The Computational Complexity Column

by

V. Arvind

Institute of Mathematical Sciences, CIT Campus, Taramani
Chennai 600113, India
arvind@imsc.res.in
http://www.imsc.res.in/~arvind

Ever since Reingold's deterministic logspace algorithm [66] for undirected graph reachability, logspace algorithms for various combinatorial problems have been discovered and it is now a flourishing area of research. Notable examples include special cases of directed graph reachability and planar graph isomorphism [23].

In this interesting article, Johannes Köbler, Sebastian Kuhnert and Oleg Verbitsky discuss the structural properties of interval graphs and other technical ingredients that go into their recent logspace isomorphism algorithm for interval graphs, along with some generalizations and new directions.

Around and Beyond the Isomorphism Problem for Interval Graphs

Johannes Köbler Sebastian Kuhnert* Oleg Verbitsky†
Humboldt-Universität zu Berlin, Institut für Informatik
{koebler,kuhnert,verbitsk}@informatik.hu-berlin.de

Abstract

The class of problems solvable in logarithmic space has recently replenished with the isomorphism testing for interval graphs. We discuss this result, prospects of its extension to larger classes of graphs, and related issues such as constructing canonical models of intersection graphs and solving the Star System Problem for restricted classes of graphs.

1 Introduction

Graph Isomorphism (GI for short) is the problem of determining whether or not two given graphs are isomorphic. The problem is in the class NP, but its complexity status is open since decades; see, e.g., the surveys [64, 31, 77, 4, 48]. Structural complexity theory provides good evidence showing that GI is hardly NP-complete; see the monograph [51]. The best known algorithm for GI, worked out by Babai, Luks, and Zemlyachenko [6], has moderately exponential running time $2^{O(\sqrt{n \log n})}$. Here and throughout, n denotes the number of vertices in an input graph. The best known lower bound is also surprisingly weak. Currently we do not even know if GI is P-hard under logspace reductions. Torán [71] shows that the problem is at least as hard as computing the determinant of an integer matrix (which in terms of complexity classes implies DET-hardness.)

*Supported by DFG grant KO 1053/7–1.
†Supported by DFG grant VE 652/1–1. On leave from the Institute for Applied Problems of Mechanics and Mathematics, Lviv, Ukraine.

In view of the fact that the general graph isomorphism problem has so far resisted all efforts to solve it more efficiently, it is natural to investigate its restrictions to particular classes of graphs or to reconsider the problem in other computational paradigms. An example of research in the latter direction is the search for parameters making GI fixed-parameter tractable [46, 28, 76, 70, 52, 68]. An interesting open problem in this area is whether or not GI is fixed-parameter tractable with respect to tree-width [45].

GI for particular classes of graphs has a rich literature. A systematic overview can be found in the monograph [69]. Like in the theory of NP-completeness, two cases can be distinguished: *isomorphism-complete* graph classes, for which the problem remains as hard as in general, and *isomorphism-tractable* graph classes, for which it is solvable in polynomial time. As another resemblance to the theory of NP-completeness, a dichotomic phenomenon can be observed: just a few classes of graphs are discussed in the literature for which neither isomorphism-completeness nor polynomial-time solvability is known; the most prominent examples are the classes of circular-arc and trapezoid graphs (see [41, 21] for discussions of the former and [69, 75] for the latter).

Well-known examples for isomorphism-complete classes include bipartite and chordal graphs; see [13] for a comprehensive list of other basic examples and [7, 8, 75, 74] for some more advanced results.

A very powerful tractability result is recently obtained by Grohe and Marx [36] who showed that GI is solvable in polynomial time for each class of graphs excluding a fixed topological subgraph. This includes graphs of bounded vertex degree and graphs excluding a fixed minor. The polynomial-time algorithm by Luks [58] for the former case is used in [36] as a subroutine. An earlier polynomial-time algorithm for the latter case was designed by Ponomarenko [63]; see also [35]. Furthermore, examples of minor-free classes include graphs embeddable into a fixed surface (earlier polynomial-time algorithms are due to [29, 60, 34]) and graphs of bounded tree-width (an earlier polynomial-time algorithm is due to [12]).

The tractable cases of GI admit a finer classification through the computational concepts of polylogarithmic parallel time or logarithmic space (logspace for short). The first, and very important, logspace isomorphism algorithm was designed by Lindell for trees [56].

Let L denote the class of recognition problems solvable in logspace. Recall the hierarchy of low-complexity classes:

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{LOGCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{TC}^1 \subseteq \mathsf{NC}^2, \qquad \mathsf{NL} \subseteq \mathsf{DET} \subseteq \mathsf{TC}^1.$$

Note that L occupies a lower position than DET. Thus, Lindell's algorithm for trees, together with Torán's lower DET-bound for the general isomorphism problem, implies that isomorphism of trees is strictly easier than isomorphism of all graphs unless, for instance, $\mathsf{NL} = \mathsf{L}$. Somewhat surprisingly, the same conclusion holds for a number of much broader classes of graphs, in particular, for planar and interval graphs.

A graph is interval if its vertices can be assigned to intervals such that two vertices are adjacent if and only if their intervals have non-empty intersection. Interval graphs have received persistent interest over the decades, finding applications (amongst others) in scheduling and computational biology; see e.g. [33]. Recognition of interval graphs played for example a role in establishing the linear structure of DNA (Benzer [10]).

Classifying classes of graphs as isomorphism-complete or polynomial-time solvable, an interesting phenomenon occurs: Once a particular isomorphism problem is put in P, it can often be put also in NC and, even more, in L. Examples of such a double jump are given by two classical classes of graphs, namely planar graphs (polynomial-time algorithm by Hopcroft and Tarjan [39], parallel AC^1 algorithm by Miller and Reif [61], logspace algorithm by Datta et al. [23]; see also survey [72]) and interval graphs (linear-time algorithm by Lueker and Booth [57], parallel AC^2 algorithm by Klein [47], logspace algorithm by Köbler et al. [49]). For graphs with bounded tree-width, the transition from P (Bodlaender [12]) to TC^1 was made by Grohe and Verbitsky [37], while the membership of this problem in L remains open. An important step towards this goal was made by Das, Torán, and Wagner [22] who put the problem in LOGCFL. A logspace isomorphism algorithm is known in the particular case of k-trees (Arvind et al. [2]). The "new wave" of logspace results on GI includes also the isomorphism test in [24] for graphs excluding one of the Kuratowski graphs K_5 and $K_{3,3}$ as a minor.

Note that in all cases where the isomorphism problem is solvable in logspace, we actually have an L-completeness result. For trees it was obtained by Jenner et al. [43].

A linear-time algorithm is also known for the isomorphism problem of planar graphs (Hopcroft and Wong [40]). It should be stressed that linear-time bounds are formally incomparable with L or NC bounds. On the other hand, the membership of a computational problem in L implies the existence of logarithmic time parallel algorithms for this problem (and then the next, practically important task is to minimize the number of processors in such an algorithm).

The remaining part of this survey is organized as follows. In Section 2 we establish several useful connections between graphs and hypergraphs. In Section 3 we describe recent logspace algorithms for computing canonical representations for interval graphs, proper interval graphs and some special classes of circular-arc graphs. We conclude this survey with a study of the

Star System Problem and its connections to isomorphism testing. In Section 4 we observe that some known polynomial-time tractable cases of the Star System Problem can even be solved in logspace.

2 Graphs and hypergraphs

Recall that a *hypergraph* is a pair $(V, \mathcal{H})$, where V is a set of vertices and $\mathcal{H}$ is a family of subsets of V, called *hyperedges*. A graph is a hypergraph whose hyperedges are all of size 2. We will use the same notation $\mathcal{H}$ to denote a hypergraph and its hyperedge set; this causes no ambiguity if V has no *isolated* vertex (i.e., a vertex that is not contained in any hyperedge). The vertex set V of $\mathcal{H}$ will be denoted by $V(\mathcal{H})$. An *isomorphism* from a hypergraph $\mathcal{H}$ to a hypergraph $\mathcal{K}$ is a bijection $\phi\colon V(\mathcal{H}) \to V(\mathcal{K})$ such that $X \in \mathcal{H}$ if and only if $\phi(X) \in \mathcal{K}$ for every $X \subseteq V(\mathcal{H})$. We allow multiple hyperedges; therefore, ϕ must also respect the multiplicity of every hyperedge X.

Hypergraphs can be used to represent certain graphs in a succinct way. For example, we can associate with a hypergraph $\mathcal{H}$ the *intersection graph* $\mathbb{I}(\mathcal{H})$ on vertex set $\mathcal{H}$ where $X \in \mathcal{H}$ and $Y \in \mathcal{H}$ are adjacent if and only if they have a non-empty intersection. We call a hypergraph *connected* if its intersection graph is connected. Of course,

$$\mathcal{H} \cong \mathcal{K} \implies \mathbb{I}(\mathcal{H}) \cong \mathbb{I}(\mathcal{K}), \tag{1}$$

but the converse implication does not hold in general.

Another graph that can be derived from a hypergraph $\mathcal{H}$ is the *incidence graph* $I(\mathcal{H})$. This is a colored bipartite graph with the class of red vertices $V(\mathcal{H})$, the class of blue vertices $\mathcal{H}$, and edges between all $v \in V(\mathcal{H})$ and $X \in \mathcal{H}$ such that $v \in X$. A hyperedge X of multiplicity k contributes k blue vertices in $I(\mathcal{H})$ (with the same adjacency pattern to the red part of the graph). In contrast to the intersection graph, the incidence graph contains full information about the underlying hypergraph, as we have

$$\mathcal{H} \cong \mathcal{K} \iff I(\mathcal{H}) \cong I(\mathcal{K}). \tag{2}$$

This equivalence reduces testing isomorphism of hypergraphs with n vertices and m hyperedges to testing isomorphism of colored graphs with $n + m$ vertices (where colors can, in fact, be removed by using standard gadgets, for example, by connecting all red vertices to an auxiliary triangle). Although hypergraph isomorphism reduces to GI, in many cases it is preferable to solve it directly; see [59, 5, 3] for the currently best algorithms. Though these algorithms have an exponential running time, it turns out that some interesting cases of GI can be solved efficiently by reducing them to the isomorphism problem for related hypergraphs.

An inclusion-maximal clique in a graph G will be called *maxclique*. The *bundle hypergraph* $\mathcal{B}(G)$ has one node for each maxclique of G, and for each vertex v of G a hyperedge B_v consisting of all maxcliques that contain v. We call B_v the *(maxclique) bundle* of v. Since two vertices are adjacent if and only if they are contained in a common maxclique, G is isomorphic to the intersection graph of the bundle hypergraph $\mathbb{I}(\mathcal{B}(G))$. Hence, (1) implies that

$$G \cong H \iff \mathcal{B}(G) \cong \mathcal{B}(H). \tag{3}$$

Unlike (2), the equivalence (3) does not yield an efficient reduction in general (because a graph can have up to $3^{n/3}$ maxcliques [62]). However, it does in the case of interval graphs; see Section 3.1.

We notice that the bundle hypergraph $\mathcal{B}(G)$ is the dual of the clique hypergraph of G. The *clique hypergraph* $\mathcal{C}(G)$ of a graph G has the same vertex set as G and the maxcliques of G as its hyperedges. The *dual* of a hypergraph $\mathcal{H}$ is the hypergraph $\mathcal{H}^* = \{ v^* : v \in V(\mathcal{H})\}$ on vertex set $\mathcal{H}$, where $v^* = \{ X \in \mathcal{H} : v \in X\}$ consists of all hyperedges in $\mathcal{H}$ containing v. Thus, taking the dual of a hypergraph corresponds to transposing its incidence matrix.

Twins in a hypergraph are two vertices such that every hyperedge contains either both or none of them. A hyperedge $X \in \mathcal{H}$ of multiplicity k contributes k twin vertices in the dual hypergraph $\mathcal{H}^*$. Conversely, if v and u are twins in $\mathcal{H}$, then $v^* = u^*$, and therefore, any class of k twins in $\mathcal{H}$ contributes a hyperedge of multiplicity k in $\mathcal{H}^*$. Clearly, the duals of isomorphic hypergraphs are again isomorphic. Since the two hypergraphs $\mathcal{H}$ and $(\mathcal{H}^*)^*$ are isomorphic via the mapping $x \mapsto x^*$, it follows that the converse implication is also true, implying that

$$\mathcal{H} \cong \mathcal{K} \iff \mathcal{H}^* \cong \mathcal{K}^*. \tag{4}$$

Other useful hypergraphs that can be associated with a graph G are the *open* and *closed neighborhood hypergraphs*, denoted by $\mathcal{N}(G)$ and $\mathcal{N}[G]$, respectively. The *open neighborhood* of a vertex v in G consists of all vertices adjacent to v and is denoted by $N(v)$, whereas the vertex set $N[v] = N(v) \cup \{v\}$ is called the *closed neighborhood* of v. Both hypergraphs $\mathcal{N}(G)$ and $\mathcal{N}[G]$ have the same vertex set as G and the open (resp. closed) neighborhoods of these vertices as hyperedges, i.e., $\mathcal{N}[G] = \{N[v]\}_{v \in V(G)}$ and $\mathcal{N}(G) = \{N(v)\}_{v \in V(G)}$. Note that in contrast to $\mathcal{N}[G]$, which never contains isolated vertices, $\mathcal{N}(G)$ inherits all isolated vertices from G.

If $N[u] = N[v]$, we call the vertices u and v *twins in the graph* G. Note that u and v are twins in G if and only if they are twins in $\mathcal{N}[G]$. Note also that the closed neighborhoods of twins are equal and form a hyperedge of multiplicity greater than one in the hypergraph $\mathcal{N}[G]$. Twins in a graph are always adjacent and their bundles $B_u = B_v$ coincide, implying that the hyperedge $B_u = B_v$ in $\mathcal{B}(G)$ is also of multiplicity greater than one. Of course,

$$G \cong H \implies \mathcal{N}[G] \cong \mathcal{N}[H]. \tag{5}$$

The converse implication is not true in general; for an example see Section 4.2. However, it holds true (and is useful) for proper interval graphs; see Corollary 4.4. The applicability of the relationship $\mathcal{N}[G] \cong \mathcal{N}[H]$ vs. $G \cong H$ (stated in the language of matrices) for isomorphism testing for particular graph classes was discovered and exploited by Chen [16, 17, 18].

In more generality we will discuss the conditions under which implication (5) can be reversed in Section 4.

3 Canonical representations for intersection graphs

We call a hypergraph $\mathcal{H}$ an *intersection model* of a graph G, if G is isomorphic to the intersection graph $\mathbb{I}(\mathcal{H})$. Any isomorphism from G to $\mathbb{I}(\mathcal{H})$ is called a *representation* of G by an intersection model. Every graph possesses an intersection model since, as mentioned above,

$$G \cong \mathbb{I}(\mathcal{B}(G)) \tag{6}$$

via the isomorphism $v \mapsto B_v$. When we put natural restrictions to intersection models, we obtain special classes of intersection graphs. Classical examples are interval graphs (having intervals on a line as their intersection models), circular-arc graphs (arcs on a circle), circle graphs (chords of a circle), permutation graphs (segments with endpoints in two opposite parallel lines), and trapezoid graphs (trapezoids with sides in two opposite parallel lines).

In order to represent interval graphs and circular-arc graphs by intersection models it is more convenient to use integer intervals and arcs on a discrete cycle. We refer to these intersection models as *interval models* and *arc models*, respectively. It is not hard to see that this convention does not affect the resulting graph classes.

The *canonical representation problem* for a class $\mathcal{C}$ of intersection graphs is defined as follows: For a given graph G, either compute a representation ρ_G of G by an appropriate intersection model (if $G \in \mathcal{C}$) or determine that no such model exists (if $G \notin \mathcal{C}$). Moreover, it is required that isomorphic graphs $G \cong H$ in $\mathcal{C}$ receive identical intersection models $\rho_G(G) = \rho_H(H)$. For a specified algorithm, we call its output ρ_G on input $G \in \mathcal{C}$ a *canonical representation* of G and the resulting model $\rho_G(G)$ a *canonical model* of G. Note that such an algorithm simultaneously solves both the recognition (even model construction) and the isomorphism (even canonical labeling) problems for $\mathcal{C}$.

We quickly recall the *canonical labeling problem* for a graph class $\mathcal{C}$. Given a graph $G \in \mathcal{C}$ with n vertices, we have to compute a map $\lambda_G \colon V(G) \to \{1, \dots, n\}$ so that the graph $\lambda_G(G)$, the image of G under λ_G on the vertex set $\{1, \dots, n\}$, is the same for isomorphic input graphs. Equivalently, for any graph $G \in \mathcal{C}$ we have to compute an isomorphism λ_G from G to a graph $G^\star$ so that

$$G \cong H \implies G^\star = H^\star.$$

In fact, the condition $V(G^\star) = \{1, \dots, n\}$ requires no special care as the vertices of $G^\star$ can be sorted and renamed. We say that λ_G is a *canonical labeling* and $\lambda_G(G)$ is a *canonical form* of G.

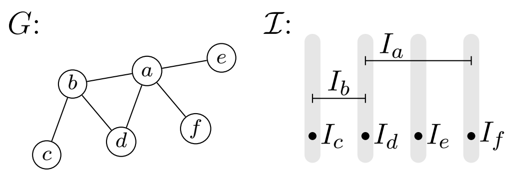
Note the similarity between the pairs of notions *canonical labeling/canonical form* and *canonical representation/canonical model* for a class of intersection graphs. Obviously, the former can be obtained from the latter by taking the intersection graph of the canonical model.

3.1 Interval graphs

In this section we describe the logspace algorithm of [49] that computes a canonical interval model for any given interval graph G. The algorithm first transforms G into its bundle hypergraph $\mathcal{B}(G)$ over the vertex set consisting of all maxcliques of G. The maxcliques of G can be found in logspace by applying the following lemma.

Lemma 3.1 (Laubner [54]). *Every maxclique C of an interval graph G contains vertices u and v such that $C = N[u] \cap N[v]$.*

For adjacent vertices u and v in an arbitrary graph holds: If $N[u] \cap N[v]$ is a clique, it is maximal. Lemma 3.1 shows that any maxclique in an interval graph is of this kind and, hence, can be represented by a pair of vertices u and v (that are adjacent and satisfy the condition that $N[u] \cap N[v]$ is a clique). An explicit representation of the bundle hypergraph $\mathcal{B}(G)$ of G can be computed in logspace by listing, for each bundle B_v, the maxcliques that contain v.



$\mathcal{B}(G)$	$\{b, c\}$	$\{a, b, d\}$	$\{a, e\}$	$\{a, f\}$
B_a	0	1	1	1
B_b	1	1	0	0
B_c	1	0	0	0
B_d	0	1	0	0
B_e	0	0	1	0
B_f	0	0	0	1

Figure 1: An interval graph G, a minimal interval model $\mathcal{I}$ of G, and the bundle hypergraph $\mathcal{B}(G)$ of G. The latter is given by its incidence matrix with columns indexed by vertices (i.e., maxcliques) and rows indexed by hyperedges (i.e., bundles).

The following lemma shows that the bundle hypergraph $\mathcal{B}(G)$ is indeed a good starting point for constructing an interval model for a given graph G. We call an interval model $\mathcal{I}$ of G *minimal* if G admits no interval model that has fewer points than $\mathcal{I}$.

Lemma 3.2 ([49, Lemma 2.3])**.** *Every minimal interval model $\mathcal{I}$ of an interval graph G is isomorphic to the bundle hypergraph $\mathcal{B}(G)$.*

Lemma 3.2 implies that the minimal interval model of G is unique up to hypergraph isomorphism, and any such model can be obtained from the bundle hypergraph $\mathcal{B}(G)$ by renaming its vertices to integers; Fig. 1 shows an example.

Given a hypergraph $\mathcal{H}$, call a linear order $<$ on $V(\mathcal{H})$ *interval* if every hyperedge of $\mathcal{H}$ forms an interval w.r.t. $<$. If $\mathcal{H}$ admits an interval order, then it is called an *interval hypergraph.* Equivalently, an interval hypergraph is a hypergraph isomorphic to a system of intervals of integers, which is then called an *interval model* of this hypergraph. For an interval order $<$ of $\mathcal{H}$, let $r_<(v)$ denote the rank of a vertex $v \in V(\mathcal{H})$ w.r.t. $<$, and let $\mathcal{H}^<$ denote the image of $\mathcal{H}$ under the map $r_<$. Clearly, $<$ is an interval order of $\mathcal{H}$ if and only if $\mathcal{H}^<$ is an interval model of $\mathcal{H}$ on the segment of integers $\{1, \ldots, n\}$.

A binary matrix has the *consecutive-ones property* if its columns can be permuted so that in each row the ones are consecutive. Viewing the matrix as incidence matrix of a hypergraph shows that testing for the consecutive-ones property is equivalent to recognizing interval hypergraphs. Dom [26] surveys algorithmic aspects of the consecutive-ones property.

The following theorem is an immediate consequence of Lemma 3.2 and the general relation (6).

Theorem 3.3 (cf. [33, Theorems 8.1 and 8.3])**.** *G is an interval graph if and only if $\mathcal{B}(G)$ is an interval hypergraph.*

Hence, in order to decide whether G is interval, it suffices to check whether the bundle hypergraph $\mathcal{B}(G)$ is interval.

Moreover, from any interval ordering $<$ of $\mathcal{B}(G)$, we an easily construct an interval representation ρ_G. For any vertex $v \in V(G)$, define $\rho_G(v) = r_<(B_v)$. Since the mapping $v \mapsto B_v$ is a graph isomorphism from G to $\mathbb{I}(\mathcal{B}(G))$ and $r_<$ is a hypergraph isomorphism from $\mathcal{B}(G)$ to the interval system $\mathcal{B}(G)^<$, the map ρ_G is an isomorphism from G to $\mathbb{I}(\mathcal{B}(G)^<)$. Hence, ρ_G is indeed an interval representation of G.

Since the bundle hypergraph $\mathcal{B}(G)$ and the map $v \mapsto B_v$ are constructible in logspace due to Lemma 3.1, it follows that ρ_G is computable in logspace, provided that an interval ordering $<_{\mathcal{H}}$ for a given interval hypergraph $\mathcal{H}$ is computable in logspace. Moreover, this reduction even gives a canonical interval representation ρ_G of G, if $<_{\mathcal{H}}$ is a *canonical ordering* of $\mathcal{H}$, meaning that $\mathcal{H}^{<_{\mathcal{H}}} = \mathcal{K}^{<_{\mathcal{K}}}$ whenever $\mathcal{H} \cong \mathcal{K}$ are isomorphic interval hypergraphs. Indeed, $G \cong H$ implies that $\mathcal{B}(G) \cong \mathcal{B}(H)$ and hence the resulting interval systems $\mathcal{B}(G)^{<_{\mathcal{B}(G)}}$ and $\mathcal{B}(H)^{<_{\mathcal{B}(H)}}$ are equal.

Lemma 3.4. *The canonical representation problem for interval graphs is reducible in logspace to the canonical ordering problem for interval hypergraphs.*

Computing an interval ordering for interval hypergraphs

In this subsection, we describe an algorithm for computing an interval ordering for a given interval hypergraph $\mathcal{H}$ (or detecting that $\mathcal{H}$ is not interval).

PQ-trees, introduced by Booth and Lueker [14], provide a succinct way to represent all possible interval orderings of an interval hypergraph $\mathcal{H}$. A *PQ-tree* for $\mathcal{H}$ is an ordered rooted tree. Its leaves are the vertices of $\mathcal{H}$ and its inner nodes are classified as either P- or Q-nodes. Clearly, the ordering of the tree induces a unique linear order on the leaves of the tree. It is possible to change the tree order of a PQ-tree according to the following rules. The children of a P-node can be reordered arbitrarily, while the ordering of the children of a Q-node can only be reversed. A tree order is *permissible* if it can be obtained from a combination of such reorderings. A PQ-tree represents the set of all linear orders on its leaves which are induced by a permissible tree order.

Booth and Lueker [14] showed that a PQ-tree encoding all interval orderings of a given interval hypergraph can be computed in linear time. Their algorithm starts with the PQ-tree T for the empty hypergraph (where all leaves are attached to a single P-node). Then it iteratively incorporates into T the restrictions caused by each hyperedge. Klein [47] reduced the number of iterations from linear to logarithmic by incorporating several hyperedges in one step. This results in a parallel AC^2 algorithm. In [49] it was shown that a PQ-tree for a given interval hypergraph $\mathcal{H}$ can even be computed in logspace. The key observation behind this is that the *overlap component tree* of $\mathcal{H}$ can be viewed as PQ-tree and is constructible in logspace. This tree comprises of slot nodes, which are interpreted as P-nodes, and overlap component nodes, which are interpreted as Q-nodes. To give its precise definition we need some more notation.

We say that two sets A and B *overlap* and write $A \between B$ if A and B have a nonempty intersection but neither of them includes the other. The *overlap graph* $\mathbb{O}(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is the subgraph of the intersection graph $\mathbb{I}(\mathcal{H})$, where the vertices corresponding to the hyperedges A and B are adjacent if and only if they overlap. Of course, $\mathbb{O}(\mathcal{H})$ can be disconnected even if $\mathbb{I}(\mathcal{H})$ is connected. A subset $\mathcal{O}$ of the hyperedges of $\mathcal{H}$ spanning a connected component of $\mathbb{O}(\mathcal{H})$ will be referred to as an *overlap component* of $\mathcal{H}$. This is a subhypergraph of $\mathcal{H}$ and should not be confused with the corresponding induced subgraph of $\mathbb{O}(\mathcal{H})$. Note that a hyperedge of an overlap component inherits the multiplicity that it has in $\mathcal{H}$. In the case that $\mathbb{O}(\mathcal{H})$ is connected, $\mathcal{H}$ will be called an *overlap-connected hypergraph*.

Lemma 3.5 (Chen and Yesha [19]). *Let $\mathcal{H}$ be an interval hypergraph. If $\mathcal{H}$ is overlap-connected, then it has, up to permutation of twins and reversing, a unique interval ordering.*

Since the resulting interval model does not depend on the ordering of twins inside a slot, it follows that an overlap-connected interval hypergraph has at most two different interval models inside the range $\{1, \ldots, n\}$ and that these models are mirror symmetric to each other.

In fact, such a model can be constructed in logspace as follows. In a pre-processing step, compute a walk $X_1, \ldots, X_N$ in the overlap graph $\mathbb{O}(\mathcal{H})$ that visits every hyperedge of $\mathcal{H}$ at least once (this can be done in logspace using Reingold's universal exploration sequences [66]). Then iterate over the hyperedges X_i in this walk, computing an interval I_i for each X_i. Once the first interval $I_1 = [1, |X_i|]$ is fixed, the cardinality of $X_1 \cap X_2$ leaves only two possibilities for I_2 (resulting in reflected representations), and once I_{i-2} and I_{i-1} are fixed, I_i is uniquely determined; see Fig. 2. As only the two previous intervals have to be remembered, this computation is possible in logspace. In a post-processing step, verify that the result is indeed an interval model of $\mathcal{H}$ and shift the model into the range $\{1, \ldots, n\}$ if necessary.

A *slot* of $\mathcal{H}$ is an inclusion-maximal set S of twins, i.e., the slots are the equivalence classes of the twin relation.

If $\mathcal{O}$ and $\mathcal{O}'$ are different overlap components, then either every two hyperedges $A \in \mathcal{O}$ and $A' \in \mathcal{O}'$ are disjoint or all hyperedges of one of the

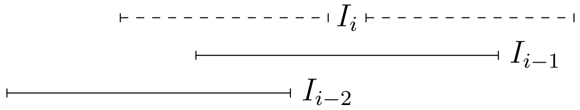


Figure 2: Proof of Lemma 3.5: Let $X_i \between X_{i-1} \between X_{i-2}$. If I_{i-1} is already determined, only two positions for I_i satisfy $|I_i| = |X_i|$ and $|I_{i-1} \cap I_i| = |X_{i-1} \cap X_i|$. If also I_{i-2} is given, at most one of them additionally satisfies $|I_{i-2} \cap I_i| = |X_{i-2} \cap X_i|$.

two components are contained in a single slot of the other component. (This follows from the simple observation that the conditions $B \subset A$, $B \between B'$, and $\neg(B' \between A)$ imply $B' \subset A$.) This containment relation determines a tree-like decomposition of $\mathcal{H}$ into its overlap components. Specifically, let S be a slot of an overlap component $\mathcal{O}$ of $\mathcal{H}$. We say that an overlap component $\mathcal{Q}$ of $\mathcal{H}$ is *located at* slot S of $\mathcal{O}$ if $V(\mathcal{Q}) \subseteq S$ and there is no "intermediate" overlap component $\mathcal{O}' \neq \mathcal{O}$ such that $V(\mathcal{O}') \subseteq S$ and $\mathcal{Q}$ is contained in some slot of $\mathcal{O}'$. Furthermore, a vertex v of $\mathcal{H}$ is *located at* slot S of $\mathcal{O}$ if $v \in S$ and there is no overlap component $\mathcal{O}'$ located at slot S of $\mathcal{O}$ such that $v \in V(\mathcal{O}')$.

Now we are ready to give a precise definition of the overlap component tree of an interval hypergraph $\mathcal{H}$. We assume that $\mathcal{H}$ is connected: To ensure this, we add an additional hyperedge $B_0 = V(\mathcal{H})$ (this has no influence on the possible interval orderings of $\mathcal{H}$).

The nodes of the overlap component tree of $\mathcal{H}$ are the overlap components of $\mathcal{H}$, their slots, and the vertices of $\mathcal{H}$. Since a slot S of $\mathcal{O}$ may belong also to another overlap component, we denote the corresponding slot node by $S_{\mathcal{O}}$. The children of an overlap component node $\mathcal{O}$ are the slots of $\mathcal{O}$. The children of a slot node $S_{\mathcal{O}}$ are the vertices and the overlap components located at the slot S of $\mathcal{O}$. As $\mathcal{H}$ is connected, there is an overlap component $\mathcal{O}_0$ with $V(\mathcal{O}_0) = V(\mathcal{H})$. Thus, $\mathcal{O}_0$ is the root of the overlap component tree. An example for an overlap component tree can be found in Fig. 3.

Suppose that $\mathcal{H}$ is an interval hypergraph. To interpret its overlap component tree as PQ-tree, treat all slot nodes as P-nodes, and all overlap component nodes as Q-nodes. By Lemma 3.5, the slots of each overlap component can be ordered uniquely up to reversing; this defines the order on the children of Q-nodes. It is easy to verify that every rearrangement of this PQ-tree again induces an interval ordering of $\mathcal{H}$. Using the uniqueness given by Lemma 3.5 and a simple inductive argument on the depth of the overlap component tree, one can show that every interval representation of $\mathcal{H}$ can be obtained in this way (cf. [42]).

It is not hard to verify that the overlap component tree (and hence a

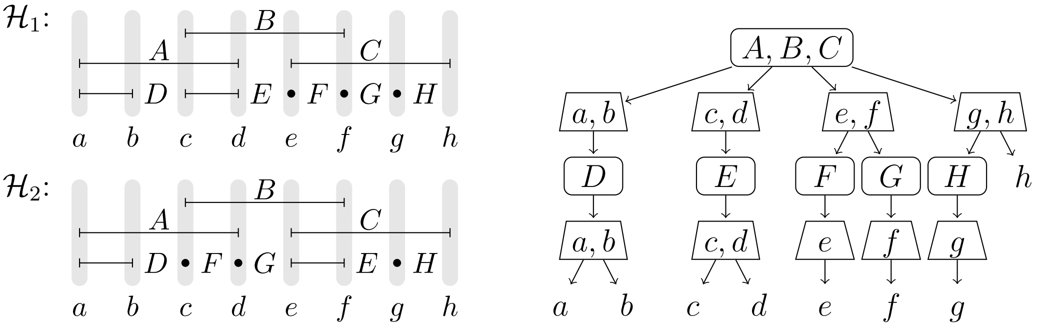


Figure 3: An interval hypergraph $\mathcal{H}_1$ and its overlap component tree. In the tree, the node of an overlap component $\mathcal{O}$ is given by listing the hyperedges in $\mathcal{O}$; a slot node $S_{\mathcal{O}}$ is given by listing the vertices contained in S (we omit the reference to $\mathcal{O}$ as it is understood from the tree structure). The hypergraph $\mathcal{H}_2$ is not isomorphic to $\mathcal{H}_1$; yet both have isomorphic overlap component trees.

PQ-tree) for a given interval hypergraph $\mathcal{H}$ can be computed in logspace. Thus, we can compute an interval ordering for $\mathcal{H}$ in logspace. In order to compute a canonical interval ordering for $\mathcal{H}$ we have to do some more work.

Of course, isomorphic interval hypergraphs have isomorphic overlap component trees (when considered as rooted trees but ignoring the order on the children). As shown in Fig. 3, the converse is not true, since the overlap component tree does not contain enough information on the underlying hypergraph. The idea is to reflect this missing information by coloring the nodes of the tree in such a way that the underlying hypergraph can be reconstructed up to isomorphism from the resulting tree. Roughly speaking, we will refine the tree in such a way that only rearrangements of the corresponding PQ-tree become isomorphic to the refined tree. Then selecting an interval ordering in a canonical way corresponds to selecting an isomorphic copy of the refined tree in a canonical way. For the latter task we can use Lindell's tree canonization algorithm.

Computing a canonical ordering for interval hypergraphs

In this subsection, we describe an algorithm for computing canonical orderings for interval hypergraphs. Together with Lemma 3.4 this gives us a logspace algorithm for computing a canonical representation for interval graphs.

Theorem 3.6 ([49, Theorem 4.6])**.** *The canonical ordering problem for interval hypergraphs can be solved in logspace.*

Corollary 3.7. *The canonical representation problem for interval graphs is solvable in logspace.*

As explained above, we reduce the task of computing a canonical interval ordering for $\mathcal{H}$ to computing a canonical labeling of an associated tree $\mathbb{T}(\mathcal{H})$. This tree has the property that $\mathcal{H} \cong \mathcal{K}$ if and only if $\mathbb{T}(\mathcal{H}) \cong \mathbb{T}(\mathcal{K})$. To construct $\mathbb{T}(\mathcal{H})$ from the overlap component tree of $\mathcal{H}$, we first compute canonical interval models for each overlap component (using the lexicographically smaller of the at most two that are possible by Lemma 3.5) and assign these models as colors to the overlap component nodes. For an asymmetric overlap component, the chosen model already fixes the order of its slots, which can be enforced by assigning ascending colors to the slot nodes. For a symmetric overlap component with at most two slots, any ordering of its slots is fine; for one with more than two slots, we employ a small gadget to ensure that the order of its slots can only be reflected: Between the overlap component node $\mathcal{O}$ and its slots, we introduce three connector nodes $lo_{\mathcal{O}}$, $mi_{\mathcal{O}}$ and $hi_{\mathcal{O}}$. Fix an arbitrary interval order $<$ of $\mathcal{O}$; it induces an order $<^*$ on the slots of $\mathcal{O}$. For each slot S of $\mathcal{O}$, denote its position from the left and right by

$$l_{\mathcal{O}}(S) = |\{\, S' : S' \text{ is a slot of } \mathcal{O} \text{ with } S' \leq^* S\}|$$
$$r_{\mathcal{O}}(S) = |\{\, S' : S' \text{ is a slot of } \mathcal{O} \text{ with } S' \geq^* S\}|$$

A slot node $S_{\mathcal{O}}$ becomes a child of $lo_{\mathcal{O}}$ if $l_{\mathcal{O}}(S) < r_{\mathcal{O}}(S)$, a child of $mi_{\mathcal{O}}$ if $l_{\mathcal{O}}(S) = r_{\mathcal{O}}(S)$, and a child of $hi_{\mathcal{O}}$ if $l_{\mathcal{O}}(S) > r_{\mathcal{O}}(S)$. (Choosing a different interval ordering for $\mathcal{O}$ would only result in exchanging the nodes $lo_{\mathcal{O}}$ and $hi_{\mathcal{O}}$, yielding an isomorphic tree.) Fig. 4 shows an example for a tree representation $\mathbb{T}(\mathcal{H})$. As indicated above, this construction ensures that a canonical labeling of $\mathbb{T}(\mathcal{H})$ specifies a canonical rearrangement of the PQ-tree, which in turn determines a canonical interval order of $\mathcal{H}$ (see [49] for details).

3.2 Proper interval graphs

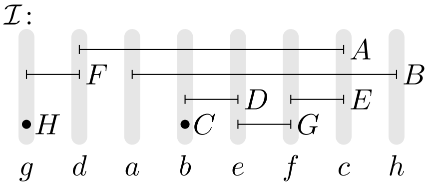
An intersection model $\mathcal{H}$ is *proper* if the sets in $\mathcal{H}$ are pairwise incomparable by inclusion. G is called a *proper interval graph* if it has a proper interval model. In this section, we describe a logspace algorithm for computing a canonical proper interval model for a given proper interval graph.

As a consequence of the following theorem, we can reduce the problem of recognizing proper interval graphs to the problem of recognizing interval hypergraphs.

Theorem 3.8 (Roberts [67, 27])**.** *G is a proper interval graph if and only if $\mathcal{N}[G]$ is an interval hypergraph.*

$\mathcal{H}$	a	b	c	d	e	f	g	h
A	1	1	1	1	1	1	0	0
B	1	1	1	0	1	1	0	1
C	0	1	0	0	0	0	0	0
D	0	1	0	0	1	0	0	0
E	0	0	1	0	0	1	0	0
F	0	0	0	1	0	0	1	0
G	0	0	0	0	0	0	1	0

$\mathcal{I}$:



$\mathbb{T}(\mathcal{H})$:

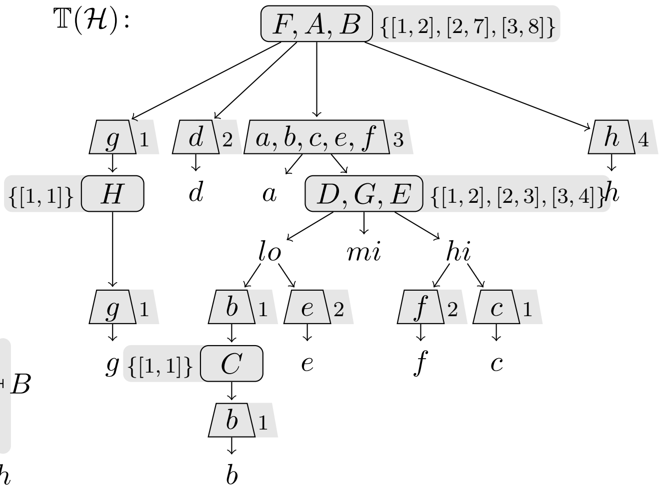


Figure 4: An interval hypergraph $\mathcal{H}$ and its tree representation $\mathbb{T}(\mathcal{H})$. Gray areas in $\mathbb{T}(\mathcal{H})$ indicate the color of overlap components and their slots. An overlap component $\mathcal{O}$ is represented by listing the hyperedges in $\mathcal{O}$ (sorted, for the reader's convenience, by their corresponding intervals in the canon of $\mathcal{O}$). We omit the references from slot and connector nodes to their overlap components as they are understood from the tree structure. The interval system $\mathcal{I} \cong \mathcal{H}$ can be derived from $\mathbb{T}(\mathcal{H})$ by reading the vertex nodes from left to right.

Theorem 3.8 does not provide us with an appropriate interval model for a proper interval graph, since $\mathcal{N}[G]$ need not even be an intersection model for G. However, it is possible to convert an interval ordering of $\mathcal{N}[G]$ into a proper interval model for G, if G is a proper interval graph. This is done via a tight interval model for G.

An interval system is *tight* if the intervals have the following property: whenever $A = [a^-, a^+]$ includes $B = [b^-, b^+]$, we have $a^- = b^-$ or $a^+ = b^+$. It is not hard to see that any tight interval model for a graph G can be converted to a proper interval model for G (cf. Tucker [73]): If several intervals start (resp. end) at the same point, introduce new points to extend the shorter intervals so that none is contained in the other anymore. In fact, this transformation is possible in logspace (see [50] for details). Thus, the following lemma provides us with a proper interval representation of G.

Lemma 3.9. *Given an interval ordering of $\mathcal{N}[G]$, a tight interval representation of G can be constructed in logspace.*

Proof. Given an interval ordering $<$ of $\mathcal{N}[G]$, for each vertex v of G we let v^- and v^+ denote the two endpoints of the interval $N[v] = [v^-, v^+]$ w.r.t. $<$.

Define $\rho(v) = N^+[v] = [v, v^+]$. The map ρ is an interval representation of G. Indeed, if u and v are adjacent, either $u \in N^+[v]$ (if $v < u$) or $v \in N^+[u]$ (if $u < v$) holds. In either case $N^+[u] \cap N^+[v] \neq \emptyset$. If u and v are not adjacent, $u \notin N^+[v]$ and $v \notin N^+[u]$, which implies that the intervals $N^+[u]$ and $N^+[v]$ are disjoint. See Fig. 5 for an example.

Next we show that ρ is tight. Suppose that $N^+[u] \subseteq N^+[v]$. Since $v \in N[v^+]$, we have also $u \in N[v^+]$. Therefore, $N^+[u] = [u, u^+]$ contains v^+ and $u^+ = v^+$.

Finally, given G and an interval ordering $<$ of $\mathcal{N}[G]$, the map ρ can be easily computed in logspace. □

To obtain a canonical proper interval representation for the class of proper interval graphs, we can combine any canonical labeling for this class with the proper interval representation of the resulting canon. To compute a canonical labeling for proper interval graphs we can for example use the algorithm provided by Corollary 3.7 (which even works for all interval graphs). Alternatively, since we anyway have to compute an interval ordering of $\mathcal{N}[G]$, we can also make use of the following lemma.

Lemma 3.10 (cf. [25, Corollary 2.5])**.** *If G is a connected proper interval graph, then $\mathcal{N}[G]$ has, up to reversing and up to permutation of twins, a unique interval ordering.*

This result can also be derived from Lemma 3.5 and the fact that, for a connected proper interval graph G, the hypergraph $\mathcal{N}[G] \setminus \{V(G)\}$ is overlap-connected; see [49].

As the interval order of $\mathcal{N}[G]$ can be computed in logspace by Theorem 3.6, Lemma 3.10 implies that we can easily compute canonical labelings for the connected components of a given proper interval graph G and combine them to a canonical labeling of the whole graph in a straightforward way.

This proves the following theorem.

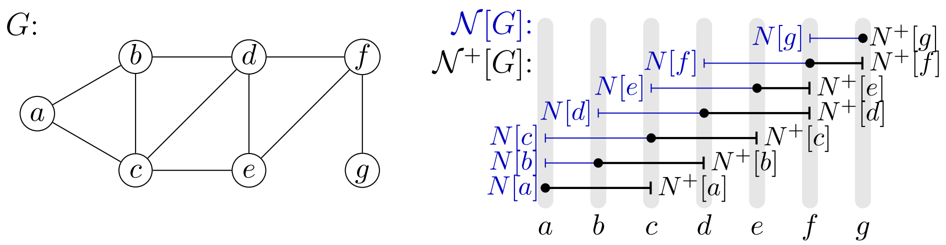


Figure 5: From an interval ordering of $\mathcal{N}[G]$ to a tight interval model $\mathcal{N}^+[G]$ of G.

Theorem 3.11 ([49, Theorem 6.3])**.** *The canonical representation problem for proper interval graphs can be solved in logspace.*

A graph is a *unit interval graph* if it has an interval model over rationals in which every interval has unit length. It is well known [67] that the class of proper interval graphs is equal to the class of unit interval graphs. Corneil et al. [20] show that unit interval representations of proper interval graphs can be computed in linear time. Based on their methods, it has been shown in [49] that this task can also be performed in logspace.

3.3 Circular-arc graphs

Though circular-arc graphs may at first glance appear close relatives of interval graphs, essential differences between the two classes are well known. For example, an interval graph has at most n maxcliques, and we used a succinct representation for each of them given by Lemma 3.1. For circular-arc graphs this is no longer possible, because these graphs can have exponentially many maxcliques; see Fig. 6 for an example. Note also that, unlike interval graphs, currently there is no characterization of the class of circular-arc graphs in terms of forbidden induced subgraphs; see [55] for an overview of circular-arc graphs and subclasses. These facts may serve as some excuse for the status of GI for circular-arc graphs staying open: Recently, Curtis et al. [21] published a counter-example to Hsu's algorithm [41], raising the following question.

Problem 3.12. *Is the isomorphism problem for circular-arc graphs in* P*?*

Furthermore, proper interval and proper circular-arc graphs also show structural distinctions. For example, while every proper interval graph is known to be representable by an intersection model consisting of unit intervals, the analogous statement for proper circular-arc graphs is not true. Another difference, very important in our context, lies in relationship to interval and circular-arc hypergraphs that we will explain shortly.

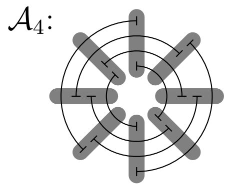


Figure 6: The complement graph G_m of m disjoint edges is circular-arc and has 2^m maxcliques. $\mathcal{A}_4$ is a circular-arc model for G_4.

A *circular ordering* of a hypergraph $\mathcal{H}$ is a circular successor relation $\succ$ such that all hyperedges $X \in \mathcal{H}$ are consecutive points w.r.t. $\succ$. A hypergraph is *circular-arc* if it admits a circular ordering.

By Theorem 3.8, G is a proper interval graph if and only if $\mathcal{N}[G]$ is an interval hypergraph. The circular-arc world is more complex. While $\mathcal{N}[G]$ is a circular-arc hypergraph if G is a proper circular-arc graph, the converse is not always true. Proper circular-arc graphs are properly contained in the class of those graphs whose neighborhood hypergraphs are circular-arc. Graphs with this property are called *concave-round* by Bang-Jensen, Huang, and Yeo [9] and *Tucker graphs* by Chen [16]. The latter name is justified by Tucker's result [73] saying that all these graphs are circular-arc (even though not necessarily proper circular-arc). Fig. 7 shows a circular-arc graph that is not concave-round.

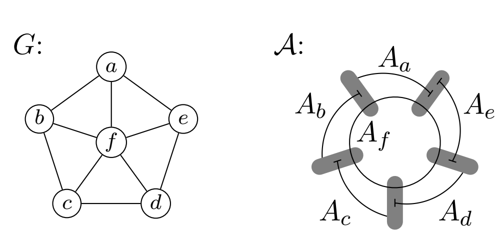
In the context of hypergraphs, however, the similarity between circular-arc and interval hypergraphs can be directly exploited, as first observed by Tucker [73]. For a circular-arc hypergraph $\mathcal{H}$ and a vertex $v \in V(\mathcal{H})$, define the hypergraph

$$\mathcal{H}_v = \{\, X : v \notin X \in \mathcal{H} \} \cup \{\, V(\mathcal{H}) \setminus X : v \in X \in \mathcal{H} \}\,.$$

This corresponds to complementing all hyperedges that contain v. Tucker observed that $\mathcal{H}_v$ is interval if and only if $\mathcal{H}$ is circular-arc. The following theorem is proved by iterating over all $v \in V(\mathcal{H})$ and distinguishing non-complemented and complemented hyperedges in $\mathcal{H}_v$ with two different colors.

Theorem 3.13 ([50])**.** *The canonical ordering problem for circular-arc hypergraphs can be solved in logspace.*

In [50] we use the algorithm of Theorem 3.13 as a starting point to design logspace algorithms for computing canonical proper circular-arc models of proper circular-arc graphs and canonical circular-arc models of concave-round graphs.



$\mathcal{N}[G]$	a	b	c	d	e	f
$N[a]$	1	1	0	0	1	1
$N[b]$	1	1	1	0	0	1
$N[c]$	0	1	1	1	0	1
$N[d]$	0	0	1	1	1	1
$N[e]$	1	0	0	1	1	1
$N[f]$	1	1	1	1	1	1

Figure 7: A circular-arc graph $G = \mathbb{I}(\mathcal{A})$ that is not concave-round: Its closed neighborhood hypergraph $\mathcal{N}[G]$ is not circular-arc.

4 Realizing Star Systems

The Star System Problem (to be abbreviated as *SSP*) consists in finding, for a given hypergraph $\mathcal{H}$, a graph G such that $\mathcal{H} = \mathcal{N}[G]$. We call any such graph G a *solution* to the SSP on input $\mathcal{H}$. Note that $\mathcal{H}$ can only have an SSP solution if $\mathcal{H}$ has an equal number of vertices and hyperedges. The terms *star* and *star system* are synonyms for the closed neighborhood of a vertex and the closed neighborhood hypergraph of a graph, respectively. The problem occurs in the literature also under the name *Closed Neighborhood Realization*. The question on the computational complexity of the SSP was posed by Sabidussi and Sós in the mid-70s. Shortly afterwards, Babai observed that the problem is at least as hard as GI; see [30] for a historical overview. Subsequently, Lalonde [53] showed that its decision version is in fact NP-complete.

In the complementary version of the SSP, called *co-SSP* here and also known as *Open Neighborhood Realization* problem in the literature, on input $\mathcal{H}$ we have to find a graph G with $\mathcal{N}(G) = \mathcal{H}$. Recall that the *complement* $\overline{G}$ *of a graph* G has the same vertex set as G, and two vertices are adjacent in $\overline{G}$ if and only if they are not in G. The *complement* $\overline{\mathcal{H}}$ *of a hypergraph* $\mathcal{H}$ also has the same vertex set as $\mathcal{H}$, but hyperedges complementing the hyperedges of $\mathcal{H}$, i.e., $X \in \overline{\mathcal{H}}$ if and only if $V(\mathcal{H}) \setminus X \in \mathcal{H}$. Now it is easy to verify that

$$\overline{\mathcal{N}[G]} = \mathcal{N}(\overline{G}) \quad \text{and} \quad \overline{\mathcal{N}(G)} = \mathcal{N}[\overline{G}]. \tag{7}$$

Hence, finding for a given hypergraph $\mathcal{H}$ a graph G with $\mathcal{N}[G] = \mathcal{H}$ is equivalent to finding for $\overline{\mathcal{H}}$ a graph G' with $\mathcal{N}(G') = \overline{\mathcal{H}}$. Thus, the SSP and the co-SSP have the same complexity.

The following simple observation characterizes open neighborhood hypergraphs of bipartite graphs.

Lemma 4.1. *Suppose that G is a connected bipartite graph with vertex classes U and W. Then the open neighborhood hypergraph $\mathcal{N}(G)$ is split into two connected components $\mathcal{U}$ and $\mathcal{W}$, on the vertex sets U and W, respectively, such that $\mathcal{U} \cong \mathcal{W}^*$.*

In the notation of the lemma, note that the incidence graphs $I(\mathcal{U})$ and $I(\mathcal{W})$ become isomorphic after interchanging the colors red and blue in one of them. Moreover, the uncolored versions of both $I(\mathcal{U})$ and $I(\mathcal{W})$ are isomorphic to G.

4.1 Case study: NP-hardness, GI-completeness, and efficient solvability

If we restrict the SSP to a particular graph class $\mathcal{C}$, we only seek for a solution G in the class $\mathcal{C}$. As mentioned above, the restriction of the SSP to $\mathcal{C}$ is equivalent to the co-SSP restricted to the co-class of $\mathcal{C}$ (consisting of the complements of all graphs in $\mathcal{C}$).

Fomin et al. [30] study the restrictions of the SSP to H-free graphs, that is, to graph classes that are characterized by forbidding a single induced subgraph H. They show that the SSP restricted to H-free graphs remains NP-hard if H is a path or a cycle of at least 5 vertices, the claw graph, or any other graph obeying a set of conditions specified in [30].

Aigner and Triesch [1] showed that the SSP for co-bipartite graphs is equivalent to GI, provided that the bipartition of the vertices is given along with the input hypergraph $\mathcal{H}$. Boros et al. [15] observed that this remains true, if only $\mathcal{H}$ is given as input.

Theorem 4.2 ([1, 15])**.** *The SSP for co-bipartite graphs is equivalent to GI.*

Proof-sketch. We show the equivalent statement that the co-SSP for bipartite graphs is equivalent to GI. Recall that GI is equivalent with its restriction to connected graphs (because $G \cong H$ if and only if $\overline{G} \cong \overline{H}$, and if G is disconnected, then its complement $\overline{G}$ must be connected). Consider an even more general problem of deciding whether two connected hypergraphs $\mathcal{H}$ and $\mathcal{K}$ are isomorphic. By (4) and Lemma 4.1,

$$\mathcal{H} \cong \mathcal{K} \iff \mathcal{H}^* \cong \mathcal{K}^* \iff \mathcal{H} \cup \mathcal{K}^* = \mathcal{N}(G) \text{ for a bipartite graph } G,$$

and hence, the reduction $(\mathcal{H}, \mathcal{K}) \mapsto \mathcal{H} \cup \mathcal{K}^*$ shows that the co-SSP for bipartite graphs is at least as hard as (hyper)graph isomorphism.

In order to show a reduction in the other direction, assume first that $\mathcal{H}$ consists of two connected components $\mathcal{U}$ and $\mathcal{W}$. By Lemma 4.1,

$$\mathcal{H} = \mathcal{N}(G) \text{ for a bipartite graph } G \iff \mathcal{U} \cong \mathcal{W}^*,$$

giving the desired reduction of the co-SSP for bipartite graphs to hypergraph isomorphism. Moreover, we can also compute a solution G to the co-SSP instance $\mathcal{H}$, since G is isomorphic to the incidence graph $G' = I(\mathcal{U}) \cong I(\mathcal{W})$, where the red-blue coloring is disregarded. In order to compute G, it suffices to establish an isomorphism π from $\mathcal{N}(G')$ to $\mathcal{H}$ and take the image of G' under π.

In general, $\mathcal{H} = \mathcal{N}(G)$ for a bipartite G if and only if the components of $\mathcal{H}$ can be arranged into pairs $\mathcal{U}_1, \mathcal{W}_1, \ldots, \mathcal{U}_m, \mathcal{W}_m$ such that $\mathcal{U}_i \cong \mathcal{W}_i^*$. Thus, the co-SSP for bipartite graphs is no harder than GI. ∎

In several cases the SSP is known to be efficiently solvable. Polynomial-time algorithms are worked out for H-free graphs with H being a cycle or a path on at most 4 vertices (Fomin et al. [30]) and for bipartite graphs (Boros et al. [15]). In [50] we give a logspace solution for the SSP for proper circular-arc and concave-round graphs. An analysis of the algorithms in [30] for C_3- and C_4-free graphs shows that the SSP for these classes is also solvable in logspace, and the same holds true for the class of bipartite graphs.

C_3-free graphs and bipartite graphs. The approach of Fomin et al. [30] to C_3-free graphs is based on the following observation. If G is C_3-free, then for any pair of vertices u and v adjacent in G there are exactly two hyperedges X and Y in $\mathcal{N}[G]$ containing both u and v. Moreover, $N[v] \in \{X, Y\}$ and the assumption that $N[v] = X$ forces the equality $N[u] = Y$.

Let us show how to derive from here the logspace solvability of the SSP for C_3-free graphs. Since the composition of logspace computable functions is logspace computable, we can split the whole algorithm into a few steps, each doable in logspace. We can assume that the input hypergraph $\mathcal{H}$ is connected; otherwise we apply the procedure below to each of its components. We first construct an auxiliary graph F. The vertices of F are all pairs (v, X) such that $v \in X \in \mathcal{H}$. Two vertices (v, X) and (u, Y) are adjacent in F if and only if $v \neq u$, $X \neq Y$, and X and Y are the only two hyperedges of $\mathcal{H}$ containing both v and u.

Fix an arbitrary vertex v of $\mathcal{H}$. For each vertex of the form (v, X) of F, we now try to construct a vertex-hyperedge assignment A_X as follows. Assign X to v. To each other u we assign an Y such that (u, Y) is reachable from (v, X) along a path in F. At this step we use the Reingold reachability algorithm [66]. For some u, the choice of Y may be impossible or ambiguous.

For each successfully constructed one-to-one assignment A_X, we then try to construct a graph G_X by connecting each u with all other vertices in the assigned hyperedge Y. For each successfully constructed G_X, it remains to check if $\mathcal{N}[G_X] = \mathcal{H}$ and if G_X is C_3-free. We will succeed at least once, unless the SSP on $\mathcal{H}$ has no C_3-free solution. This completes the description of the algorithm.

Note a useful fact that follows from the above discussion: If a hypergraph $\mathcal{H}$ is connected, then for any hyperedge $X \in \mathcal{H}$ and vertex $v \in X$ there is at most one C_3-free graph G such that $\mathcal{H} = \mathcal{N}[G]$ and $X = N[v]$. Thus, the SSP on $\mathcal{H}$ has at most $\min_{X \in \mathcal{H}} |X|$ triangle-free solutions, and all of them can be computed in logspace. It readily follows that the SSP is solvable in logspace for any logspace recognizable class consisting of C_3-free graphs. In particular, this applies to the class of bipartite graphs.

C_4-free graphs. The algorithm of Fomin et al. [30] for C_4-free graphs is implementable in logspace in a straightforward way. It is based on the following argument.

Suppose that G is C_4-free. Given two vertices u and v in G, let $X_1, \ldots, X_t$ be all hyperedges in $\mathcal{N}[G]$ containing both u and v. If u and v are adjacent, then

$$2 \le \left|\bigcap_{i=1}^{t} X_i\right| \le t. \tag{8}$$

This follows from the observation that u and v have exactly $t-2$ common neighbors, and every vertex in $\bigcap_{i=1}^{t} X_i \subseteq N[u] \cap N[v]$ must be one of them or one of u and v.

If u and v are not adjacent, then

$$t = 0 \quad \text{or} \quad \left|\bigcap_{i=1}^{t} X_i\right| \ge t+2.$$

Indeed, in this case u and v have exactly t common neighbors. Let $t > 0$. By the assumption that G is C_4-free, these t vertices form a clique. Therefore, $\bigcap_{i=1}^{t} X_i$ contains all of them as well as u and v themselves.

Thus, the graph G can be reconstructed from the hypergraph $\mathcal{H} = \mathcal{N}[G]$ by joining two vertices u and v by an edge whenever the condition (8) is true for this pair. Solving the SSP on an input $\mathcal{H}$, we first construct G by this rule and then check if $\mathcal{H} = \mathcal{N}[G]$ and if G is C_4-free. In the case of failure, no solution among C_4-free graphs exists.

Proper interval graphs. As we will discuss in more detail in Section 4.2, the SSP for proper interval graphs is solvable in logspace because these graphs form a logspace-recognizable subclass of C_4-free graphs. We now outline a different argument exemplifying our approach from [50] to the SSP for the broader classes of proper circular-arc and concave-round graphs.

Three important ingredients of our argument already appeared in Section 3.1. By Theorem 3.8, G is a proper interval graph if and only if $\mathcal{N}[G]$ is an interval hypergraph, i.e., this hypergraph admits an interval order of its vertices. By Lemma 3.10, if G is, moreover, connected, then such an interval order is unique (up to reversing and up to permutation of twins). The interval order of $\mathcal{N}[G]$ can be computed in logspace by Theorem 3.6. We now state another key element of our analysis. Given a linear order $<$ on a set V, we introduce the linear order $<^*$ on the set of all intervals in V by comparing the endpoints of intervals lexicographically w.r.t. $<$.

Lemma 4.3 (cf. [50, Lemma 5.8.1]). *Suppose that a graph G (and hence $\mathcal{N}[G]$) is twin-free. If $<$ is an interval order for $\mathcal{N}[G]$, then*

$$u < v \iff N[u] <^* N[v]. \tag{9}$$

Putting it together, we come to the following logspace algorithm for the SSP for proper interval graphs on input hypergraph $\mathcal{H}$. We first consider the case that $\mathcal{H}$ is twin-free.

Compute an interval order $<$ for $\mathcal{H}$. If this fails, no solution among proper interval graphs exists; otherwise, any solution will be surely a proper interval graph.

Next, sort the hyperedges of $\mathcal{H}$ according to the lexicographic order $<^*$. The equivalence (9) allows us to establish the v-to-$N[v]$ correspondence, that is, for each hyperedge $X \in \mathcal{H}$, to find a vertex v such that $N[v] = X$ (assuming that a solution G to the SSP on $\mathcal{H}$ exists).

Finally, we have to check that this correspondence really defines a graph, that is, whenever two vertices v and v' receive hyperedges X and X' as their neighborhoods, we have to check that $v \in X$ and that $v \in X'$ if and only if $v' \in X$. If this is not true, the SSP on input $\mathcal{H}$ has no solution.

The general case, when $\mathcal{H}$ may have twins, reduces to the twin-free case by considering the *quotient-hypergraph* $\mathcal{H}'$ w.r.t. the equivalence relation of being twins, where the vertices are the twin-classes of $\mathcal{H}$, and a set of twin-classes is a hyperedge in $\mathcal{H}'$ if and only if the union of these twin-classes is a hyperedge in $\mathcal{H}$.

4.2 Uniqueness of a solution

The argument employed in the proof of Theorem 4.2 leads us to the following observation: If we know that a graph is bipartite, then it is reconstructible from its open neighborhood hypergraph up to isomorphism. More precisely, if two graphs G and H are both bipartite, then the equality $\mathcal{N}(G) = \mathcal{N}(H)$ implies the isomorphism $G \cong H$. (See Fig. 8 below for an example of an hypergraph that is the open neighborhood hypergraph of a bipartite and of a non-bipartite graph.) Equivalently, if G and H are both co-bipartite, then

$$\mathcal{N}[G] = \mathcal{N}[H] \implies G \cong H. \tag{10}$$

In other words, any instance of the SSP for co-bipartite graphs has at most one solution up to isomorphism. The argument of Fomin et al. [30] presented above leads to the same conclusion in the case that both G and H are C_4-free. Moreover, in this case the equality $\mathcal{N}[G] = \mathcal{N}[H]$ even implies the equality $G = H$. For a smaller class of chordal graphs this was observed earlier by Harary and McKee [38]. Due to Boros et al. [15], the implication (10) is also known to be true if both G and H are bipartite.

Chen [16, 18] showed an even stronger fact for any concave-round graph G:

$$\text{for } \textit{any} \text{ graph } H, \quad \mathcal{N}[G] = \mathcal{N}[H] \implies G \cong H. \tag{11}$$

In other words, each concave-round graph is reconstructible from its closed neighborhood hypergraph up to isomorphism. Earlier such a reconstructibility result was shown for complements of forests by Aigner and Triesch [1].

Our treatment of the SSP for proper interval graphs reveals a fact that is yet stronger than (11).

Corollary 4.4. *Let G be a proper interval graph. Then, for any graph H,*

$$\mathcal{N}[G] = \mathcal{N}[H] \implies G = H.$$

In this stronger form, the reconstructibility from the closed neighborhood hypergraph was earlier known only for complete graphs; see Aigner and Triesch [1].

The implication (10) can be rephrased as the equivalence of the isomorphisms $G \cong H$ and $\mathcal{N}[G] \cong \mathcal{N}[H]$. This provides the shortest way to testing isomorphism of concave-round graphs in logspace, if we do not care of coming up with a canonical arc model. Given concave-round graphs G and H, it suffices to compute the canons of $\mathcal{N}[G]$ and $\mathcal{N}[H]$ by the algorithm of Theorem 3.13 and to check if they are equal.

Moreover, the implication (10) has important consequences for the SSP. In general, the logspace solvability of the SSP for a class of graphs $\mathcal{C}$ does still not imply the logspace solvability of the SSP for any subclass $\mathcal{C}'$ of $\mathcal{C}$. However, it does if $\mathcal{C}'$ is recognizable in logspace and (10) holds true for all G and H in $\mathcal{C}$. This observation applies to the classes of chordal, interval, and proper interval graphs, which are subclasses of C_4-free graphs. Each of these classes is recognizable in logspace by Reif [65] in combination with Reingold [66] or by methods of [49]. Therefore, the results of Fomin et al. [30] about C_4-free graphs imply that the SSP for the classes of chordal, interval, and proper interval graphs is solvable in logspace.

While the case of interval graphs is therewith efficiently solvable, note that the complexity status of the SSP for circular-arc graphs remains open.

Problem 4.5. *Is the SSP for circular-arc graphs solvable by a poly-time algorithm?*

By Corollary 4.4 and Theorem 3.8, the SSP on a given interval hypergraph has either none or exactly one solution. In general, the problem can have different solutions. For example, on a given set of 4 vertices we can draw a cycle C_4 in 3 different ways, and all three graphs will have the same closed neighborhood hypergraph. Moreover, the SSP can even have non-isomorphic solutions. This is especially easy to see after switching to the co-SSP. Fig. 8 shows an example of two non-isomorphic graphs with the same open neighborhood hypergraph. This is an instance of the following general construction

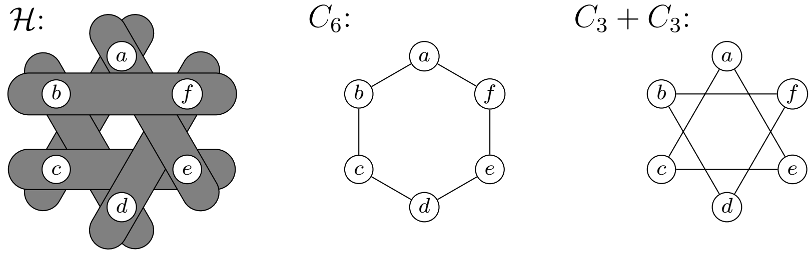


Figure 8: The open neighborhood hypergraph $\mathcal{H}$ of the two non-isomorphic graphs C_6 and $C_3 + C_3$.

in Aigner and Triesch [1]. Given an arbitrary graph G, take two copies of its vertex set, $V = \{v_1, \dots, v_n\}$ and $V' = \{v'_1, \dots, v'_n\}$, and define two graphs on the $2n$ vertices. Let $G + G$ consist of two disjoint copies of G, one on V and the other on V'. Furthermore, let $G \times G$ be a bipartite graph with vertex classes V and V', where v_i and v'_j are adjacent if and only if v_i and v_j are adjacent in G. Then $\mathcal{N}(G + G) = \mathcal{N}(G \times G)$.

Call a hypergraph $\mathcal{H}$ *uniquely realizable* if there is a unique G such that $\mathcal{H} = \mathcal{N}[G]$, that is, the SSP has a unique solution on $\mathcal{H}$. Thus, any realizable interval hypergraph is uniquely realizable.

The recognition problem of uniquely realizable hypergraphs belongs to the complexity class US (abbreviated from *Unique Solution*) introduced by Blass and Gurevich [11].

Problem 4.6. *Is the unique realizability problem* US*-complete?*

A related hardness result is obtained by Aigner and Triesch [1]: Given a connected bipartite graph G, deciding whether or not $\mathcal{N}(G) = \mathcal{N}(H)$ for some $H \not\cong G$ is NP-complete.

References

[1] M. Aigner and E. Triesch. Reconstructing a graph from its neighborhood lists. *Combinatorics, Probability & Computing*, 2:103–113, 1993.

[2] V. Arvind, B. Das, J. Köbler, and S. Kuhnert. The isomorphism problem for k-trees is complete for logspace. *Information and Computation*, 217:1–11, 2012.

[3] V. Arvind, B. Das, J. Köbler, and S. Toda. Colored hypergraph isomorphism is fixed parameter tractable. In *Proc. 30th FSTTCS*, volume 8 of *LIPIcs*, pages 327–337, Dagstuhl, 2010. Leibniz-Zentrum für Informatik.

[4] V. Arvind and J. Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.

[5] L. Babai and P. Codenotti. Isomorhism of hypergraphs of low rank in moderately exponential time. In *Proc. of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 667–676. IEEE Computer Society, 2008.

[6] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proceedings of the 15-th Annual ACM Symposium on Theory of Computing*, pages 171–183, 1983.

[7] L. Babel and S. Olariu. On the isomorphism of graphs with few P_4s. In M. Nagl, editor, *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 1017 of *LNCS*, pages 24–36. Springer, 1995.

[8] L. Babel, I. N. Ponomarenko, and G. Tinhofer. The isomorphism problem for directed path graphs and for rooted directed path graphs. *J. Algorithms*, 21(3):542–564, 1996.

[9] J. Bang-Jensen, J. Huang, and A. Yeo. Convex-round and concave-round graphs. *SIAM J. Discrete Math.*, 13(2):179–193, 2000.

[10] S. Benzer. On the topology of the genetic fine structure. *Proceedings of the National Academy of Sciences of the United States of America*, 45(11):1607–1620, 1995.

[11] A. Blass and Y. Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1–3):80–88, 1982.

[12] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990.

[13] K. Booth and C. Colbourn. Problems polynomially equivalent to Graph Isomorphism. Technical Report CS-77-04, Comp. Sci. Dep., Univ. Waterloo, 1979.

[14] K. Booth and G. 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.

[15] E. Boros, V. Gurvich, and I. E. Zverovich. Neighborhood hypergraphs of bipartite graphs. *Journal of Graph Theory*, 58(1):69–95, 2008.

[16] L. Chen. Graph isomorphism and identification matrices: Parallel algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 7(3):308–319, 1996.

[17] L. Chen. Graph isomorphism and identification matrices: Sequential algorithms. *J. Comput. Syst. Sci.*, 59(3):450–475, 1999.

[18] L. Chen. A selected tour of the theory of identification matrices. *Theor. Comput. Sci.*, 240(2):299–318, 2000.

[19] L. Chen and Y. Yesha. Parallel recognition of the consecutive ones property with applications. *J. Algorithms*, 12(3):375–392, 1991.

[20] D. G. Corneil, H. Kim, S. Natarajan, S. Olariu, and A. P. Sprague. Simple linear time recognition of unit interval graphs. *Inform. Proc. Lett.*, 55:99–104, 1995.

[21] A. Curtis, M. Lin, R. McConnell, Y. Nussbaum, F. Soulignac, J. Spinrad, and J. Szwarcfiter. Isomorphism of graph classes related to the circular-ones property. *E-print*, *http://arxiv.org/abs/1203.4822v1*, 2012.

[22] B. Das, J. Torán, and F. Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. In J.-Y. Marion and T. Schwentick, editors, *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science*, volume 5 of *LIPIcs*, pages 227–238, Dagstuhl, 2010. Leibniz-Zentrum für Informatik.

[23] S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. 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.

[24] S. Datta, P. Nimbhorkar, T. Thierauf, and F. Wagner. Graph Isomorphism for $K_{3,3}$-free and K_5-free graphs is in Log-Space. In R. Kannan and K. N. Kumar, editors, *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4 of *LIPIcs*, pages 145–156, Dagstuhl, 2009. Leibniz-Zentrum für Informatik.

[25] X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25(2):390–403, 1996.

[26] M. Dom. Algorithimic aspects of the consecutive-ones property. *Bulletin of the EATCS*, 98:27–59, 2009.

[27] P. Duchet. Classical perfect graphs. An introduction with emphasis on triangulated and interval graphs. Perfect graphs, Ann. Discrete Math. 21, 67–96 (1984)., 1984.

[28] S. Evdokimov and I. N. Ponomarenko. Isomorphism of coloured graphs with slowly increasing multiplicity of jordan blocks. *Combinatorica*, 19(3):321–333, 1999.

[29] I. Filotti and J. N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 236–243, 1980.

[30] F. V. Fomin, J. Kratochvíl, D. Lokshtanov, F. Mancini, and J. A. Telle. On the complexity of reconstructing H-free graphs from their Star Systems. *Journal of Graph Theory*, 68(2):113–124, 2011.

[31] G. Gati. Further annotated bibliography on the isomorphism disease. *J. Graph Theory*, 3:95–109, 1979.

[32] F. Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357–369, 1974.

[33] M. C. Golumbic. *Algorithmic graph theory and perfect graphs. 2nd ed.* Amsterdam: Elsevier, 2004.

[34] M. Grohe. Isomorphism testing for embeddable graphs through definability. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 63–72, 2000.

[35] M. Grohe. From polynomial time queries to graph structure theory. *Commun. ACM*, 54(6):104–112, 2011.

[36] M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, 2012. To appear. Preprint at *http://arxiv.org/abs/1111.1109.*

[37] M. Grohe and O. Verbitsky. Testing graph isomorphism in parallel by playing a game. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, Part I*, volume 4051 of *LNCS*, pages 3–14. Springer, 2006.

[38] F. Harary and T. A. McKee. The square of a chordal graph. *Discrete Mathematics*, 128(1–3):165–172, 1994.

[39] J. Hopcroft and R. Tarjan. A V^2 algorithm for determining isomorphism of planar graphs. *Inf. Process. Lett.*, 1:32–34, 1971.

[40] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing*, pages 172–184, 1974.

[41] W.-L. Hsu. $O(m \cdot n)$ isomorphism algorithms for circular-arc graphs and circle graphs. In R. Kannan and W. R. Pulleyblank, editors, *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, pages 297–311. University of Waterloo Press, 1990.

[42] W.-L. Hsu and R. M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 3 2003.

[43] B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.

[44] B. L. Joeris, M. C. Lin, R. M. McConnell, J. P. Spinrad, and J. L. Szwarcfiter. Linear time recognition of Helly circular-arc models and graphs. *Algorithmica*, 59(2):215–239, 2 2011.

[45] K.-I. Kawarabayashi and B. Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *Proc. of the 40th Ann. ACM Symp. on Theory of Computing*, pages 471–480, 2008.

[46] M. Klawe, D. Corneil, and A. Proskurowski. Isomorphism testing in hookup classes. *SIAM J. Algebraic Discrete Methods*, 3:260–274, 1982.

[47] P. N. Klein. Efficient parallel algorithms for chordal graphs. *SIAM J. Comput.*, 25(4):797–827, 1996.

[48] J. Köbler. On graph isomorphism for restricted graph classes. In A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker, editors, *Logical Approaches to Computational Barriers, Proceedings of the 2nd Conference on Computability in Europe*, volume 3988 of *LNCS*, pages 241–256. Springer, 2006.

[49] J. Köbler, S. Kuhnert, B. Laubner, and O. Verbitsky. Interval graphs: Canonical representations in Logspace. *SIAM J. on Computing*, 40(5):1292–1315, 2011.

[50] J. Köbler, S. Kuhnert, and O. Verbitsky. Solving the canonical representation and star system problem for proper circular-arc graphs in logspace. *E-print, http://arxiv.org/abs/1202.4406*, 2012.

[51] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Progress in Theoretical Computer Science. Birkhäuser, 1993.

[52] S. Kratsch and P. Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In H. Kaplan, editor, *Proc. of the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *LNCS*, pages 81–92. Springer, 2010.

[53] F. Lalonde. Le probleme d'etoiles pour graphes est NP-complet. *Discrete Mathematics*, 33(3):271–280, 1981.

[54] B. Laubner. Capturing polynomial time on interval graphs. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science*, 2010.

[55] M. C. Lin and J. L. Szwarcfiter. Characterizations and recognition of circular-arc graphs and subclasses: A survey. *Discrete Mathematics*, 309(18):5618–5635, 2009.

[56] S. Lindell. A logspace algorithm for tree canonization. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 400–404, 1992.

[57] G. Lueker and K. Booth. A linear time algorithm for deciding interval graph isomorphism. *J. ACM*, 26(2):183–195, 1979.

[58] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

[59] E. M. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In J. S. Vitter, L. L. Larmore, and F. T. Leighton, editors, *Proc. of the 31st Ann. ACM Symposium on Theory of Computing*, pages 652–658. ACM, 1999.

[60] G. L. Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 225–235, 1980.

[61] G. L. Miller and J. H. Reif. Parallel tree contraction. Part 2: Further applications. *SIAM J. Comput.*, 20(6):1128–1147, 1991.

[62] J. Moon and L. Moser. On cliques in graphs. *Isr. J. Math.*, 3:23–28, 1965.

[63] I. Ponomarenko. The isomorphism problem for classes of graphs that are invariant with respect to contraction. In *Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov., 174 (Teor. Slozhn. Vychisl. 3)*, pages 147–177. LOMI, 1988. Translation from Russian in *J. Soviet Math.* 55(2):1621–1643 (1991).

[64] R. C. Read and D. G. Corneil. The graph isomorphism disease. *J. Graph Theory*, 1:339–363, 1977.

[65] J. Reif. Symmetric complementation. *J. ACM*, 31(2):401–421, 1984.

[66] O. Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[67] F. Roberts. Indifference graphs. Proof Tech. Graph Theory, Proc. 2nd Ann Arbor Graph Theory Conf. 1968, 139-146 (1969)., 1969.

[68] P. Schweitzer. Isomorphism of (mis)labeled graphs. In *Proc. of the 19th Ann. European Symposium on Algorithms*, volume 6942 of *LNCS*, pages 370–381. Springer, 2011.

[69] J. Spinrad. *Efficient graph representations.* Number 19 in Field Institute Monographs. AMS, 2003.

[70] S. Toda. Computing automorphism groups of chordal graphs whose simplicial components are of small size. *IEICE Transactions*, 89-D(8):2388–2401, 2006.

[71] J. Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33(5):1093–1108, 2004.

[72] J. Torán and F. Wagner. The complexity of planar graph isomorphism. *Bulletin of the EATCS*, 97:60–82, 2009.

[73] A. Tucker. Matrix characterizations of circular-arc graphs. *Pac. J. Math.*, 39:535–545, 1971.

[74] R. Uehara. Simple geometrical intersection graphs. In S.-I. Nakano and M. S. Rahman, editors, *Proceedings of the 2nd International Workshop on Algorithms and Computation*, volume 4921 of *LNCS*, pages 25–33. Springer, 2008.

[75] R. Uehara, S. Toda, and T. Nagoya. Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs. *Discrete Applied Mathematics*, 145(3):479–482, 2005.

[76] K. Yamazaki, H. L. Bodlaender, B. de Fluiter, and D. M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999.

[77] V. Zemlyachenko, N. Kornienko, and R. Tyshkevich. Graph isomorphism problem. *J. Sov. Math.*, 29:1426–1481, 1985.