Electronic Colloquium on Computational Complexity, Report No. 43 (2010)

Interval Graphs: Canonical Representation in Logspace

Johannes Köbler[1], Sebastian Kuhnert[1], Bastian Laubner[1], and Oleg Verbitsky[2]

[1] Humboldt-Universität zu Berlin
[2] Institute for Applied Problems of Mechanics and Mathematics, Ukrainian Academy of Sciences, Lviv. Supported in part by the Alexander von Humboldt foundation.
{koebler,kuhnert,laubner,verbitsk}@informatik.hu-berlin.de

Abstract. We present a logspace algorithm for computing a canonical interval representation and a canonical labeling of interval graphs. As a consequence, the isomorphism and automorphism problems for interval graphs are solvable in logspace.

1 Introduction

There has been persistent interest in the algorithmic aspects of interval graphs in the past decades, also spurred by their applicability to DNA sequencing (cf. [23]) and scheduling problems (cf. [18]). In 1976, Booth and Lueker presented the first recognition algorithm for interval graphs [2] running in time linear in the number of vertices and edges, which they followed up by a linear-time interval graph isomorphism algorithm [17]. These algorithms are based on a special data structure called *PQ-trees*. By pre-processing the graph's modular decomposition tree, Hsu and Ma [9] later presented a simpler linear-time recognition algorithm that avoids the use of PQ-trees. Habib et al. [7] achieve the same time bound employing the lexicographic breadth-first search of Rose, Tarjan, and Lueker [21] and in combination with smart pivoting. A parallel NC^2 algorithm was given by Klein in [11].

All of the above algorithms have in common that they compute a *perfect elimination ordering* (peo) of the graph's vertices. This ordering has the property that for every vertex, its neighborhood among its successors in the ordering forms a clique. Fulkerson and Gross [6] show that a graph has a peo if and only if it is chordal, and the above methods determine whether a graph is an interval graph once its peo has been determined in linear time.

The method we present here does neither rely on computing the graph's peo, nor do we use transitive orientation algorithms for comparability graphs as in [14]. Instead, the basis of our work is the observation of Laubner [15] that in an interval graph, the graph's maximal cliques and a certain modular decomposition tree are definable by means of first-order logic. This makes these objects tractable in logarithmic space and leads us to an L-algorithm for computing a canonical interval representation for any interval graph. This result is presented in Sections 4 and 5.

ISSN 1433-8092

In the work of Reif [19], the recognition of the class of interval graphs is reduced to the problem of deciding the connectivity query in undirected graphs. By the result of Reingold [20], interval graphs can therefore be recognized in L. In Section 3 we additionally show that recognition and isomorphism testing is hard for logspace, thereby proving L-completeness for both problems.

Finding logspace algorithms for the graph isomorphism problem of restricted graph classes is an active research area. It was started by Lindell with his canonization algorithm for trees [16]. In a series of results, Datta, Limaye, Nimbhorkar, Thierauf and Wagner generalize this to planar graphs [4]. Köbler and Kuhnert show the generalization to k-trees [13]. In each of these cases the isomorphism problem has a matching lower bound, i. e. it is shown to be L-complete. The graph classes considered in these results have in common that their clique size is bounded by a constant. To the best of our knowledge, the L-completeness result for interval graph isomorphism is the first for a natural graph class containing cliques of arbitrary size.

2 Preliminaries

As usual, L is the class of all languages decidable by Turing machines with read-only input tape and an $\mathcal{O}(\log n)$ bound on the space used on the working tapes. FL is the class of all functions computable by Turing machines that additionally have a write-only output tape.

2.1 Graphs

We write $G \cong H$ to say that G and H are isomorphic graphs. The vertex set of a graph G is denoted by $V(G)$. The set of all vertices having distance at most 1 from a vertex $v \in V(G)$ is called its *neighborhood* and denoted by $N(v)$. Note that $v \in N(v)$. We also use $N(u, v) = N(u) \cap N(v)$ for the common neighborhood of two vertices $u, v \in V(G)$.

We say that a vertex w *separates* vertices u and v if it is adjacent to exactly one of them. Two vertices inseparable by any other vertex are called *twins*.

We denote *intervals* (over nonnegative integers) as $[a, b] = \big\{ i \in \mathbb{N} \,\big|\, a \le i \le b \big\}$. We say $[a_1, b_1] < [a_2, b_2]$ if $a_1 < a_2$ or if $a_1 = a_2$ and $b_1 < b_2$. When we store an interval $[a, b]$ we represent it as tuple (a, b).

Let $\mathcal{F}$ be a family of sets, which will also be called a *set system*. We do not exclude the possibility that $A = B$ for some $A, B \in \mathcal{F}$, i.e., $\mathcal{F}$ is a multiset whose elements are sets. The *intersection graph* of $\mathcal{F}$ is the graph with vertex set $\mathcal{F}$ where A and B are adjacent if they have a nonempty intersection. Note that, if $A = B$, these two vertices are twins in the intersection graph.

A graph G is an *interval graph* if it is isomorphic to the intersection graph of a family of intervals. The latter is called an *interval representation* of G. Given an interval representation of G and a correspondence between the intervals and the vertices, we will use notation I_v for the interval corresponding to a

vertex $v \in V(G)$. An interval representation is called *minimal*, if the union of its intervals has minimal size.

An *interval labeling* of a graph G is a function $\ell\colon V(G) \to \{[l,r] \mid l,r \in \mathbb{N}\}$, such that ℓ is an isomorphism from G to the intersection graph G^ℓ of the set system $\{\ell(v) \mid v \in V(G)\}$. A *canonical interval labeling* is a function that maps an interval graph G to an interval labeling ℓ_G such that $G \cong H \Leftrightarrow G^{\ell_G} = H^{\ell_H}$. We call $V(G^{\ell_G})$ the *canonical interval representation* of G. Note that a canonical interval labeling implies a canonical labeling, as the intervals can be sorted and renamed. We show how a canonical interval labeling can be constructed in logspace.

2.2 Bundles of maxcliques

An inclusion-maximal clique in a graph G will be called a *maxclique*. We denote the set of all maxcliques by $\mathcal{M}$. The following lemma allows us to use maxcliques in logspace algorithms.

Lemma 1. *Let G be an interval graph. Every maxclique C in G contains vertices u and v such that $C = N(u,v)$.*

Proof. We have $C \subseteq N(u,v)$ for any $u,v \in C$ and, therefore, we only need to find u,v such that $N(u,v) \subseteq C$. For this purpose, consider an interval representation of G and choose $u,v \in C$ so that $I_u \cap I_v$ is inclusion-minimal. For any $w \in C$, we have $I_w \supseteq I_u \cap I_v$ for else $I_u \cap I_w$ or $I_w \cap I_v$ would be strictly included in $I_u \cap I_v$. Suppose now that $z \in N(u,v)$. Since I_z intersects $I_u \cap I_v$, it has nonempty intersection with I_w for each $w \in C$. By maximality, $z \in C$. □

The *(maxclique) bundle* B_v at a vertex v consists of all maxcliques containing v. The *bundle graph* $\mathbb{B}(G)$ is the intersection graph of the family of all bundles $\{B_v\}_{v \in V(G)}$.

Lemma 2. $\mathbb{B}(G) \cong G$.

Proof. The mapping $f(v) = B_v$ is an isomorphism from G to $\mathbb{B}(G)$. Indeed, if B_v and B_u have a common maxclique C, then C contains both v and u and, hence, those are adjacent. Conversely, if v and u are adjacent, then B_v and B_u share any maxclique containing the two vertices. □

Lemma 2 will allow us to deal with $\mathbb{B}(G)$ instead of the original graph G, simulating adjacency of u and v in G by the intersection property of B_u and B_v. We will gain more expressiveness, obtaining a possibility to speak also on inclusion and other set-theoretic relations between the bundles.

We say that sets A and B *overlap* and write $A \between B$ if A and B intersect (i.e., have a nonempty intersection) but neither of them includes the other. The *overlap graph* $\mathbb{O}(G)$ is a spanning subgraph of $\mathbb{B}(G)$ where bundles B_v and B_u are adjacent iff they overlap or are equal. Of course, $\mathbb{O}(G)$ can be disconnected even if $\mathbb{B}(G)$ is connected. Connected components of $\mathbb{O}(G)$ will be referred to

as *overlap components* of the bundle graph. By $\langle B \rangle$ we will denote the overlap component containing a bundle B. For an overlap component L, we denote the set of maxcliques occurring in some bundle $B \in V(L)$ by $\mathcal{M}_L = \bigcup V(L)$.

Lemma 3. *Let L and L' be different overlap components. If a bundle $A \in L$ includes at least one bundle in L', then A includes all bundles in L'.*

Proof. The lemma follows from a simple observation that the conditions $B \subset A$, $B \between B'$, and $\neg(B' \between A)$ imply that $B' \subset A$. □

It is well known that there is a one-to-one correspondence between the maxcliques of an interval graph G and the numbers used in a minimal interval representation of G. Moreover the hypergraph of bundles and the hypergraph of intervals are isomorphic. This will allow us to apply a geometric intuition for investigating properties of the bundle graph.

We call a bundle B *$\between$-isolated* if there is no bundle B' such that $B \between B'$. Suppose that B is not $\between$-isolated. If $B \between B'$, we will call the intersection $B \cap B'$ a *cut* of B. Let $\mathrm{Cut}(B)$ denote the set of all cuts of B. Observe that $\mathrm{Cut}(B)$ is either an inclusion chain or it is split into two inclusion chains, as the cuts must be at the sides of B in any interval representation. We will call B *marginal* if it has only one inclusion chain.

The following lemma follows from the mentioned isomorphism between the hypergraph of bundles and the hypergraph of intervals.

Lemma 4. *Unless an overlap component consists of a single $\between$-isolated bundle, it has, up to set equality, exactly two inclusion-maximal marginal bundles.*

3 Hardness of interval graph problems

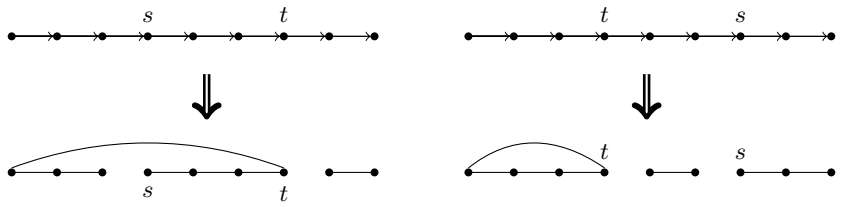
All hardness results in this section are under DLOGTIME-uniform AC^0 reductions.

Lemma 5. *The problem of deciding whether a given graph is an interval graph is* L*-hard.*

Proof. We reduce from the L-complete problem ORD (cf. [5]): Given a directed path P and vertices $s, t \in V(P)$, decide if there is a path from s to t.

If P is such a directed path, it can be checked uniformly in constant depth if s or t are among the first 3 vertices in the path. If so, output a trivial *yes* or *no*-instance depending on whether s has been encountered first.

If neither s nor t are among the first 3 vertices of P, then we construct an undirected graph G on the same vertex set as P as follows: delete the incoming edge of s, delete the outgoing edge of t (if any), insert an edge connecting the first vertex in P and t, and then forget about the directions of the edges. Below is an illustration of this construction. It can clearly be done in DLOGTIME-uniform AC^0. We claim that G is an interval graph if and only if there is a path from s to t in P.



In fact, if there is no such path, then t lies on an undirected cycle of length at least 4, thus G is not an interval graph. If there is such a path, however, then G consists of two disjoint paths, which is an interval graph. □

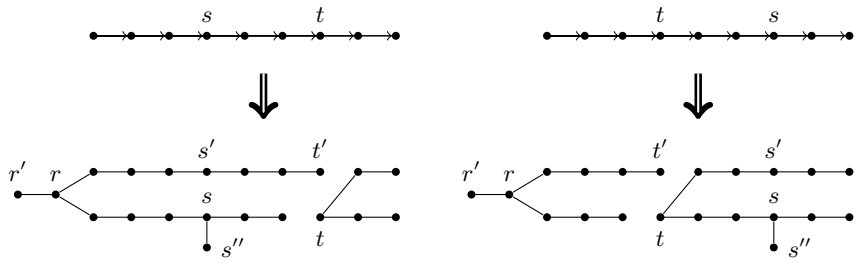
By results of Reif [19] and Reingold [20], this problem is also contained in L. Thus, interval graph recognition is L-complete.

Remark 6. The reduction given in the proof of Lemma 5 also proves that it is L-hard to decide whether a given graph is *chordal*. Again, by results of Reif [19] and Reingold [20], recognition of chordal graphs is therefore L-complete.

Lemma 7. *Given an interval graph G, the problem of deciding if it has a non-trivial automorphism is* L*-hard. The same holds for the problem of deciding if two interval graphs are isomorphic.*

Proof. We only show the hardness of deciding whether there is a non-trivial automorphism. The construction for the hardness of the isomorphism problem is very similar.

As in Lemma 5, we reduce from ORD. Let a directed path P and vertices $s, t \in V(P) = \{v_1, \dots v_n\}$ be given. In constant depth, we can check for the cases when s or t are contained among the first or last two vertices in P, or s is within distance 1 from t. Otherwise, let P' be an isomorphic copy of P on the vertex set $\{v'_1, \dots, v'_n\}$, and construct the undirected graph G as follows: start with the disjoint union of P and P' and vertices $\{r, r', s''\}$. If v' is the vertex directly following t' in P, then delete the edge from t' to v' and add instead an edge from t to v'. Now forget about the edge directions and add undirected edges rv_1, rv'_1, rr', and ss'' (where v_1, v'_1 are the first vertices in P, P', respectively). The following drawing illustrates the construction.



It is clear that G is an interval graph. If there is a path from s to t in P, then the path in G which contains t has a nontrivial automorphism. However, if there

is no such path in P, then the vertices r' and s'' fix the two non-isomorphic connected components of G, so that it has no automorphism except for the identity morphism. □

4 Canonizing overlap components

In this section we show how to compute a canonical interval labeling of an overlap component L of some interval graph G. Our canonical interval labeling has the additional property that for all $B_v \in V(L) : \|\ell(B_v)\| = \|B_v\|$. This property facilitates easier combination of the canonical interval labelings of the overlap components to a canonical interval labeling of the whole graph G, as the span of the intervals is already that of the interval labeling of G. We can assume that there are no twins, as our algorithm also handles colored graphs and twins can be replaced by a single node colored with their multiplicity.

The key observation for obtaining the canonical interval labeling is that an overlap component L (without twins) is either rigid or has exactly one nontrivial isomorphism α. The latter case occurs iff L has a symmetric interval representation; α then corresponds to mirroring this interval representation.

Let L be an overlap component of an interval graph G. We call two maxcliques $M, M' \in \mathcal{M}_L$ *indistinguishable in* L (denoted as $M \sim_L M'$), if there is no bundle $B_v \in V(L)$ that contains exactly one of them. Clearly, $\sim_L$ is an equivalence relation on $\mathcal{M}_L$.

If L consists of a single bundle B_v, we have the canonical interval labeling ℓ_L that maps B_v to $[0, \|\mathcal{M}_L\| - 1]$. So from now on we can assume that L consist of at least two bundles. By Lemma 4 we know that L has only two inclusion-maximal marginal bundles B_1, B_2, one at each side of any interval representation of L. We now choose two maxcliques M_1, M_2 representing the two sides of L: Let $\mathcal{B}_i$ be the set of marginal bundles that are contained in B_i (including B_i itself; this excludes marginal bundles at the other side). Choose $M_i \in B_i$ such that it is not contained in any bundle $B \in V(L) \setminus \mathcal{B}_i$ and in as few $B \in \mathcal{B}_i$ as possible. We call M_1 and M_2 *side-cliques* of L.

Following [15] we can use a side-clique M to define a partial order $\prec_M$ on $\mathcal{M}_L$ as the smallest relation that satisfies the following two properties:

1. $M \prec_M C$ for all maxcliques $C \in \mathcal{M}_L \setminus \{M\}$.
2. For each bundle B_v in L and maxcliques $C_1, C_2 \in B_v$, $D \notin B_v$:

$$C_1 \prec_M D \Leftrightarrow C_2 \prec_M D \qquad \text{and} \qquad D \prec_M C_1 \Leftrightarrow D \prec_M C_2 \tag{1}$$

$C \prec_M D$ can be read as "if M is leftmost, then C is left of D". Note that $\prec_{M_1}$ does not depend on the choice of the side-clique M_1, as all possible choices for M_1 are indistinguishable in L. Likewise, $\prec_{M_2}$ does not depend on the choice of M_2. Laubner proves that incomparability w. r. t. $\prec_M$ is an equivalence relation. He also proves the following properties which we rephrase to match our terminology.

Lemma 8 ([15, Corollary 4.4]). *If two maxcliques $C, D \in \mathcal{M}_L$ are incomparable w. r. t. $\prec_M$ (M being a side-clique in $\mathcal{M}_L$), then C and D are indistinguishable in L.* □

Lemma 9 ([15, Section 4.2]). *For each bundle B_v and maxcliques $C_1, C_2 \in B_v$ there is no maxclique $D \in \mathcal{M}_L \setminus B_v$ with $C_1 \prec_M D \prec_M C_2$.* □

Also it is easy to see that $\prec_M$ is invariant under isomorphism, as it only depends on the structure of G. By Lemma 8 we can use $\prec_M$ to define an interval labeling ℓ_M of L:

$$\begin{aligned} \ell_M \colon V(L) &\to \left\{ [l,r] \;\middle|\; l,r \in \left[0, \|\mathcal{M}_L\| - 1\right] \right\} \\ B_v &\mapsto \left[\operatorname{pos}(B_v), \operatorname{pos}(B_v) + \|B_v\| - 1 \right], \end{aligned}$$

where a maxclique $C \in \mathcal{M}_L$ has position $\operatorname{pos}(C) = \|\{D \in \mathcal{M}_L \mid D \prec_M C\}\|$ and a bundle $B_v \in V(L)$ has position $\operatorname{pos}(B_v) = \min\{\operatorname{pos}(C) \mid C \in B_v\}$.

Lemma 10. *ℓ_{M_1} and ℓ_{M_2} are indeed interval labelings for L.*

Proof. Let $M \in \{M_1, M_2\}$. We first show that for each maxclique $C \in \mathcal{M}_L$ and each bundle $B_v \in V(L)$ we have $C \in B_v \Leftrightarrow \operatorname{pos}(C) \in \ell_M(B_v)$. If $C \in B_v$, then $\operatorname{pos}(B_v) \leq \operatorname{pos}(C)$ (by definition of $\operatorname{pos}(B_v)$) and $\operatorname{pos}(C) < \operatorname{pos}(B_v) + \|B_v\|$ (by Lemma 9) and thus $\operatorname{pos}(C) \in \ell_M(B_v)$. If $C \notin B_v$, then either $\operatorname{pos}(C) < \operatorname{pos}(B_v)$ or $\operatorname{pos}(C) \geq \operatorname{pos}(B_v) + \|B_v\|$, again by Lemma 9. This implies $\operatorname{pos}(C) \notin \ell_M(B_v)$.

This claim already proves that if two bundles $B_u, B_v \in V(L)$ share a maxclique C, then $\ell_M(B_u)$ and $\ell_M(B_v)$ intersect. For the opposite direction let $\ell_M(B_u) \cap \ell_M(B_v)$ be nonempty and w. l. o. g. $\operatorname{pos}(B_u) \leq \operatorname{pos}(B_v)$. This implies $\operatorname{pos}(B_v) \in \ell_M(B_u)$ and thus the maxclique $C \in B_v$ with $\operatorname{pos}(C) = \operatorname{pos}(B_v)$ is also in B_u. □

By these observations, we can choose the canonical interval labeling ℓ_L of L among ℓ_{M_1} and ℓ_{M_2} such that the set L^{ℓ_L} of intervals becomes minimal. We denote the corresponding order on the maxcliques of L by $\prec_L$. By $\vec{L}$ we denote the list of vertices v whose bundles B_v are in L, ordered by their intervals $\ell_L(B_v)$.

Lemma 11. *Given an interval graph G and an overlap component L of G, the following can be done in logspace:*

1. *Computing the partial order $\prec_L$ on $\mathcal{M}_L$,*
2. *computing a canonical interval labeling ℓ_L of L,*
3. *computing the corresponding ordered list of vertices $\vec{L}$, and*
4. *deciding if L^{ℓ_L} is symmetric or not.*

Proof. By Lemma 1 each maxclique of an interval graph G can be represented as $N(u, v)$ for some $u, v \in V(G)$. This implies that the set of all maxcliques and the bundles B_v can be enumerated in logspace.

To prove that $C \prec_M D$ can be decided in logspace, we construct an undirected graph H with nodes $V(G) = \left\{(C_1, C_2) \mid C_1 \neq C_2 \text{ maxcliques in } L\right\}$ meaning "$C_1 \prec_M C_2$" and edges corresponding to the equivalences given in the definition (1) of $\prec_M$. Additionally, we add a start vertex s and connect it to all nodes of the form (M, C). Now we have $C \prec_M D$ iff (C, D) is reachable from s in

H. Reachability in undirected graphs is decidable in logspace using Reingold's algorithm [20].

Once $\prec_M$ can be decided in logspace, it is easy to compute ℓ_M and to choose the side-clique $M \in \{M_1, M_2\}$ such that L^{ℓ_M} becomes minimal. L^{ℓ_L} is symmetric iff both are minimal.

Given ℓ_L, it is easy to compare the bundles in L, and thereby compute $\vec{L}$. □

5 Canonizing interval graphs

Let G be an interval graph. Again we assume that G has no twins, but allow a coloring of G instead. We also assume that G is connected – if not, we add a new vertex and connect it to all others. We define the *slots* of an overlap component L of G as the equivalence classes $[C]_{\sim_L}$ of the relation $\sim_L$, namely $\mathit{slots}(L) = \{[C]_{\sim_L} \mid C \in \mathcal{M}_L\}$. The position of a slot S in L is defined analogously to that of a bundle, namely $\mathrm{pos}(S) = \min\{\mathrm{pos}(C) \mid C \in S\}$. Additionally we define the position of S from the right: $\mathrm{rpos}(S) = \min_{C \in S} \|\{D \in \mathcal{M}_L \mid C \prec_L D\}\|$. If an overlap component L has a symmetric canon L^{ℓ_L}, we call a slot S of L *low* if $\mathrm{pos}(S) < \mathrm{rpos}(S)$, *middle* if $\mathrm{pos}(S) = \mathrm{rpos}(S)$, and *high* if $\mathrm{pos}(S) > \mathrm{rpos}(S)$. If L^{ℓ_L} is not symmetric, we call all its slots *low*. We say an overlap component L' is *located at* a slot S (of L), if $\mathcal{M}_{L'} \subseteq S$ and there is no overlap component L'' such that $\mathcal{M}_{L'} \subset \mathcal{M}_{L''} \subset \mathcal{M}_L$ (note that different overlap components have different maxclique sets, as twins are not allowed).

5.1 Tree representation

Using the above notions, we now define a tree representation for interval graphs.

Definition 12. *For a connected interval graph G without twins, its* tree representation $\mathbb{T}(G)$ *is defined by*

$$\begin{aligned} V(\mathbb{T}(G)) =& \{\vec{L}, lo_L, mi_L, hi_L \mid L \text{ is an overlap component of } G\} \\ & \cup \{S \mid S \text{ is a slot of some overlap component } L \text{ of } G\} \\ E(\mathbb{T}(G)) =& \{(\vec{L}, lo_L), (\vec{L}, mi_L), (\vec{L}, hi_L) \mid L \text{ is an overlap component of } G\} \\ & \cup \{(lo_L, S), (mi_L, S), (hi_L, S) \mid S \text{ is a low/middle/high slot in } L\} \\ & \cup \{(S, \vec{L}) \mid \text{the overlap component } L \text{ is located at slot } S\} \end{aligned}$$

Further we define a coloring c of the component-nodes $\vec{L}$ and slot-nodes S by

$$c(\vec{L}) = L^{\ell_L}$$

$$c(S) = \begin{cases} \mathrm{pos}(S) & \text{if } S \text{ is low or middle} \\ \mathrm{rpos}(S) & \text{if } S \text{ is high} \end{cases}$$

If G is colored, the intervals in $c(\vec{L}) = L^{\ell_L}$ also inherit the colors of the corresponding vertices in $V(G)$.

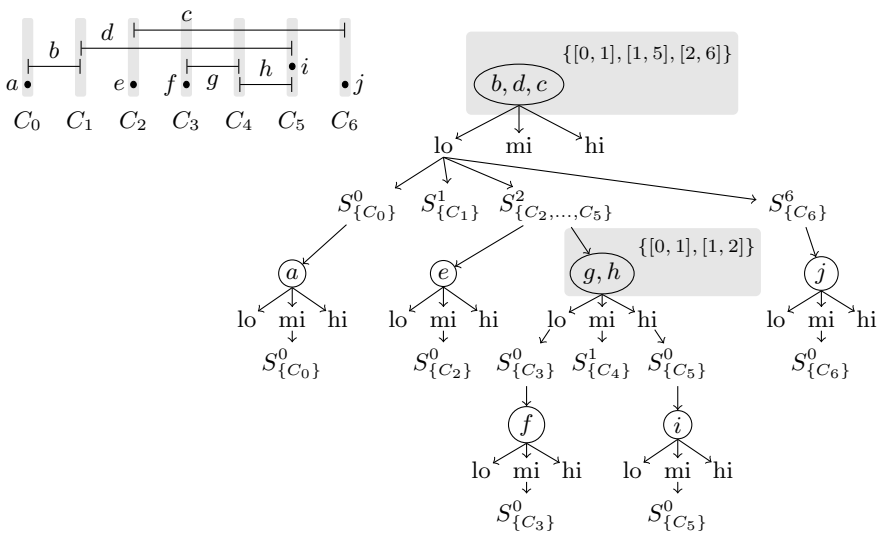


Fig. 1. An interval graph representation of a graph G and the corresponding tree representation $\mathbb{T}(G)$. Gray rectangles in $\mathbb{T}(G)$ indicate the color of overlap components. Overlap components have color $\{[0,0]\}$ where not indicated. Slot name $S^k_{\{C_i,\dots,C_j\}}$ denotes slot $\{C_i,\dots,C_j\}$ and indicates that its color is k.

As G is connected, there is an overlap component L_0 such that $\mathcal{M}_{L_0} = \mathcal{M}$. $\vec{L}_0$ is the root of the directed tree $\mathbb{T}(G)$.

Our goal is to compute a canonical interval labeling of G using a modified version of Lindell's canonization algorithm for trees [16] on $\mathbb{T}(G)$. For this approach, we must first compute $\mathbb{T}(G)$ in logspace.

Lemma 13. *For a given interval graph G, $\mathbb{T}(G)$ can be computed in* FL.

Proof. Lemma 11 allows us to compute $\vec{L}$, $\prec_L$ and ℓ_L for an overlap component L in logspace. A slot S of an overlap component L can be represented by a tuple (L, M), where M is a maxclique contained in S: We then have $S = [M]_{\sim_L}$. To make this representation unique, we choose the $M \in S$ that occurs first in the input. Using this representation and $\prec_L$, we can enumerate all slots and compute $\mathrm{pos}(S)$ and $\mathrm{rpos}(S)$. Using this information, $\mathbb{T}(G)$ can easily be constructed in logspace. □

We proceed to show a basic structural property of $\mathbb{T}(G)$ that we exploit to compute the canonical interval labeling.

Lemma 14. *There is a one-to-one correspondence between maxcliques of G and leaf-nodes of $\mathbb{T}(G)$ that are slots.*

Proof. Let S be a slot of some overlap component L such that S is a leaf of $\mathbb{T}(G)$. By definition of slots, the maxcliques in S are not distinguished by L. This implies that the maxcliques in S are also not distinguished by any bundle

in an overlap component on the path to the root. By Lemma 3 no maxclique in S can be contained in any overlap component not on the path from S to the root. So there is no vertex $v \in V(G)$ whose bundle B_v contains only a part of S. So S must be a single maxclique of G.

For the opposite direction, let M be any maxclique of G. Then $M \in \mathcal{M}_{L_0}$, as the root overlap component L_0 contains all maxcliques. Now if $M \in \mathcal{M}_L$ for some overlap component L, then M is in exactly one of the slots of L, as slots are equivalence classes and thus partition $\mathcal{M}_L$. And if M is contained in some slot S that is not a child in $\mathbb{T}(G)$, M must be contained in exactly one overlap component located at S (if none, M would not be maximal, if more, these overlap components would overlap or contain twins). Using these observations, we can trace M to a single slot S that is a leaf of $\mathbb{T}(G)$. □

We will need to compute a canonical labeling of $\mathbb{T}(G)$ and call it $\ell_{\mathbb{T}(G)}$. We observe a generalization of Lindell's tree canonization algorithm [16].

Lemma 15. *Lindell's algorithm [16] can be extended to colored trees and to output not only a canonical form, but also a canonical labeling. This modification preserves the logarithmic space bound.*

Proof sketch. Colors can be handled by extending the *tree isomorphism order* defined in [16] by using $color(s) < color(t)$ as additional condition (where s and t are the roots of the trees to compare). The canonical labeling can be computed by using a counter i initialized to 0: Instead of printing (the opening parenthesis of) the canon of a node v, increment i and print "$v \mapsto i$". □

5.2 Computing a canonical interval labeling

Our aim is a traversal of $\mathbb{T}(G)$ that is left-to-right in the resulting canon. That is, we visit the leaf slots in ascending order of the positions of their corresponding maxcliques in the computed canonical interval representation. To achieve this, we use the canonical labeling $\ell_{\mathbb{T}(G)}$.

We first recall the *logspace tree traversal* that is used in Lindell's canonization algorithm. Only the current node must be remembered, as the following operations are possible in logspace:

- Given a node, go to its first child.
- Given a node, go to its next sibling.
- Given a node, go to its parent.

"First" and "next" can be respective to any order on the children of a node that can be evaluated in logspace. In our left-to-right traversal we use the following order:

- The children of an overlap component node $\vec{L}$ are either ordered $lo_L < mi_L < hi_L$ (if L^{ℓ_L} is not symmetric or if $\ell_{\mathbb{T}(G)}(lo_L) < \ell_{\mathbb{T}(G)}(hi_L)$) or $hi_L < mi_L < lo_L$ (otherwise).

- The children of the first child of an overlap component node $\vec{L}$ (this can be either lo_L or hi_L) are visited in ascending order of their colors.
- The children of the last child of an overlap component node $\vec{L}$ (this can be either hi_L or lo_L) are visited in descending order of their colors.
- The children of a slot node are ordered by their label assigned by $\ell_{\mathbb{T}(G)}$.

Note that the children of lo_L and hi_L all have different colors. Also, mi_L can have at most one child. All these conditions can be evaluated in logspace without using non-local information. Traversing $\mathbb{T}(G)$ in this order makes sure that the slots of an overlap component L are visited either in ascending or descending order of their positions. The latter case can only occur if L^{ℓ_L} is symmetric.

We complete the description of our algorithm by showing how, while processing $\mathbb{T}(G)$, a canonical interval labeling can be computed in logspace. Additionally to the current node we store a *current offset* o that equals to the number of maxcliques we have passed already. Formally, we initialize $o = 0$ and increment it by 1 whenever the logspace tree traversal passes a slot node that is a leaf. Whenever we enter an overlap component node $\vec{L} = (v_1, \dots, v_k)$ for the first time, we output the mappings $v_i \mapsto [l_i + o, r_i + o]$ where $[l_i, r_i]$ is the ith-smallest interval in $c(\vec{L}) = L^{\ell_L}$ if $lo_L < mi_L < hi_L$, and the ith-largest interval otherwise. In the first case this results in $\ell_G(v) = \ell_L(B_v) + o$. In the second case this association is mirrored: If $\alpha\colon V(L) \to V(L)$ mirrors L, then $\ell_G(v) = \ell_L(\alpha(B_v)) + o$. After traversing all of $\mathbb{T}(G)$, we have output a mapping for each $v \in V(G)$. We call this mapping ℓ_G.

Lemma 16. *ℓ_G is an interval labeling of G.*

Proof. Take any $u, v \in V(G)$. Let $L = \langle B_u \rangle$ and $L' = \langle B_v \rangle$. Let o (resp. o') be the current offset when L (resp. L') is entered for the first time. If $L = L'$, we are done by Lemma 10, as the offset o preserves intersection. If $\mathcal{M}_L$ and $\mathcal{M}_{L'}$ do not intersect, then u and v are not adjacent. W. l. o. g. assume $o < o'$. Indeed we have $o + \|\mathcal{M}_L\| \leq o'$, as the offset is advanced by one for each slot leaf in the subtree of $\vec{L}$ (which correspond to the maxcliques in $\mathcal{M}_L$ by Lemma 14). As $\ell_L(u) < \|\mathcal{M}_L\|$ (see the definition of ℓ_L), $\ell(u)$ and $\ell(v)$ do not intersect. If $\mathcal{M}_L$ and $\mathcal{M}_{L'}$ do intersect, one must be contained in the other. W. l. o. g. assume $\mathcal{M}_{L'} \subset \mathcal{M}_L$ and let S be the slot of L in which $\mathcal{M}_{L'}$ is contained. If u is contained in some maxclique $M \in S$ (and thereby contained in all $M \in S$), then u and v are adjacent. By Lemma 14 and the order of our tree traversal we have $\ell_L(B_u) + o \supseteq [o', o' + \|\mathcal{M}_{L'}\|]$. Finally, if u is contained in no maxclique $M \in S$, then u and v are not adjacent. Also the slot leaves corresponding to the maxcliques in B_u will be processed all before or all after $\vec{L}'$, so $\ell_G(v)$ and $\ell_G(u)$ do not intersect. □

In the remainder of this section we prove that the interval labeling ℓ_G is canonical. The next lemma shows that the colored tree representations of isomorphic interval graphs are also isomorphic.

Lemma 17. *Let $\phi \in S_n$ be an isomorphism between two connected interval graphs G and H without twins. Then there is an isomorphism ϕ' between $\mathbb{T}(G)$ and $\mathbb{T}(H)$.*

Proof. Since ϕ induces a unique mapping of the overlap components L of G to isomorphic overlap components L' of H, it is clear how to define ϕ' on the component-nodes $\vec{L}$ of $\mathbb{T}(G)$. Further, since the canonical interval representations L^{ℓ_L} and $L'^{\ell_{L'}}$ coincide, $\vec{L}$ and $\phi'(\vec{L})$ indeed have the same colors.

In order to define ϕ' on the *lo*, *mi* and *hi* nodes of $\mathbb{T}(G)$, consider a component-node $\vec{L} = (u_1, \ldots, u_k)$ of $\mathbb{T}(G)$. If L^{ℓ_L} is not symmetric, then it follows that $\vec{L'} = (\phi(u_1), \ldots, \phi(u_k))$. Otherwise, it is also possible that $\vec{L'} = (\phi(u_k), \ldots, \phi(u_1))$. In the first case we let ϕ' map the children lo_L, mi_L, hi_L of $\vec{L}$ to $\phi'(lo_L) = lo_{L'}$, $\phi'(mi_L) = mi_{L'}$, $\phi'(hi_L) = hi_{L'}$. If however $\vec{L'} = (\phi(u_k), \ldots, \phi(u_1))$, then we let ϕ' map the children of $\vec{L}$ to $\phi'(lo_L) = hi_{L'}$, $\phi'(mi_L) = mi_{L'}$ and $\phi'(hi_L) = lo_{L'}$.

Finally, since all children of a *lo*, *mi* or *hi* node have different colors there is a unique way to define ϕ' on the slot-nodes of $\mathbb{T}(G)$.

Now it can be easily checked that ϕ' indeed is an isomorphism between $\mathbb{T}(G)$ and $\mathbb{T}(H)$. □

Now we are ready to prove our main result.

Theorem 18. *Given an interval graph G, a canonical interval labeling ℓ_G for G can be computed in* FL.

Proof. We have already shown that the labeling ℓ_G is computable in FL and that G^{ℓ_G} is isomorphic to G. It remains to show that the labelings ℓ_G and ℓ_H of any two isomorphic interval graphs G and H map these graphs to the same interval representation $G^{\ell_G} = H^{\ell_H}$.

To see this, note that by Lemma 17, the colored trees $\mathbb{T}(G)$ and $\mathbb{T}(H)$ are isomorphic. Hence it follows that the canonical labelings $\ell_{\mathbb{T}(G)}$ and $\ell_{\mathbb{T}(H)}$ map these trees to the same colored tree $\mathbb{T}(G)^{\ell_{\mathbb{T}(G)}} = \mathbb{T}(H)^{\ell_{\mathbb{T}(H)}}$. Further, it is easy to see that the interval representation G^{ℓ_G} only depends on the tree $\mathbb{T}(G)^{\ell_{\mathbb{T}(G)}}$, implying that $G^{\ell_G} = H^{\ell_H}$. □

There is a standard Turing reduction of the automorphism group problem (i.e., computing a generating set of the automorphism group of a given graph) to the search version of graph isomorphism for colored graphs (cf. [8, 12]). It is not hard to see that this reduction can be performed in logspace.

Corollary 19. *Computing a generating set of the automorphism group of a given interval graph, and hence computing a canonical labeling coset for a given interval graph is in* FL. *Further, the automorphism problem (i. e., deciding if a given graph has a non-trivial automorphism) for interval graphs is* L*-complete.*

6 Conclusion

We have proved that a canonical interval labeling of an interval graph can be computed in logarithmic space. In particular, this puts into FL the problems of computing an interval representation, deciding interval graph isomorphism, and finding a generating set of an interval graph's automorphism group. Furthermore, we showed L-hardness of the problems of recognition and isomorphism of interval graphs and of deciding if an interval graph has a non-trivial automorphism. We conclude that all these problems are in fact L-complete.

Going beyond interval graphs, there are several natural graph classes that suggest an investigation whether they can similarly be handled in L. For example, circular-arc graphs generalize interval graphs as intersection graphs of arcs on a circle. Just like interval graphs, circular-arc graphs can be recognized efficiently in linear time (cf. [10]). However, while intuition suggests a reduction of circular-arc graphs to interval graphs by "cutting open" the circle that carries the graph's circular-arc representation, all known algorithms require additional techniques that are fairly specific to circular-arc graphs. One of the obstacles is that maxcliques cannot be handled as easily as in Lemma 1, since there are possibly exponentially many of them.

Another generalization of interval graphs is the class of rooted directed path graphs, i.e., intersection graphs of paths in a rooted and directed tree. While in this class, maxcliques can still be recognized in a similar way as in this paper, the recursive procedure for linearly ordering maxcliques as given in Section 4 cannot be employed in the presence of tree nodes of degree ≥ 3 (cf. [15]).

We observe that in the above paragraph, it is important that trees are rooted and directed accordingly, since otherwise the class becomes graph isomorphism-complete (cf. [1]). The same is true for boxicity-d graphs ($d \geq 2$), the intersection graphs of axis-parallel boxes in $\mathbb{R}^d$ (cf. [22]). Finally, we would like to point to [3] for further graph classes for which it is unknown if recognition and isomorphism can be handled in L.

References

1. Luitpold Babel, Ilia N. Ponomarenko, and Gottfried Tinhofer. The isomorphism problem for directed path graphs and for rooted directed path graphs. *J. Algorithms*, 21(3):542–564, 1996.
2. Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
3. Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Monographs on Discrete Mathematics and Applications. SIAM, 2004.
4. Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity*, pages 203–214. IEEE Computer Society, 2009.
5. Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400 – 411, 1997.

6. D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965.
7. Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000.
8. C. M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer, 1982.
9. Wen-Lian Hsu and Tze-Heng Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM J. Comput.*, 28(3):1004–1020, 1999.
10. Haim Kaplan and Yahav Nussbaum. A simpler linear-time recognition of circular-arc graphs. In *SWAT*, pages 41–52, 2006.
11. Philip N. Klein. Efficient parallel algorithms for chordal graphs. *SIAM J. Comput.*, 25(4):797–827, 1996.
12. J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1993.
13. Johannes Köbler and Sebastian Kuhnert. The isomorphism problem for k-trees is complete for logspace. In *34th Mathematical Foundations of Computer Science 2009 (MFCS)*, pages 537–548, 2009.
14. Dexter Kozen, Umesh V. Vazirani, and Vijay V. Vazirani. Nc algorithms for comparability graphs, interval gaphs, and testing for unique perfect matching. In S. N. Maheshwari, editor, *FSTTCS*, volume 206 of *Lecture Notes in Computer Science*, pages 496–503. Springer, 1985.
15. Bastian Laubner. Capturing polynomial time on interval graphs. *arXiv*, 0911.3799, 2010.
16. Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 400–404, New York, NY, USA, 1992. ACM.
17. George S. Lueker and Kellogg S. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979.
18. R.H. Möhring. *Graphs and Order*, volume 147 of *NATO ASI Series C, Mathematical and Physical Sciences*, pages 41–102. D. Reidel, 1984.
19. John H. Reif. Symmetric complementation. *J. ACM*, 31(2):401–421, 1984.
20. Omer Reingold. Undirected st-connectivity in log-space. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 376–385, New York, NY, USA, 2005. ACM.
21. Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.
22. Ryuhei Uehara. Simple geometrical intersection graphs. In *WALCOM*, pages 25–33, 2008.
23. Peisen Zhang, Eric A. Schon, Stuart G. Fischer, Eftihia Cayanis, Janie Weiss, Susan Kistler, and Philip E. Bourne. An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA. *Bioinformatics*, 10(3):309–317, 1994.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de