints are taken by $v_i$ and $v_{i+1}$.) The bounds of the paths force their representations to be inside these gaps, and each path lives in exactly one gap. Hence the representation induces a partition of the paths.

Now, the path $P_{2A_i}$ needs space at least $A_i$ in every representation since it has an independent set of the size $A_i$. The representations of the paths may not overlap and the space in each gap is less than $M + 1$, hence the sum of all $A_i$'s in each part is at most $M$. Since the total sum of $A_i$'s is exactly $kM$, the sum in each part has to be $M$. Thus the obtained partition solves the 3-Partition problem.

Conversely, every solution of 3-Partition can be realized in this way. □

## 5. Bounded Representations of Unit Interval Graphs with Prescribed Ordering

In this section, we deal with the BoundRep problem when a fixed ordering $\blacktriangleleft$ of the components is prescribed. First we solve the problem using linear programming. Then we describe additional structure of bounded representations, and using this structure we construct an almost quadratic-time algorithm that solves the linear programs.

### *5.1. LP Approach for* BoundRep

According to Lemma 3.2, each component of $G$ can be represented in at most two different ways, up to local reordering of groups of indistinguishable vertices. Unlike the case of proper interval graphs, we cannot arbitrarily choose one of the orderings, since neighboring components restrict each other's space. For example, only one of the two orderings for the component $C_1$ in Figure 7 makes a representation of $C_2$ possible.

In the algorithm, we process components $C_1 \blacktriangleleft C_2 \blacktriangleleft \cdots \blacktriangleleft C_c$ from left to right and construct representations for them. When we process a component $C_t$, we want to represent it on the right of the previous component $C_{t-1}$, and we want to push the representation of $C_t$ as far to the left as possible, leaving as much space for $C_{t+1}, \ldots, C_c$ as possible.

Now, we describe in details, how we process a component $C_t$. We calculate by the algorithm of Corneil et al. the partial ordering $<$ described in Section 3 and its reversal. The elements that are incomparable by these partial orderings are vertices of the same group of indistinguishable vertices. For these vertices, the following holds:

**Lemma 5.1.** *Suppose there exists some bounded representation $\mathcal{R}$. Then there exists a bounded representation $\mathcal{R}'$ such that, for every indistinguishable pair $v_i$ and $v_j$ satisfying* $\mathrm{lbound}(v_i) \leq \mathrm{lbound}(v_j)$*, it holds that $\ell'_i \leq \ell'_j$.*

Proof. Given a representation $\mathcal{R}$, we call a pair $(v_i, v_j)$ *bad* if $v_i$ and $v_j$ are indistinguishable, $\mathrm{lbound}(v_i) \leq \mathrm{lbound}(v_j)$ and $\ell_i > \ell_j$. We describe a process which iteratively constructs $\mathcal{R}'$ from $\mathcal{R}$, by constructing a sequence of representations $\mathcal{R} = \mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_k = \mathcal{R}'$, where the positions in a representation $\mathcal{R}_s$ are denoted by $\ell_i^s$'s.

In each step $s$, we create $\mathcal{R}_s$ from $\mathcal{R}_{s-1}$ by fixing one bad pair $(v_i, v_j)$: we set $\ell_i^s = \ell_j^{s-1}$ and the rest of the representation remains the same. Since $v_i$ and $v_j$ are indistinguishable and $\mathcal{R}_{s-1}$ is correct, the obtained $\mathcal{R}_s$ is a representation. Regarding bound constraints,

$$\mathrm{lbound}(v_i) \leq \mathrm{lbound}(v_j) \leq \ell_j^{s-1} = \ell_i^s < \ell_i^{s-1} \leq \mathrm{ubound}(v_i),$$

so the bounds of $v_i$ are satisfied.

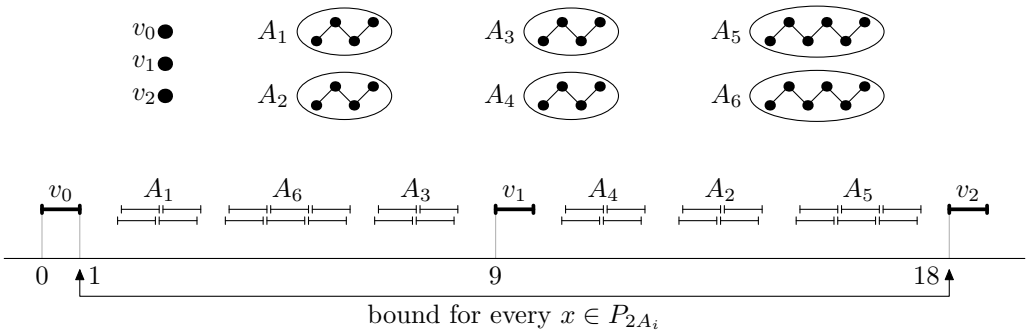


Figure 6: We consider the following input for 3-Partition: $k = 2$, $M = 7$, $A_1 = A_2 = A_3 = A_4 = 2$ and $A_5 = A_6 = 3$. The associated unit interval graph is depicted on top, and at the bottom we find one of its correct bounded representations, giving 3-partitioning $\{A_1, A_3, A_6\}$ and $\{A_2, A_4, A_5\}$.

Now, in each $\mathcal{R}_s$ the set of all left endpoints is a subset of the set of all left endpoints of $\mathcal{R}$. In each step, we move one left-endpoint to the left, so each endpoint is moved at most $n-1$ times. Hence the process terminates after $\mathcal{O}(n^2)$ iterations and produces a representation $\mathcal{R}'$ without bad pairs as requested. □

For $<$ and its reversal, we use Lemma 5.1 to construct linear orderings $\lhd$: If $v_i$ and $v_j$ belong to the same group of indistinguishable vertices and $\mathrm{lbound}(v_i) < \mathrm{lbound}(v_j)$, then $v_i \lhd v_j$. If $\mathrm{lbound}(v_i) = \mathrm{lbound}(v_j)$, we choose any order $\lhd$ between $v_i$ and $v_j$.

We obtain two total orderings $\lhd$, and we solve a linear program for each of them. Let $v_1 \lhd v_2 \lhd \cdots \lhd v_k$ be one of these orderings. We denote the right-most endpoint of a representation of a component $C_t$ by $E_t$. Additionally, we define $E_0 = -\infty$. Let $\varepsilon$ be defined as in (1). We modify all lower bounds by putting $\mathrm{lbound}(v_i) = \max\{\mathrm{lbound}(v_i), E_{t-1}+\varepsilon\}$ for every interval $v_i$, which forces the representation of $C_t$ to be on the right of the previously constructed representation of $C_{t-1}$. The linear program has variables $\ell_1, \ldots, \ell_k$, and it minimizes the value of $E_t$. We solve:

$$
\begin{array}{lrll}
\text{Minimize:} & E_t := \ell_k + 1, & & \\
\text{subject to:} & \ell_i \le \ell_{i+1}, & \forall i = 1, \ldots, k-1, & (2)\\
& \ell_i \ge \mathrm{lbound}(v_i), & \forall i = 1, \ldots, k, & (3)\\
& \ell_i \le \mathrm{ubound}(v_i), & \forall i = 1, \ldots, k, & (4)\\
& \ell_i \ge \ell_j - 1, & \forall v_i v_j \in E(G), v_i \lhd v_j, & (5)\\
& \ell_i + \varepsilon \le \ell_j - 1, & \forall v_i v_j \notin E(G), v_i \lhd v_j. & (6)
\end{array}
$$

We solve the same linear program for the other ordering of the vertices of $C_t$. If none of the two programs is feasible, we report that no bounded representation exists. If exactly one of them is feasible, we keep the values obtained for $\ell_1, \ldots, \ell_k$ and $E_t$, and process the next component $C_{t+1}$. If the two problems are feasible, we keep the solution in which the value of $E_t$ is smaller, and process $C_{t+1}$.

**Lemma 5.2.** *Let the representation of $C_{t-1}$ be fixed. Every bounded $\varepsilon$-grid representation of the component $C_t$ with the left-to-right order $v_1 \lhd \cdots \lhd v_k$ which is on the right of the representation of $C_{t-1}$ satisfies constraints (3)–(6).*

Proof. Constraints of types (3) and (4) are satisfied, since the representation is bounded and on the right of $C_{t-1}$. Constraints of type (5) correspond to a correct representation of intersecting pairs of intervals. The non-intersecting pairs of an $\varepsilon$-grid representation are at distance at least $\varepsilon$, which makes constraints of type (6) satisfied. □

Now, we are ready to show:

**Proposition 5.3.** *The* BoundRep *problem with prescribed ◀ can be solved in polynomial time.*

Proof. Concerning the running time, it depends polynomially on the sizes of $n$ and $\varepsilon$, which are polynomial in the size of the input $r$. It remains to show correctness.

Suppose that the algorithm returns a candidate for a bounded representation. The formulation of the linear program ensures that it is a correct representation: Constraints of type (2) make the representation

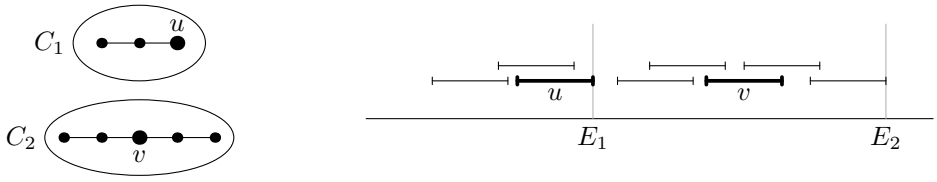


Figure 7: The positions of the vertices $u$ and $v$ are fixed by the bound constraints. The component $C_1$ can only be represented with $u$ being the right-most interval, since otherwise $C_1$ would block space for the component $C_2$.

respect $\lhd$. Constraints of type (3) and (4) enforce that the given lower and upper bounds for the positions of the intervals are satisfied, force the prescribed ordering $\blacktriangleleft$ on the representation of $G$, and force the drawings of the distinct components to be disjoint. Finally, constraints of type (2), (5) and (6) make the drawing of the vertices of a particular component $C_t$ to be a correct representation.

Suppose next that a bounded representation exists. According to Lemma 4.1 and Lemma 5.1, there also exists an $\varepsilon$-grid bounded representation $\mathcal{R}'$ having the order in the indistinguishable groups as defined above. So for each component $C_t$, one of the two orderings $\lhd$ constructed for the linear programs agrees with the left-to-right order of $C_t$ in $\mathcal{R}'$.

We want to show that the representation of each component $C_t$ in $\mathcal{R}'$ gives a solution to one of the two linear programs associated to $C_t$. We denote by $E'_t$ the value of $E_t$ in the representation $\mathcal{R}'$, and by $E_t^{\min}$ the value of $E_t$ obtained by the algorithm after solving the two linear programming problems associated to $C_t$. We show by induction on $t$ that $E_t^{\min} \leq E'_t$, which specifically implies that $E_t^{\min}$ exists and at least one of the linear programs for $C_t$ is solvable.

We start with $C_1$. As argued above, the left-to-right order in $\mathcal{R}'$ agrees with one of the orderings $\lhd$, so the representation of $C_1$ satisfies the constraints (2). Since $E_0 = -\infty$, the lower bounds are not modified. By Lemma 5.2, the rest of the constraints are also satisfied. Thus the representation of $C_1$ gives a feasible solution for the program and gives $E_1^{\min} \leq E'_1$.

Assume now that, for some $C_t$ with $t \geq 1$, at least one of the two linear programming problems associated to $C_t$ admits a solution, and from induction hypothesis we have $E_t^{\min} \leq E'_t$. In $\mathcal{R}'$, two neighboring components are represented at distance at least $\varepsilon$. Therefore for every vertex $v_i$ of $C_{t+1}$, it holds $\ell_i \geq E'_t + \varepsilon \geq E_t^{\min} + \varepsilon$, so the modification of the lower bound constraints is satisfied by $\mathcal{R}'$. Similarly as above using Lemma 5.2, the representation of $C_{t+1}$ in $\mathcal{R}'$ satisfies the remaining constraints. It gives some solution to one of the programs and we get $E_{t+1}^{\min} \leq E'_{t+1}$.

In summary, if there exists a bounded representation, for each component $C_t$ at least one of the two linear programming problems associated to $C_t$ admits a solution. Therefore, the algorithm returns a correct bounded representation $\mathcal{R}$ (as discussed in the beginning of the proof). We note that $\mathcal{R}$ does not have to be an $\varepsilon$-grid representation since the linear program just states that non-intersecting intervals are at distance at least $\varepsilon$. To construct an $\varepsilon$-grid representation if necessary, we can proceed as in the proof of Lemma 4.1. □

We note that it is possible to reduce the number of constraints of the linear program from $\mathcal{O}(k^2)$ to $\mathcal{O}(k)$, since neighbors of each $v_i$ appear according to Lemma 3.1 consecutively in $\lhd$. Using the ordering constraints (2), we can replace constraints (5) and (6) by a linear number of constraints as follows. For each $v_j$, there are two cases. If $v_j$ is adjacent to all vertices $v_i$ such that $v_i \lhd v_j$, then we only state the constraint (5) for $v_1$ and $v_j$. Otherwise, let $v_i$ be the rightmost vertex such that $v_i \lhd v_j$ and $v_i v_j \notin E$. Then we only state the constraint (5) for $v_{i+1}$ and $v_j$, and the constraint (6) for $v_i$ and $v_j$. This is equivalent to the original formulation of the problem.

In general, any linear program can be solved in $\mathcal{O}(n^{3.5} r^2 \log r \log\log r)$ time by using Karmarkar's algorithm [28]. However, our linear program is special which allows to use faster techniques:

**Proposition 5.4.** *The* BoundRep *problem with prescribed* $\blacktriangleleft$ *can be solved in time* $\mathcal{O}(n^2 r + nD(r))$.

Proof. Without loss of generality, we assume that the upper and lower bounds restrict the final representation (if it exists) to lie in the interval $[1, n+3]$. For a given $i$, let $j_i$ be the index such that $v_{j_i}$ is the rightmost neighbor of $v_i$ in $\lhd$. Let $h_i$ be the index such that $v_{h_i}$ is the rightmost vertex such that $v_{h_i} \lhd v_i$ and $v_{h_i} v_i \notin E$. (Notice that $h_i$ might not be defined, in which case we ignore inequalities containing it.)

We replace the variables $\ell_1, \ldots, \ell_k$ by $x_0, \ldots, x_k$ such that $\ell_i = x_i - x_0$. We want to solve the following

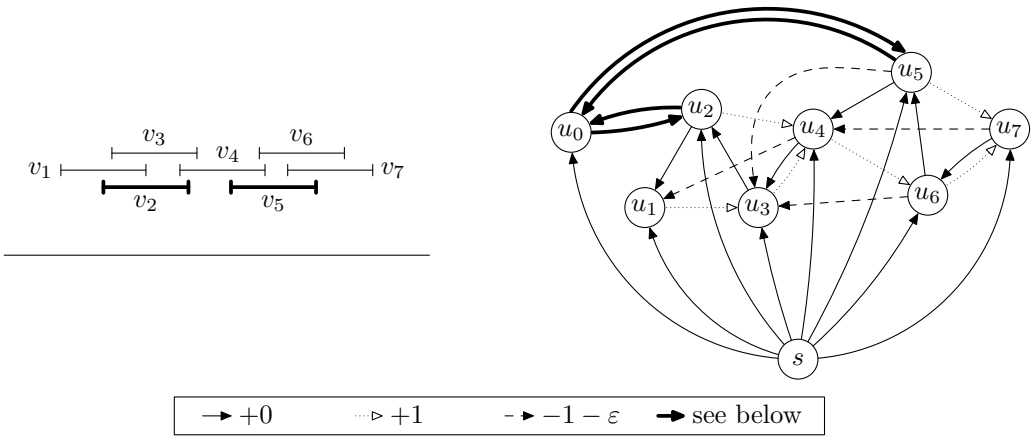


Figure 8: On the left, a unit interval graph with two pre-drawn intervals. On the right, the corresponding digraph $D$ with the weight encoded as in the box. The weights of the bold edges are as follows: $w(u_0, u_2) = \text{ubound}(v_2)$, $w(u_0, u_5) = \text{ubound}(v_5)$, $w(u_2, u_0) = -\text{lbound}(v_2)$, and $w(u_5, u_0) = -\text{lbound}(v_5)$.

linear system:

$$\begin{array}{lll}
\text{Minimize:} & E_t := x_k - x_0 + 1, & \\
\text{subject to:} & x_i - x_{i+1} \leq 0, & \forall i = 1, \ldots, k-1, \\
& x_0 - x_i \leq -\text{lbound}(v_i), & \forall i = 1, \ldots, k, \\
& x_i - x_0 \leq \text{ubound}(v_i), & \forall i = 1, \ldots, k, \\
& x_{j_i} - x_i \leq 1, & \forall i = 1, \ldots, k, \\
& x_{h_i} - x_i \leq -1 - \varepsilon, & \forall i = 1, \ldots, k.
\end{array}$$

The obtained linear program is a *system of difference constraints*, since each inequality has the form $x_i - x_j \leq b_{i,j}$.

Following [21, Chapter 24.4], if the system is feasible, a solution, which is not necessarily optimal, can be found as follows. We define a weighted digraph $D$ as follows. As the vertices, we have $V(D) = \{s, u_0, u_1, \ldots, u_k\}$ where $u_i$ corresponds to $x_i$ and $s$ is a special vertex. For the edges $\vec{E}(D)$, we first have an edge $(s, u_i)$ of the weight zero for every $u_i$. Then for every constraint $x_i - x_j \leq b_{i,j}$, we add the edge $(u_j, u_i)$ of the weight $b_{i,j}$. See Figure 8.

As proved in [21, Chapter 24.4], there are two possible cases. If $G$ *contains* a negative-weight cycle, then there is no feasible solution for the system. If $G$ *does not contain* negative-weight cycles, then we define $\delta(s, u_i)$ as the weight of the minimum-weight path connecting $s$ to $u_i$ in $G$. Then we put $x_i = \delta(s, u_i)$ for each $i$ which defines a feasible solution of the system. Moreover, this solution minimizes the objective function $\max\{x_i\} - \min\{x_i\}$. We next show that this function is equivalent to the objective function in our linear program.

Suppose that we have a solution of our system, satisfying the constraints but not necessarily optimizing the objective function. Because of our assumption that the representation lies in the interval $[1, n+3]$, we know that $\ell_i > 0$ for all $i$. Therefore, $x_i > x_0$. So $\min\{x_i\}$ is always attained by $x_0$, while $\max\{x_i\}$ is always attained by $x_k$. So minimization of the objective function $\max\{x_i\} - \min\{x_i\}$ is equivalent to the original minimization of $E_t = x_k - x_0 + 1$.

In order to find a negative-weight cycle in $D$ or, alternatively, compute the weight of the minimum-weight paths from $s$ to all the other vertices of $D$, we use the Bellman-Ford algorithm. Notice that Dijkstra's algorithm cannot be used in this case, since some edges of $D$ have negative weight. We next analyze the running time of the whole procedure.

We assume that the cost of arithmetic operations with large numbers is not constant. The algorithm

computes the value $\varepsilon$ in the beginning which can be clearly done in time $\mathcal{O}(nD(r))$. (Instead of the least common multiple we can simply compute the product of $q_i$'s.)

Afterwards, we compute the weights of the edges of $D$ as multiples of $\varepsilon$, which takes time $\mathcal{O}(kD(r))$. Then each step of the Bellman-Ford algorithm requires time $\mathcal{O}(r)$, and the algorithm runs $\mathcal{O}(k^2)$ steps in total. The total time to solve each linear program is therefore $\mathcal{O}(k^2r + kD(r))$. Finally, the total time of the algorithm is $\mathcal{O}(n^2r + nD(r))$. □

In the next subsections, we improve the time complexity of the BOUNDREP problem with prescribed ◀ to $\mathcal{O}(n^2 + nD(r))$. Our algorithm makes use of several structural properties of the set of all representations. We note that structural properties of the polyhedron of our linear program, in the case where all lower bounds equal zero and there are no upper bounds, have been considered in several papers in the context of semiorders [29, 30].

### *5.2. The Partially Ordered Set* $\mathfrak{Rep}$

Let the graph $G$ in consideration be a connected unit interval graph. We study structural properties of its representations. Suppose that we fix one of the two partial left-to-right orders $<$ of the intervals from Section 3, so that only indistinguishable vertices are incomparable. We also fix some positive $\varepsilon = \frac{1}{K}$. For most of this section, we work just with lower bounds and completely ignore upper bounds.

We define $\mathfrak{Rep}$ as the set of all $\varepsilon$-grid representations satisfying the lower bounds and in some left-to-right ordering that extends $<$. We define a very natural partial ordering $\leq$ on $\mathfrak{Rep}$: We say that $\mathcal{R} \leq \mathcal{R}'$ if and only if $\ell_i \leq \ell'_i$ for every $v_i \in V(G)$; i.e., $\leq$ is the carthesian ordering of vectors $(\ell_1, \ldots, \ell_n)$. In this section, we study structural properties of the poset $(\mathfrak{Rep}, \leq)$.

If $\varepsilon \leq \frac{1}{n}$, then $\mathfrak{Rep} \neq \emptyset$. The reason is that the graph $G$ is a unit interval graph, and thus there always exists an $\varepsilon$-grid representation $\mathcal{R}$ far to the right satisfying the lower bound contraints.

**The Semilattice Structure.** Let us assume that $\mathrm{lbound}(v_i) > -\infty$ for some $v_i \in V(G)$. Let $S$ be a subset of $\mathfrak{Rep}$. The infimum $\inf(S)$ is the greatest representation $\mathcal{R} \in \mathfrak{Rep}$ such that $\mathcal{R} \leq \mathcal{R}'$ for every $\mathcal{R}' \in S$. In a general poset, infimums may not exist, but if they exist, they are always unique. For $\mathfrak{Rep}$, we show:

**Lemma 5.5.** *Every non-empty $S \subseteq \mathfrak{Rep}$ has an infimum* $\inf(S)$.

PROOF. We construct the requested infimum $\mathcal{R}$ as follows:

$$\ell_i = \min\{\ell'_i : \mathcal{R}' \in S\}, \qquad \forall v_i \in V(G).$$

Notice that the positions in $\mathcal{R}$ are well-defined, since the position of each interval in each $\mathcal{R}'$ is bounded and always on the $\varepsilon$-grid. Clearly, if $\mathcal{R}$ is a correct representation, it is the infimum $\inf(S)$. It remains to show that $\mathcal{R} \in \mathfrak{Rep}$.

Clearly, all positions in $\mathcal{R}$ belong to the $\varepsilon$-grid and satisfy the lower bound constraints. Let $v_i$ and $v_j$ be two vertices. The values $\ell_i$ and $\ell_j$ in $\mathcal{R}$ are given by two representations $\mathcal{R}_1, \mathcal{R}_2 \in S$, that is, $\ell_i = \ell_i^1$ and $\ell_j = \ell_j^2$. Notice that the left-to-right order in $\mathcal{R}$ has to extend $<$: If $v_i < v_j$, then $\ell_i = \ell_i^1 \leq \ell_i^2 < \ell_j^2 = \ell_j$, since $\mathcal{R}_1$ minimizes the position of $v_i$ and the left-to-right order in $\mathcal{R}_2$ extends $<$. Concerning correctness of the representation of the pair $v_i$ and $v_j$, we suppose that $\ell_i = \ell_i^1 \leq \ell_j^2 = \ell_j$; otherwise we swap $v_i$ and $v_j$.

- First we suppose that $v_iv_j \in E(G)$. Then $\ell_j^2 \leq \ell_j^1$, since $\mathcal{R}_2$ minimizes the position of $v_j$. Since $\mathcal{R}_1$ is a correct representation, $\ell_j^1 - 1 \leq \ell_i^1$. So $\ell_j - 1 \leq \ell_i \leq \ell_j$, and the intervals $v_1$ and $v_2$ intersect.
- The other case is when $v_iv_j \notin E(G)$. Then $\ell_i^1 \leq \ell_i^2 \leq \ell_j^2 - 1 - \varepsilon$, since $\mathcal{R}_1$ minimizes the position of $v_i$, $\mathcal{R}_2$ is a correct representation and $v_i < v_j$ in both representations. So $v_i$ and $v_j$ do not intersect in $\mathcal{R}$ as requested.

Consequently, $\mathcal{R}$ represents correctly each pair $v_i$ and $v_j$, and hence $\mathcal{R} \in \mathfrak{Rep}$. □

A poset is a *(meet)-semilattice* if every pair of elements $a, b$ has an infimum $\inf(\{a, b\})$. Lemma 5.5 shows that the poset $(\mathfrak{Rep}, \leq)$ forms a (meet)-semilattice. Similarly as $\mathfrak{Rep}$, we could consider the poset set of all ($\varepsilon$-grid) representations satisfying both the lower and the upper bounds. The structure of this poset is a *complete lattice*, since all subsets have infimums and supremums. Lattices and semilattices are frequently studied, and posets that are lattices satisfy very strong algebraic properties.

**The Left-most Representation.** We are interested in a specific representation in $\mathfrak{Rep}$, called the *left-most representation*. An $\varepsilon$-grid representation $\mathcal{R} \in \mathfrak{Rep}$ is the left-most representation if $\mathcal{R} \leq \mathcal{R}'$ for every $\mathcal{R}' \in \mathfrak{Rep}$; so the left-most representation is left-most in each interval at the same time. We note that the notion of the left-most representation does not make sense if we consider general representations (not on the $\varepsilon$-grid). The left-most representation is the infimum $\inf(\mathfrak{Rep})$, and thus by Lemma 5.5 we get:

**Corollary 5.6.** *The left-most representation always exists and it is unique.*

There are two algorithmic motivations for studying left-most representations. First, in the linear program of Section 5.1 we need to find a representation minimizing $E_t$. Clearly, the left-most representation is minimizing $E_t$ and in addition it is minimizing the rest of the endpoints as well. The second motivation is that we want to construct a representation satisfying the upper bounds as well, so it seems reasonable to try to place every interval as far to the left as possible. The left-most representation is indeed a good candidate for a bounded representation:

**Lemma 5.7.** *There exists a representation $\mathcal{R}'$ satisfying both lower and upper bound constraints if and only if the left-most representation $\mathcal{R}$ satisfies the upper bound constraints.*

Proof. Since $\mathcal{R} \in \mathfrak{Rep}$, it satisfies the lower bounds. If $\mathcal{R}$ satisfies the upper bound constraints, it is a bounded representation. On the other hand, let $\mathcal{R}'$ be a bounded representation. Then

$$\mathrm{lbound}(v_i) \leq \ell_i \leq \ell'_i \leq \mathrm{ubound}(v_i), \qquad \forall v_i \in V(G),$$

and the left-most representation is also a bounded representation. □

*5.3. Why Left-most Representations Cannot Be Easily Constructed by Iterations?*

A very natural idea for an algorithm is to construct the left-most representation iteratively, by adding the vertices $v_1, \ldots, v_n$ one by one and recomputing the left-most representation in each step. In this section, we describe why this natural algorithm does not run in quadratic time. More precisely, we do not claim that it is not possible to implement it in quadratic time or faster using some additional tricks and structural results, but we did not succeeded in this matter.

**The Iterative Algorithm.** Let $G$ be a connected unit interval graph, and let $<$ be the left-to-right partial ordering of its vertices $v_1, \ldots, v_n$ numbered from left to right. We denote by $G_k$ the graph induced by $\{v_1, \ldots, v_k\}$. Let $\mathcal{R}_k$ be the left-most representation of $G_k$, and let $\ell_i^k$ be the position of the left endpoint of $v_i$ in $\mathcal{R}_k$. The iterative algorithm runs as follows.

We initiate $\mathcal{R}_1$ with $\ell_1^1 = \mathrm{lbound}(v_1)$. To compute $\mathcal{R}_k$ from $\mathcal{R}_{k-1}$, we first put $\ell_i^k := \ell_i^{k-1}$ for all $1 \leq i \leq k-1$, and $\ell_k^k := \max\{\mathrm{lbound}(v_k), \ell_j^k + 1 + \varepsilon\}$ where $v_j$ is the rightmost placed non-neighbor of $v_k$. Since $\mathcal{R}_k$ is not likely a correct representation of $G_k$, we proceed by a series of *fixes* till we obtain a correct representation:

- If $v_i v_j \in E(G_k)$, $i < j$, and $\ell_i^k < \ell_j^k - 1$, we *fix* $\mathcal{R}_k$ by setting $\ell_i^k := \ell_j^k - 1$.
- If $v_i v_j \notin E(G_k)$, $i < j$, and $\ell_i^k \geq \ell_j^k - 1$, we *fix* $\mathcal{R}_k$ by setting $\ell_j^k := \ell_i^k + 1 + \varepsilon$.

**Correctness.** We start by proving that the above algorithm is correct.

**Proposition 5.8.** *The above iterative algorithm stops after finite number of steps and outputs the left-most representation $\mathcal{R}$.*

PROOF. It is just sufficient to show that it constructs the left-most representation $\mathcal{R}_k$ from the left-most representation $\mathcal{R}_{k-1}$, and the rest is true by induction. Let $\mathcal{R}_k^s$ be a vector of positions created by the algorithm after $s$ fixes, so $\mathcal{R}_k^s$ might not be a correct representation. We prove by induction according to $s$ that $\mathcal{R}_k^s \leq \mathcal{R}_k$.

Since $\mathcal{R}_{k-1}$ is the left-most representation of $G_{k-1}$, we get $\mathcal{R}_{k-1} \leq \mathcal{R}_k|_{G_{k-1}}$. We initiate $\ell_k^k$ as far to the left as possible, and thus $\mathcal{R}_k^0 \leq \mathcal{R}_k$. Now let $\mathcal{R}_k^{s-1} \leq \mathcal{R}_k$. Then we easily get $\mathcal{R}_k^s \leq \mathcal{R}_k$ since the fix of $(v_i, v_j)$ shifts one of them as little to the right as necessary; since $\mathcal{R}_k$ is a correct representation, it clearly cannot have the shifted interval more to the left than $\mathcal{R}_k^s$.

Since each fix strictly increases the position of one interval and according to Corrolary 5.6 the left-most representation $\mathcal{R}_k$ always exists, we cannot apply fixes indefinitely and the algorithm outputs some correct representation $\mathcal{R}_k^s$. Since $\mathcal{R}_k^s \leq \mathcal{R}_k$, we get $\mathcal{R}_k^s = \mathcal{R}_k$. □

**Unclear Complexity.** Even though the above algorithm is correct, it is not even clear that its complexity is polynomial in $n$ and does not depend on $\varepsilon$. We did not try to further estimate this complexity but it seems one could bound the number of fixes in each iteration by something like $\mathcal{O}(n^2)$ which would give a cubic-time algorithm. The reason why this does not give a quadratic-time algorithm is that the position of each interval can be updated by multiple fixes. We always shift as little as possible, and not as much as it is required by the structure of the graph. Furthermore, we simplified our analysis by assuming that we can locate a wrongly represented pair $(v_i, v_j)$ in constant time, and that we compute on the arithmetic machine (so we ignored numerical issues with small values of $\varepsilon$).

Nevertheless, we believe that the complexity of this algorithm could be improved which might lead to a different quadratic-time (or potentially even linear-time) algorithm for the bounded representation problem with prescribed ordering $\lhd$. As a good starting point, we suggest that one should get a good structural understand how much $\mathcal{R}_k$ differs from $\mathcal{R}_{k-1}$. Even through we give some additional properties concerning the left-most representation, we still do not fully understand its structure. Therefore we derived a different algorithm based on shifting which we describe in the rest of Section 5.

### *5.4. Left-Shifting of Intervals*

Suppose that we construct some initial $\varepsilon$-grid representation that is not the left-most representation. We want to transform this initial representation in $\mathfrak{Rep}$ into the left-most representation of $\mathfrak{Rep}$ by applying a sequence of the following simple operations called the *left-shifting*. The left-shifting operation shifts one interval of the representations by $\varepsilon$ to the left such that this shift maintains the correctness of the representation; for an example see Figure 9a. The main goal of this section is to prove that by left-shifting we can always produce the left-most representation.

**Proposition 5.9.** *For $\varepsilon = \frac{1}{K}$ and $K \geq \frac{n}{2}$, an $\varepsilon$-grid representation $\mathcal{R} \in \mathfrak{Rep}$ is the left-most representation if and only if it is not possible to shift any single interval to the left by $\varepsilon$ while maintaining correctness of the representation.*

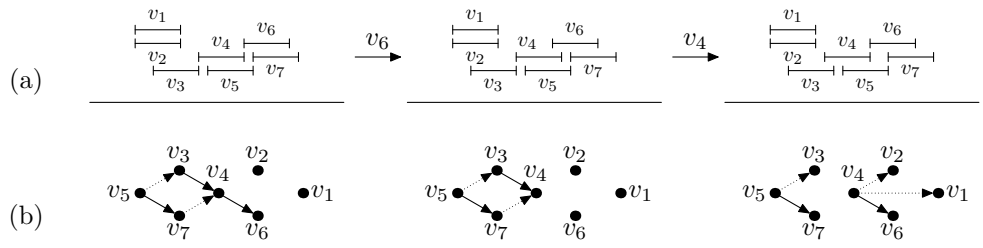


Figure 9: (a) A representation modified by left-shifting of $v_6$ and $v_4$. (b) The corresponding obstruction digraphs $H$ for each of the representations. Only sinks of the obstruction digraphs can be left-shifted.

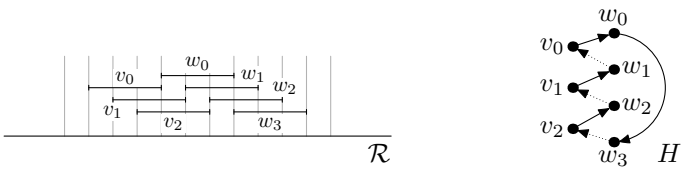


Figure 10: An $\varepsilon$-grid representation for $\varepsilon = \frac{1}{3}$ on the left and the obstruction digraph $H$ containing a cycle on the right.

Before proving the proposition, we describe some additional combinatorial structure of left-shifting. An interval $v_i$ is called *fixed* if it is in the left-most position and cannot ever be shifted more to the left, i.e., $\ell_i = \min\{\ell'_i : \mathcal{R}' \in \mathfrak{Rep}\}$. For example, an interval $v_i$ is fixed if $\ell_i = \text{lbound}(v_i)$. A representation is the left-most representation if and only if every interval is fixed.

**Obstruction Digraph.** An interval $v_i$, having $\ell_i > \text{lbound}(v_i)$, can be left-shifted if it does not make the representation incorrect, and the incorrectness can be obtained in two ways. First, there could be some interval $v_j$, $v_j \lhd v_i$ such that $v_iv_j \notin E(G)$ and $\ell_j + 1 + \varepsilon = \ell_i$; we call $v_j$ a *left obstruction* of $v_i$. Second, there could be some interval $v_j$, $v_i \lhd v_j$ such that $v_iv_j \in E(G)$ and $\ell_i + 1 = \ell_j$ (so $v_i$ and $v_j$ are touching); then we call $v_j$ a *right obstruction* of $v_i$. In both cases, we first need to move $v_j$ before moving $v_i$.

For the current representation $\mathcal{R}$, we define the *obstruction digraph* $H$ on the vertices of $G$ as follows. We put $V(H) = V(G)$ and $(v_i, v_j) \in E(H)$ if and only if $v_j$ is an obstruction of $v_i$. For an edge $(v_i, v_j)$, if $v_j \lhd v_i$, we call it a *left edge*; if $v_i \lhd v_j$, we call it a *right edge*. As we apply left-shifting, the structure of $H$ changes; see Figure 9b.

**Lemma 5.10.** *An interval $v_i$ is fixed if and only if there exists a directed path in $H$ from $v_i$ to $v_j$ such that $\ell_j = \text{lbound}(v_j)$.*

Proof. Suppose that $v_i$ is connected to $v_j$ by a path in $H$. By the definition of $H$, $v_xv_y \in E(H)$ implies that $v_y$ has to be shifted before $v_x$. Thus $v_j$ has to be shifted before moving $v_i$ which is not possible since $\ell_j = \text{lbound}(v_j)$.

On the other hand, suppose that $v_i$ is fixed, i.e., $\ell_i = \inf\{\ell'_i : \forall\mathcal{R}'\}$. Let $H'$ be the induced subgraph of $H$ of the vertices $v_j$ such that there exists a directed path from $v_i$ to $v_j$. If for all $v_j \in H'$, $\ell_j > \text{lbound}(v_j)$, we can shift all vertices of $H'$ by $\varepsilon$ to the left which constructs a correct representation and contradicts that $v_i$ is fixed. Therefore, there exists $v_j \in H'$ having $\ell_j = \text{lbound}(v_j)$ as requested. □

For example in Figure 9 on the left, if $\ell_4 = \text{lbound}(v_4)$, then the intervals $v_3$, $v_4$, $v_5$ and $v_7$ are fixed. Also, we can prove:

**Lemma 5.11.** *If $\varepsilon = \frac{1}{K}$ and $K \geq \frac{n}{2}$, the obstruction digraph $H$ is acyclic.*

Proof. Suppose for contradiction that $H$ contains some cycle $u_1, \ldots, u_c$. This cycle contains $a$ left edges and $b$ right edges. Recall that if $(u_i, u_{i+1})$ is a left edge, then $\ell_{u_{i+1}} = \ell_{u_i} - 1 - \varepsilon$, and if it is a right edge, $\ell_{u_{i+1}} = \ell_{u_i} + 1$ (and similarly for $(u_c, u_1)$). If we go along the cycle from $u_1$ to $u_1$, the initial and the final positions have to be the same. Therefore $a(1 + \varepsilon) = b$.

Now if this equation holds, then $a$ has to be a multiple of $K$. Therefore $a \geq K$ and $b \geq K + 1$, and thus $n \geq c = a + b \geq 2K + 1$ which is not possible. □

We note that the assumption $K \geq \frac{n}{2}$ is necessary and tight. For every $\varepsilon = \frac{1}{K}$, there exists a representation of a graph with $2K + 1$ vertices having a cycle in $H$. The graph contains two cliques $v_0, \ldots, v_{K-1}$ and $w_0, \ldots, w_K$ such that $v_i$ is also adjacent to $w_0, \ldots, w_i$. Then the assignment $\ell_{v_0} = 0$, $\ell_{v_i} = \ell_{v_0} + i\varepsilon$ and $\ell_{w_i} = \ell_{v_0} + 1 + i\varepsilon$ is a correct representation. Observe that $H$ contains a cycle $w_kv_{k-1}w_{k-1}v_{k-2}w_{k-2}\ldots v_1w_1v_0w_0w_k$. See Figure 10 for $K = 3$.

**Predecessors of Poset $\mathfrak{Rep}$.** A representation $\mathcal{R}' \in \mathfrak{Rep}$ is a *predecessor* of $\mathcal{R} \in \mathfrak{Rep}$ if $\mathcal{R}' < \mathcal{R}$ and there is no representation $\bar{\mathcal{R}} \in \mathfrak{Rep}$ such that $\mathcal{R}' < \bar{\mathcal{R}} < \mathcal{R}$. We denote the predecessor relation by $\prec$. In a general

poset, predecessors may not exist. But they always exist for a poset of a discrete structure like $(\mathfrak{Rep}, \leq)$: Indeed, there are finitely many representations $\bar{\mathcal{R}}$ between any $\mathcal{R}' < \mathcal{R}$, and thus the predecessors always exist. Also, for any two representations $\mathcal{R}' < \mathcal{R}$, there exists a finite *chain* of predecessors $\mathcal{R}' = \mathcal{R}_0 \prec \mathcal{R}_1 \prec \cdots \prec \mathcal{R}_k = \mathcal{R}$.

For the poset $(\mathfrak{Rep}, \leq)$, we are able to fully describe the predecessor structure:

**Lemma 5.12.** *For $\varepsilon = \frac{1}{K}$ and $K \geq \frac{n}{2}$, the representation $\mathcal{R}'$ is a predecessor of $\mathcal{R}$ if and only if $\mathcal{R}'$ is obtained from $\mathcal{R}$ by applying one left-shifting operation.*

Proof. Clearly, if $\mathcal{R}'$ is obtained from $\mathcal{R}$ by one left-shifting, it is a predecessor of $\mathcal{R}$.

On the other hand, suppose we have $\mathcal{R}' < \mathcal{R}$. Let $H$ be the obstruction digraph of $\mathcal{R}$ and $\bar{H}$ be the subgraph of $H$ induced by the intervals having different positions in $\mathcal{R}$ and $\mathcal{R}'$. Then there are no directed edges from $\bar{H}$ to $H \setminus \bar{H}$ (otherwise $\mathcal{R}'$ would be an incorrect representation). According to Lemma 5.11, the digraph $\bar{H}$ is acyclic. Therefore, it contains at least one sink $v_i$. By left-shifting $v_i$ in $\mathcal{R}$, we create a correct representation $\bar{\mathcal{R}} \in \mathfrak{Rep}$. Clearly, $\mathcal{R}' \leq \bar{\mathcal{R}} \prec \mathcal{R}$, and so $\mathcal{R}'$ is a predecessor of $\mathcal{R}$ if and only if $\mathcal{R}' = \bar{\mathcal{R}}$. □

Again, the assumption on the value of $\varepsilon$ is necessary. For example in Figure 10, the structure of $\mathfrak{Rep}$ is just a single chain where a predecessor of some representation is obtained by shifting all intervals by $\varepsilon$ to the left.

**Proof of Left-shifting Proposition.** The main proposition of this subsection is a simple corollary of Lemma 5.12.

Proof (Proposition 5.9). The left-most representation $\mathcal{R}$ is $\inf(\mathfrak{Rep})$, so it has no predecessors and nothing can be left-shifted. On the other hand, if $\inf(\mathfrak{Rep}) < \mathcal{R}$, there is a chain of predecessors in between which implies using Lemma 5.12 that it is possible to left-shift some interval. □

### *5.5. Preliminaries for the Shifting Algorithm*

Before describing the shifting algorithm, we present several results which simplify the graph and the description of the algorithm.

**Pruned Graph.** The obstruction digraph $H$ may contain many edges since each vertex $v_i$ can have many obstructions. But if $v_i$ has many, say, left obstructions, these obstructions have to be positioned the same. If two intervals $u$ and $v$ have the same position in a correct unit interval representation, then $N[u] = N[v]$ and they are indistinguishable. Our goal is to construct a *pruned graph* $G'$ which replaces each group of indistinguishable vertices of $G$ by a single vertex. This construction is not completely straightforward since indistinguishable vertices may have different lower and upper bounds.

Let $\{\Gamma_1, \ldots, \Gamma_k\}$ be the partitioning of $V(G)$ by the groups of indistinguishable vertices (and the groups are ordered by $\lhd$ from left to right). We construct a unit interval graph $G'$, where the vertices are $\gamma_1, \ldots, \gamma_k$ with $\mathrm{lbound}(\gamma_i) = \max\{\mathrm{lbound}(v_j) : v_j \in \Gamma_i\}$, and the edges $E(G')$ correspond to the edges between the groups of $G$.

Suppose that we have the left-most representation $\mathcal{R}'$ of the pruned graph $G'$ and we want to construct the left-most representation $\mathcal{R}$ of $G$. Let $\Gamma_\ell$ be a group. We place each interval $v_i \in \Gamma_\ell$ as follows. Let $\gamma^\ell_{\leftarrow}$ be the first non-neighbor of $\gamma_\ell$ on the left and $\gamma^\ell_{\rightarrow}$ be the right-most neighbor of $\gamma_\ell$ (possibly $\gamma^\ell_{\rightarrow} = \gamma_\ell$). We set

$$\ell_i = \max\{\mathrm{lbound}(v_i), \ell_{\gamma^\ell_{\leftarrow}} + 1 + \varepsilon, \ell_{\gamma^\ell_{\rightarrow}} - 1\}, \tag{7}$$

and if $\gamma^\ell_{\leftarrow}$ does not exist, we ignore it in max. The meaning of this formula is to place each interval as far to the left as possible while maintaining the structure of $\mathcal{R}'$. Figure 11 contains an example of the construction of $\mathcal{R}$.

Before proving correctness of the construction of $\mathcal{R}$, we show two general properties of the formula (7). The first lemma states that each interval $v_i \in \Gamma_\ell$ is not placed in $\mathcal{R}$ too far from the position of $\gamma_\ell$ is $\mathcal{R}'$.

**Lemma 5.13.** *For each $v_i \in \Gamma_\ell$, it holds*

$$\ell_{\gamma_\ell} - 1 \leq \ell_i \leq \ell_{\gamma_\ell}. \tag{8}$$

PROOF. The first inequality is true since $\ell_{\gamma_\ell} - 1 \leq \ell_{\gamma^\ell_\rightarrow} - 1 \leq \ell_i$ holds according to (7) and the ordering $\lhd$ for $\mathcal{R}'$. The second inequality holds since $\mathcal{R}'$ is a correct bounded representation, and so $\ell_{\gamma_\ell}$ is greater than or equal to each term in (7). □

The second lemma states that the representations $\mathcal{R}$ and $\mathcal{R}'$ are *intertwining* each other. If $\mathcal{R}$ is drawn on top of $\mathcal{R}'$, then the vertices of each group $\Gamma_\ell$ are in between of $\gamma_{\ell-1}$ and $\gamma_\ell$; see Figure 11.

**Lemma 5.14.** *For each $v_i \in \Gamma_\ell$ and $\ell > 1$, it holds*

$$\ell_{\gamma_{\ell-1}} < \ell_i \leq \ell_{\gamma_\ell}, \tag{9}$$

PROOF. The second inequality holds by (8). For the first inequality, there are two possible cases why the groups $\Gamma_{\ell-1}$ and $\Gamma_\ell$ are distinct:

- The first case is when $\gamma^\ell_\leftarrow$ is a neighbor of $\gamma_{\ell-1}$. Then $\ell_{\gamma_{\ell-1}} \leq \ell_{\gamma^\ell_\leftarrow} + 1 < \ell_i$; the first inequality holds since $\gamma^\ell_\leftarrow \gamma_{\ell-1} \in E(G')$ and $\mathcal{R}'$ is a correct representation, and the second inequality is given by (7).
- The second case is when $\gamma^\ell_\rightarrow$ is a non-neighbor of $\gamma_{\ell-1}$. Then $\ell_{\gamma_{\ell-1}} < \ell_{\gamma^\ell_\rightarrow} - 1 \leq \ell_i$ by the fact that $\gamma_{\ell-1}\gamma^\ell_\rightarrow \notin E(G')$ and by (7).

In both cases, we get $\ell_{\gamma_{\ell-1}} < \ell_i$. □

Now, we are ready to show correctness of the construction of $\mathcal{R}$.

**Proposition 5.15.** *From the left-most representation $\mathcal{R}'$ of the pruned graph $G'$, we can construct the correct left-most representation $\mathcal{R}$ of $G$ by placing the intervals according to (7).*

PROOF. We argue the correctness of the representation $\mathcal{R}$. Let $v_i$ and $v_j$ be a pair of vertices of $G$. Let $v_iv_j \in E(G)$. If $v_i$ and $v_j$ belong to the same group $\Gamma_\ell$, they intersect each other at position $\ell_{\gamma_\ell}$ by (8). Otherwise let $v_i \in \Gamma_\ell$ and $v_j \in \Gamma_{\ell'}$, and assume that $\Gamma_\ell < \Gamma_{\ell'}$. Then $\ell_i \leq \ell_{\gamma_\ell} \leq \ell_j$ by the intertwining property (9). Also, $\ell_j \leq \ell_{\gamma_{\ell'}} \leq \ell_{\gamma^\ell_\rightarrow} \leq \ell_i + 1$ since $\gamma_{\ell'}$ is a right neighbor of $\gamma_\ell$ and (8). Therefore, $\ell_i \leq \ell_j \leq \ell_i + 1$ and $v_i$ intersects $v_j$ in $\mathcal{R}$. Now, let $v_iv_j \notin E(G)$, $v_i \in \Gamma_\ell$, $v_j \in \Gamma_{\ell'}$ and $v_i < v_j$. Then $\ell_i \leq \ell_{\gamma_\ell} \leq \ell_{\gamma^{\ell'}_\leftarrow} \leq \ell_j - 1 - \varepsilon$ by (7) and (8), so $v_i$ and $v_j$ do not intersect. So the assignment $\mathcal{R}$ is a correct representation of $G$.

It remains to show that $\mathcal{R}$ is the left-most representation of $G$. We can identify each $\gamma_\ell$ with one interval $v_i \in \Gamma_\ell$ having $\text{lbound}(v_i) = \text{lbound}(\gamma_\ell)$; for an example see Figure 11. So $G'$ can be viewed as an induced subgraph of $G$. We want to show that the intervals of $G'$ are represented in $\mathcal{R}$ exactly the same as in $\mathcal{R}'$. Since $\mathcal{R}|_{G'}$ (which denotes $\mathcal{R}$ restricted to $G'$) is some representation of $G'$ and $\mathcal{R}'$ is the left-most representation of $G'$, we get $\ell'_{\gamma_\ell} \leq \ell_{\gamma_\ell}$ for every $\gamma_\ell$. By (8), we get $\ell'_{\gamma_\ell} = \ell_{\gamma_\ell}$.

We know that $\mathcal{R}|_{G'}$ is the left-most representation, or in other words each interval of $G'$ is fixed in $\mathcal{R}$. The rest of the intervals are placed so that they are either trivially fixed by $\ell_i = \text{lbound}(v_i)$, or they have as obstructions some fixed intervals from $G'$, in which case they are fixed by Lemma 5.10. Therefore, every interval of $G$ is fixed and $\mathcal{R}$ is the left-most representation. □

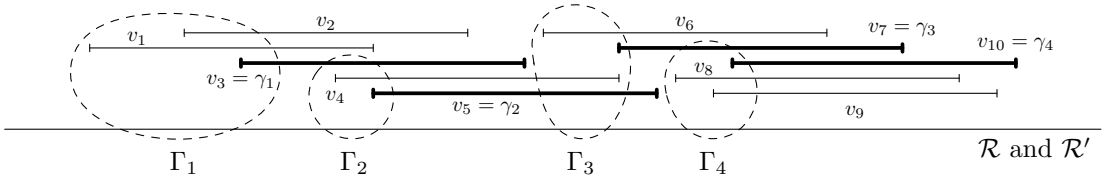


Figure 11: Both representations $\mathcal{R}$ and $\mathcal{R}'$ in one figure, with the intervals of $\mathcal{R}'$ depicted in bold. The left endpoints of the intervals of each group are enclosed by dashed curves, and these curves are ordered from left to right according to $<$.

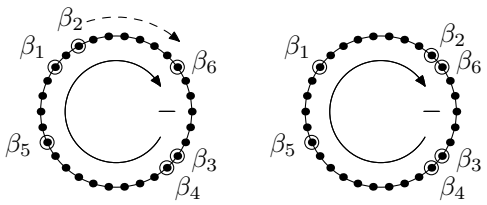


Figure 12: Examples of position cycles. In the cycle on the left, we can shift $\beta_2$ in the clockwise direction towards $\beta_6$, which gives a new representation whose position cycle is depicted on the right. We note that after this left-shifting, $v_6$ is not necessarily an obstruction of $v_2$.

For the pruned graph $G'$, the obstruction digraph $H$ has in- and out-degree at most two. Each interval has at most one left obstruction and at most one right obstruction, and these obstructions are always the same intervals. More precisely, if $v_j$ is a left obstruction of $v_i$, then $v_j = v^i_{\leftarrow}$, whereas if $v_j$ is a right obstruction of $v_i$, then $v_j = v^i_{\rightarrow}$.

The pruning operation can be done in time $\mathcal{O}(n+m)$, so we may assume that our graph $G$ is already pruned and contains no indistinguishable vertices. And the structure of obstructions in $G$ can be computed in time $\mathcal{O}(n+m)$ as well.

**Position Cycle.** For each interval in some $\varepsilon$-grid representation, we can write its position in this form:

$$\ell_i = \alpha_i + \beta_i\varepsilon, \qquad \alpha_i \in \mathbb{Z},\ \beta_i \in \mathbb{Z}_K, \tag{10}$$

where $\varepsilon = \frac{1}{K}$. In other words, $\alpha_i$ is the integer position of $v_i$ in the grid and $\beta_i$ describes how far is this interval from this integer position.

Concerning left-shifting, the values $\beta_i$ are more important. We can depict $\mathbb{Z}_K = \{0, \ldots, K-1\}$ as a cycle with $K$ vertices where the value decreases clockwise. The value $\beta_i$ assigns to each interval $v_i$ one vertex of the cycle. The cycle $\mathbb{Z}_K$ together with marked positions of $\beta_i$'s is called the *position cycle*. A vertex of the position cycle is called *taken* if some $\beta_i$ is assigned to it, and *empty* otherwise. The position cycle allows us to visualize and work with left-shifting very intuitively. When an interval $v_i$ is left-shifted, $\beta_i$ cyclically decreases by one, so $\beta_i$ moves clockwise along the cycle. For an illustration, see Figure 12.

If $(v_i, v_j)$ is a left edge of $H$, then $\beta_j = \beta_i - 1$, and if $(v_i, v_j)$ is a right edge, then $\beta_i = \beta_j$. So if $v_j$ is an obstruction of $v_i$, $\beta_j$ has to be very close to $\beta_i$ (either at the same position or at the next clockwise position). If there is a big empty space in the clockwise direction from $\beta_i$, the interval $v_i$ can be left-shifted many times (or till it becomes fixed by $\ell_i = \mathrm{lbound}(v_i)$). Notice that if $\beta_i$ is very close to $\beta_j$, it does not mean that $\ell_i$ is very close to $\ell_j$ because the values $\alpha_i$ and $\alpha_j$ are ignored in the position cycle.

### *5.6. The Shifting Algorithm for* BoundRep

We want to solve an instance of BoundRep with a prescribed ordering $\blacktriangleleft$. We work with an $\varepsilon$-grid which is different from the one in Section 4.1. The new value of $\varepsilon$ is the value given by (1) refined $n$ times, so

$$\varepsilon = \frac{1}{n^2} \cdot \varepsilon'.$$

Lemma 4.1 applies for this value of $\varepsilon$ as well, so if the instance is solvable, there exists a solution which is on this $\varepsilon$-grid.

The algorithm works exactly as the algorithm of Subsection 5.1. The only difference is that for a component with $k$ vertices we can solve the linear program in time $\mathcal{O}(k^2 + kD(r))$, and now we describe how to do it. We assume that the input component is already pruned, otherwise we prune it and use Proposition 5.15 to complete the representation. We expect that the left-to-right order $\lhd$ of the vertices is given. The algorithm requires time $\mathcal{O}(kD(r))$ since the bounds are given in the form $\frac{p_i}{q_i}$ and we need to perform arithmetic operations with these bounds. Therefore the total complexity of the algorithm for the BoundRep problem is $\mathcal{O}(n^2 + nD(r))$.

**Overview.** The algorithm for solving one linear program works in three basic steps:

(1) We construct an initial $\varepsilon$-grid representation (in the ordering $\lhd$) having $\ell_i \geq \text{lbound}(v_i)$ for all intervals, using the algorithm of Corneil et al. [26].
(2) We shift the intervals to the left while maintaining correctness of the representation until the left-most representation is constructed, using Proposition 5.9.
(3) We check whether the left-most representation satisfies the upper bounds. If so, we have the left-most representation satisfying all bound constraints. This representation solves the linear program of Subsection 5.1 and minimizes $E_t$. Otherwise, the left-most representation does not satisfy the upper bound constraints. Thus by Lemma 5.7 no representation satisfies the upper bound constraints, and the linear program has no solution.

**Input Size.** Let $r$ be the size of the input describing bound constraints. A standard complexity assumption is that we can operate with polynomially large numbers (having $\mathcal{O}(\log r)$ bits in binary) in constant time, to avoid the extra factor $\mathcal{O}(\log r)$ in the complexity of most of the algorithms. However, the value of $\varepsilon$ given by (1) might require $\mathcal{O}(r)$ digits when written in binary. The assumption that we can computate with numbers having $\mathcal{O}(r)$ digits in contant time would break most of the computational models. Therefore, our computational model requires a larger time for arithmetic operations with numbers having $\mathcal{O}(r)$ digits in binary. For example, the best known algorithm for multiplication/division on a Turing machine requires time $\mathcal{O}(D(r))$.

The problem is that a straightforward implementation of our algorithm working with the $\varepsilon$-grid would require time $\mathcal{O}(k^2 r^c)$ for some $c$ instead of $\mathcal{O}(k^2 + kD(r))$. There is an easy way out. Instead of computing with *long numbers* having $\mathcal{O}(r)$ digits, we mostly compute with *short numbers* having just $\mathcal{O}(\log r)$ digits. Instead of the $\varepsilon$-grid, we mostly work in a larger $\Delta$-grid where $\Delta = \frac{1}{n^2}$. The algorithm computes with the long numbers only in two places. First, some initial computations concerning the input are performed. Second, when the shifting makes some interval fixed, the algorithm estimes the final $\varepsilon$-grid position of the interval. All these computations can be done in total time $\mathcal{O}(kD(r))$ and we describe everything in detail later.

**Left-Shifting.** The basic operation of the algorithm is the LeftShift procedure which we describe here. We deal separately with fixed and unfixed intervals (and some intervals might be fixed initially). Unfixed intervals are on the $\Delta$-grid and fixed intervals have precise positions calculated on the $\varepsilon$-grid. We place only unfixed intervals on the position cycle for the $\Delta$-grid. At any moment of the algorithm, each vertex of the position cycle is taken by at most one $\beta_i$; this is true for the initial representation and the shifting keeps this property.

We define the procedure LeftShift($v_i$) which shifts $v_i$ from the position $\ell_i$ into a new position $\ell'_i$ such that the representation remains correct. The procedure LeftShift($v_i$) consists of two steps:

(1) Since $v_i$ is unfixed, it has some $\beta_i$ placed on the position cycle. Let $k$ be such that the vertices $\beta_i + 1, \ldots, \beta_i + k$ of the position cycle are empty and the vertex $\beta_i + k + 1$ is taken by some $\beta_b$. Then a candidate for the new position of $v_i$ is $\bar{\ell}_i = \ell_i - k\Delta$.
(2) We need to ensure that this shift from $\ell_i$ to $\bar{\ell}_i$ is valid with respect to $\text{lbound}(v_i)$ and the positions of the fixed intervals. Concerning the lower bound, we cannot shift further than $\text{lbound}(v_i)$. Concerning the fixed intervals, the shift is limited by positions of fixed obstructions of $v_i$. If $v_j$ is a fixed left obstruction, we cannot shift further than $\ell_j + 1 + \varepsilon$, and if $v_{j'}$ a fixed right obstruction, we cannot shift further than $\ell_{j'} - 1$.

The resulting position after applying LeftShift($v_i$) is

$$\ell'_i = \max\{\bar{\ell}_i, \text{lbound}(v_i), \ell_j + 1 + \varepsilon, \ell_{j'} - 1\}. \tag{11}$$

**Lemma 5.16.** *If the original representation $\mathcal{R}$ is correct, than the* LeftShift($v_i$) *procedure produces a correct representation $\mathcal{R}'$.*

Proof. Clearly, the lower bound for $v_i$ is satisfied in $\mathcal{R}'$. The shift of $v_i$ from $\ell_i$ to $\ell'_i$ can be viewed as a repeated application of the left-shifting operation from Section 5.4. We just need to argue that each left-shifting operation can be applied till the position $\ell'_i$ is reached.

If at some point, the left-shifting operation could not be applied, there would have to be some obstruction $v_j$ of $v_i$. There is no unfixed obstruction since all vertices of the position cycle $\beta_i + 1, \ldots, \beta_i + k$ are empty. And $v_j$ cannot be fixed as well since we check positions of both possible obstructions. So there is no obstruction $v_j$. Therefore, by repeated applying the left-shifting operation, the interval $v_i$ gets at a position $\ell'_i$ and the resulting representation is correct. □

After LeftShift($v_i$), if $\bar{\ell}_i$ is not a strict maximum of the four terms in (11), the interval $v_i$ becomes fixed; either trivially since $\ell'_i = \text{lbound}(v_i)$, or by Lemma 5.10 since $v_i$ becomes obstructed by some fixed interval. In such a case, we remove $\beta_i$ from the position cycle.

**Fast Implementation of Left-Shifting.** Since we apply the LeftShift procedure repeatedly, we want to implement it in time $\mathcal{O}(1)$. Considering the terms in (11), the first term $\bar{\ell}_i$ is a short number (on the $\Delta$-grid) and the remaining terms are long numbers (on the $\varepsilon$-grid). We first compare $\bar{\ell}_i$ to the remaining terms which are three comparisons of short and long numbers and we are going to show how to compare them in $\mathcal{O}(1)$. If $\bar{\ell}_i$ is a strict maximum, we use it for $\ell'_i$. Otherwise, we need to compute the maximum of the remaining three terms which takes time $\mathcal{O}(D(r))$. But then the interval $v_i$ becomes fixed, and so this costly step is done exactly $k$ times, and takes the total time $\mathcal{O}(kD(r))$.

**Lemma 5.17.** *With the total precomputation time $\mathcal{O}(kD(r))$, it is possible to compare $\bar{\ell}_i$ to the remaining terms in (11) in time $\mathcal{O}(1)$ per* LeftShift *procedure.*

Proof. Initially, we do the following precomputation for the lower bounds. By the input, we have $b$ lower bounds given in the form $\frac{p_1}{q_1}, \ldots, \frac{p_b}{q_b}$ as irreducible fractions. For each bound, we first compute its position $(\alpha_i, \beta_i)$ on the $\varepsilon$-grid; see (10).

If $\text{lbound}(v_i) \ll \text{lbound}(v_j)$ for some vertices $v_i$ and $v_j$, then $\text{lbound}(v_i)$ is never achieved since the graph is connected and every representation takes space at most $k$. Therefore we can increase $\text{lbound}(v_i)$ without any change in the solution of the instance. More precisely, let $\alpha = \max \alpha_i$. Then we modify each bound by setting $\alpha_i := \max\{\alpha - k - 1, \alpha_i\}$. In addition, we shift all the bounds by substructing a constant $C$ such that each $\alpha_i - C \in [0, k+1]$. Concerning $\beta_i$, we round the position $(\alpha_i, \beta_i)$ down to a position $(\alpha_i, \bar{\beta}_i)$ of the $\Delta$-grid. These precomputations can be done for all lower bounds in time $\mathcal{O}(kD(r))$.

Suppose that we want to find out whether $\bar{\ell}_i \leq \text{lbound}(v_j) = \alpha_j + \beta_j \cdot \varepsilon$ where $\bar{\ell}_i$ is in the $\Delta$-grid. Then it is sufficient to check whether $\bar{\ell}_i \leq \alpha_j + \bar{\beta}_j \Delta$ which can be done in constant time since both $\alpha_j$ and $\bar{\beta}_j$ are short numbers.

When $v_j$ becomes fixed, its precise position is computed using (11). Then we compute the values $\ell_j - 1$ and $\ell_j + 1 + \varepsilon$ used in (11) and round them down to the $\Delta$-grid. Using these precomputed values, $\bar{\ell}_i$ can be compared with the remaining terms in (11) in time $\mathcal{O}(1)$. When an interval becomes fixed, time $\mathcal{O}(D(r))$ is used. Since each interval becomes fixed exactly once, this rounding also takes the total time $\mathcal{O}(kD(r))$. □

Notice that the representation is constructed in a position shifted by $C$. Later, before checking the upper bound, we shift the whole representation back.

**Initial Representation.** Recall that the position cycle has $n^2$ vertices and $\Delta = \frac{1}{n^2}$. The algorithm of Corneil et al. [26] gives a representation in the $\frac{1}{k}$-grid. Using the proof of Lemma 4.1, we construct from it the initial $\Delta$-grid representation. Then we shift it such that $\ell_i \geq \text{lbound}(v_i)$ for each $v_i$ and $\ell_i \leq \text{lbound}(v_i) + \Delta$ for some $v_i$. For this initial representation, each interval can be shifted to the left in total by at most $\mathcal{O}(k)$.

The initial representation obtained from the representation of the algorithm of Corneil et al. [26] places all intervals in such a way that $\beta_i$'s are almost positioned equidistantly in the position cycle; refer to the left-most position cycle in Figure 13. As we say in the description of the LeftShift procedure, we only require that all $\beta_i$'s are placed to pairwise different vertices of the position cycle.

**Shifting Phases.** All shifting of the algorithm is done by repeated application of the LeftShift procedure. Using Lemma 5.16, we know that the representation created in each step is correct. We apply the procedure in such a way that each interval is almost always shifted by almost one. The shifting of unfixed intervals proceeds in two phases:

- *The first phase* creates one big gap by clustering all $\beta_i$'s in one part of the cycle. To do so, we apply the LeftShift procedure to each interval, in the order given by the position cycle. Of course, some intervals might become fixed and disappear from the position cycle. We obtain one big gap of size at least $n(n-1)$. Again, refer to Figure 13.
- *In the second phase,* we use this big gap to shift intervals one by one, which also moves the cluster along the position cycle. Again, if some interval becomes fixed, it is removed from the position cycle. The second phase finishes when each interval becomes fixed and the left-most representation is constructed. For an example, see Figure 14.

**Putting It All Together.** First, we show correctness of the shifting algorithm and its complexity:

**Lemma 5.18.** *For a component having $k$ vertices, the shifting algorithm constructs a correct left-most representation in time $\mathcal{O}(k^2 + kD(r))$.*

Proof. First, we argue correctness of the algorithm. The algorithm starts with an initial representation which is correct and satisfies the lower bounds. By Lemma 5.16, after applying each LeftShift procedure, the resulting representation is still correct. The algorithm keeps a correct list of fixed intervals which is increased by shifting. So after finitely many applications of the LeftShift procedure, every interval becomes fixed, and we obtain the left-most representation.

Concerning complexity, all precomputations take total time $\mathcal{O}(kD(r))$. Using Lemma 5.17, each LeftShift$(v_i)$ procedure can be applied in time $\mathcal{O}(1)$ unless $v_i$ becomes fixed. The first phase is applying the LeftShift procedure $k-1$ times. In the second phase, each interval is shifted by at least $\frac{n-1}{n}$ (unless it becomes fixed). Since each interval can be shifted by at most $\mathcal{O}(k)$ from its initial position, the second phase applies the LeftShift procedure $\mathcal{O}(k^2)$ times. So the total running time of the algorithm is $\mathcal{O}(k^2 + kD(r))$. □

We are ready to prove that BoundRep with a prescribe ordering $\blacktriangleleft$ can be solved in time $\mathcal{O}(n^2+nD(r))$:

Proof (Theorem 1.3). We proceed exactly as in the algorithm of Section 5.1, so we process the components $C_1 \blacktriangleleft \cdots \blacktriangleleft C_c$ from left to right, and for each of them we solve two linear programs. For each linear program, we find the left-most representation using Lemma 5.18, and we test for this representation (shifted back by $C$) whether the upper bounds are satisfied. According to Lemma 5.7, the linear program is solvable if and only if the left-most representation satisfy the upper bounds, and clearly the left-most representation minimizes $E_t$. The time complexity of the algorithm is $\mathcal{O}(n^2+nD(r))$ and the proof of correctness is exactly the same as in Proposition 5.9. □

We finally present an FPT algorithm for BoundRep with respect to the number of components $c$. The algorithm is based on Theorem 1.3.

Proof (Corollary 1.4). There are $c!$ possible left-to-right orderings of the components of $G$. For each of them, we can decide in time $\mathcal{O}(n^2 + nD(r))$ whether there exists a bounded representation in the order, using Theorem 1.3. So the total time necessary is $\mathcal{O}((n^2 + nD(r))c!)$. □

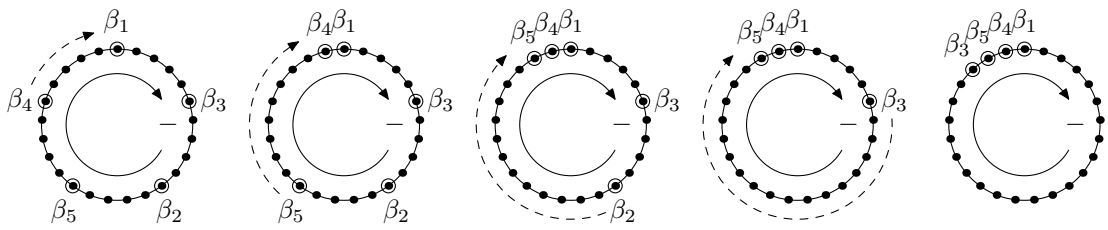


Figure 13: The position cycle during the first phase, changing from left to right. The first phase clusters the $\beta_i$'s by moving $\beta_4$, $\beta_5$, $\beta_2$ and $\beta_3$ towards $\beta_1$. When LeftShift$(v_2)$ is applied, $v_2$ becomes fixed and $\beta_2$ disappears from the position cycle.

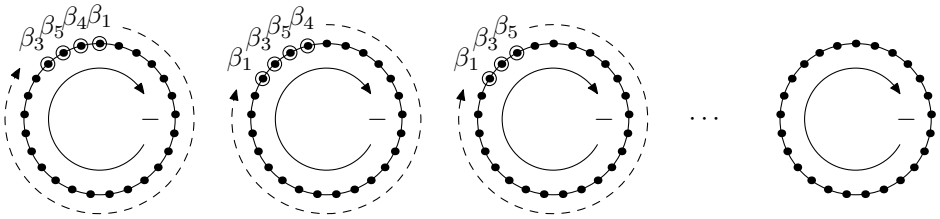


Figure 14: The position cycle during the second phase, changing from left to right. We shift $\beta_i$'s across the big gap till all $\beta_i$'s disappear.

## 6. Extending Unit Interval Graphs

The RepExt(UNIT INT) problem can be solved using Theorem 1.3. We just need to show that it is a particular instance of BoundRep in which the ordering $\blacktriangleleft$ of the components can be derived:

Proof (Theorem 1.5). The graph $G$ contains unlocated components and located components. Similarly to Section 3, unlocated components can be placed far to the right and we can deal with them using a standard recognition algorithm.

Concerning located components $C_1, \ldots, C_c$, they have to be ordered in $\mathcal{R}'$ from left to right, which gives the required ordering $\blacktriangleleft$. We straightforwardly construct the instance of BoundRep with this $\blacktriangleleft$ as follows. For each pre-drawn interval $v_i$ at position $\ell_i$, we put $\mathrm{lbound}(v_i) = \mathrm{ubound}(v_i) = \ell_i$. For the rest of the intervals, we set no bounds. Clearly, this instance of BoundRep is equivalent with the original RepExt(UNIT INT) problem. And we can solve it in time $\mathcal{O}(n^2 + nD(r))$ using Theorem 1.3. □

## 7. Conclusions

**Assumption on the Input.** Almost every graph algorithm is not able to achieve time $\mathcal{O}(n+m)$ if the input is given by an adjacency matrix of the graph. Similarly, to get linear time in Theorem 1.1, we have to assume that the partial representation of a proper interval graph is given in a nice form.

We say that a partial representation is *normalized* if the pre-drawn endpoints have positions $\{1, \ldots, 2n\}$. This assumption is natural since according to Lemma 3.4, the extendibility of a partial representation only depends on the left-to-right order of the pre-drawn intervals and not on the precise positions. For a normalized partial representation, the order $<^{G'}$ can be computed in time $\mathcal{O}(n)$. If the representation is not given in this way, the algorithm needs an additional time $\mathcal{O}(k \log k)$ to construct $<^{G'}$, where $k$ is the number of pre-drawn intervals.

**Polyhedron Interpretation.** Consider the linear program of Section 5.1. The described shifting algorithm has the following geometric interpretation. When the constraints (4) are omitted, all solutions of the linear program form an unbounded polyhedron. The initial solution is one point of the polyhedron and the left-most representation is the vertex of the polyhedron minimizing all values $\ell_i$. One application of the LeftShift procedure corresponds to decreasing one variable while staying in the polyhedron. The algorithm computes a Manhatten-like path from the initial solution to the left-most representation consisting of $\mathcal{O}(n^2)$ shifts.

We believe that the polyhedron has some additional useful structure which might be exploited for constructing faster algorithms and might lead to discovering new useful properties of unit interval representations. It is also an interesting question whether some of our techniques can be generalized to other systems of difference constraints.

**Simultaneous Representations.** Let $G_1, \ldots, G_k$ be graphs having $V(G_i) \cap V(G_j) = I$ for each $i \neq j$. The SimRep($\mathcal{C}$) problem asks whether there exists representations $\mathcal{R}_1, \ldots, \mathcal{R}_k$ of $G_1, \ldots, G_k$ (of class $\mathcal{C}$) which assign the same sets to the vertices of $I$. This problem was considered in [15] and its relations to the partial representation extension problem were discussed in [8, 9].

We believe that it is possible to apply results and techniques to solve these problems for proper and unit interval graphs. First, one needs to construct simultaneous left-to-right orderings $<_1, \ldots, <_k$ having the same order on $I$. Then, we can use linear programming/shifting approach to construct the simultaneous representation. This is a possible direction of future research.

**Open Problem.** To conclude the paper, we present two open problems.

**Problem 1.** *Is it possible to solve the problem* RepExt(UNIT INT) *in faster time than* $\mathcal{O}(n^2 + nD(r))$*?*

We consider the other problem as currently the major open problem concerning restricted representations of graphs. The class of the intersection graphs of arcs of a circle is called *circular-arc graphs* (CIRCULAR-ARC); for references see [7]. We ask the following question:

**Problem 2.** *Can the problem* RepExt(CIRCULAR-ARC) *be solved in polynomial time?*

We believe that solving this problem might lead to a better understanding of the class itself. All known polynomial-time recognition algorithms are quite complex, and construct specific types of representations called *canonical representations*. Further, many results concerning circular-arc graphs were later shown to be false; for instance recently the graph isomorphism problem of circular-arc graphs is again open. To solve RepExt(CIRCULAR-ARC), the structure of all representations needs to be better understood which could be a major breakthrough concerning this and other classes.

## References


[1] P. Klavík, J. Kratochvíl, Y. Otachi, I. Rutter, T. Saitoh, M. Saumell, T. Vyskočil, Extending partial representations of proper and unit interval graphs, in: Algorithm Theory – SWAT 2014, Vol. 8503 of Lecture Notes in Computer Science, 2014, pp. 253–264.

[2] G. Hajós, Über eine Art von Graphen, Internationale Mathematische Nachrichten 11 (1957) 65.

[3] P. C. Gilmore, A. J. Hoffman, A characterization of comparability graphs and of interval graphs, Can. J. Math. 16 (1964) 539–548.

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

[5] D. G. Corneil, S. Olariu, L. Stewart, The LBFS structure and recognition of interval graphs, SIAM J. Discrete Math. 23 (4) (2009) 1905–1953.

[6] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, North-Holland Publishing Co., 2004.

[7] J. P. Spinrad, Efficient Graph Representations, Field Institute Monographs, 2003.

[8] P. Klavík, J. Kratochvíl, T. Vyskočil, Extending partial representations of interval graphs, in: Theory and Applications of Models of Computation, TAMC 2011, Vol. 6648 of Lecture Notes in Computer Science, 2011, pp. 276–285.

[9] T. Bläsius, I. Rutter, Simultaneous PQ-ordering with applications to constrained embedding problems, in: SODA'13: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 2013, pp. 1030–1043.

[10] P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh, T. Vyskočil, Linear-time algorithm for partial representation extension of interval graphs, In preparation.

[11] P. Klavík, J. Kratochvíl, T. Krawczyk, B. Walczak, Extending partial representations of function graphs and permutation graphs, in: Algorithms, ESA 2012, Vol. 7501 of Lecture Notes in Computer Science, 2012, pp. 671–682.

[12] S. Chaplick, R. Fulek, P. Klavk, Extending partial representations of circle graphs, in: Graph Drawing, Vol. 8242 of LNCS, Springer, 2013, pp. 131–142.

[13] P. Klavík, J. Kratochvíl, Y. Otachi, T. Saitoh, Extending partial representations of subclasses of chordal graphs, in: Algorithms and Computation, ISAAC 2012, Vol. 7676 of Lecture Notes in Computer Science, 2012, pp. 444–454.

[14] S. Chaplick, P. Dorbec, J. Kratochvíl, M. Montassier, J. Stacho, Contact representations of planar graph: Rebuilding is hard, To appear in WG 2014.

[15] K. R. Jampani, A. Lubiw, Simultaneous interval graphs, in: Algorithms and Computation, ISAAC 2010, Vol. 6506 of Lecture Notes in Computer Science, 2010, pp. 206–217.

[16] K. R. Jampani, A. Lubiw, The simultaneous representation problem for chordal, comparability and permutation graphs, Journal of Graph Algorithms and Applications 16 (2) (2012) 283–315.

[17] F. S. Roberts, Indifference graphs, in: F. Harary (Ed.), Proof Techniques in Graph Theory, Academic Press, 1969, pp. 139–146.

[18] P. Angelini, G. D. Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, I. Rutter, Testing planarity of partially embedded graphs, in: SODA'10: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, 2010, pp. 202–221.

[19] M. Patrignani, On extending a partial straight-line drawing, Int. J. Found. Comput. Sci. 17 (5) (2006) 1061–1070.

[20] M. Balko, P. Klavk, Y. Otachi, Bounded representations of interval and proper interval graphs, in: Algorithms and Computation, Vol. 8283 of LNCS, Springer, 2013, pp. 535–546.
[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Third Edition, 3rd Edition, The MIT Press, 2009.
[22] M. Fürer, Faster integer multiplication, SIAM J. Comput. 39 (3) (2009) 979–1005.
[23] X. Deng, P. Hell, J. Huang, Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs, SIAM J. Comput. 25 (2) (1996) 390–403.
[24] D. J. Rose, R. E. Tarjan, G. S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM Journal on Computing 5 (2) (1976) 266–283.
[25] F. S. Roberts, Representations of indifference relations, Ph.D. Thesis, Stanford University, 1968.
[26] D. G. Corneil, H. Kim, S. Natarajan, S. Olariu, A. P. Sprague, Simple linear time recognition of unit interval graphs, Inform. Process. Lett. 55 (2) (1995) 99–104.
[27] M. R. Garey, D. S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comput. 4 (4) (1975) 397–411.
[28] N. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (4) (1984) 373–395.
[29] M. Pirlot, Minimal representation of a semiorder, Theory and Decision 28 (1990) 109–141.
[30] B. Balof, J. P. Doignon, S. Fiorini, The representation polyhedron of a semiorder, Order 30 (1) (2013) 103–135.