Graph Isomorphism Restricted by Lists*

Pavel Klavík[1], Dušan Knop[2], and Peter Zeman[2]

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

[2] Department of Applied Mathematics,
Faculty of Mathematics and Physics,
Charles University in Prague, Czech Republic.
{knop,zeman}@kam.mff.cuni.cz.

Abstract. The complexity of *graph isomorphism* (GraphIso) is a famous unresolved problem in theoretical computer science. For graphs G and H, it asks whether they are the same up to a relabeling of vertices. In 1981, Lubiw proved that *list restricted graph isomorphism* (ListIso) is NP-complete: for each $u \in V(G)$, we are given a list $\mathfrak{L}(u) \subseteq V(H)$ of possible images of u. After 35 years, we revive the study of this problem and consider which results for GraphIso can be used to solve ListIso.

We prove the following: 1) When GraphIso is GI-complete for a class of graphs, it usually implies NP-completeness of ListIso. 2) Combinatorial algorithms for GraphIso can be modified into algorithms for ListIso: for trees, planar graphs, interval graphs, circle graphs, permutation graphs, bounded genus graphs, and bounded treewidth graphs. 3) Two basic algorithms based on group theory cannot be modified: ListIso remains NP-complete for cubic colored graphs with sizes of color classes bounded by 8.

Also, ListIso allows to classify results for the graph isomorphism problem. Some algorithms, which we call *robust*, can be modified to ListIso. A fundamental problem is to construct a combinatorial polynomial-time algorithm for cubic graph isomorphism, avoiding group theory. By the 3rd result, ListIso is NP-hard for them, so no robust polynomial-time algorithm for cubic graph isomorphism exists, unless $\mathsf{P} = \mathsf{NP}$.

Keywords: graph isomorphism, restricted computational problem, polynomial-time algorithms, NP-completeness, bounded genus graphs, bounded treewidth, bounded degree graphs.

Diagram: For a dynamic structural diagram of our results, see the following website (supported Firefox and Google Chrome): `http://pavel.klavik.cz/orgpad/list_isomorphism.html`

*The authors are supported by CE-ITI (P202/12/G061 of GAČR), the first and the third author by Charles University as GAUK 1334217.

1 Introduction

For graphs G and H, a bijection $\pi : G \to H$ is called an *isomorphism* if $uv \in E(G) \iff \pi(u)\pi(v) \in E(H)$. The *graph isomorphism problem* (GraphIso) asks whether there exists an isomorphism from G to H. It obviously belongs to NP, and no polynomial-time algorithm is known. It is a prime candidate for an intermediate problem with complexity between P and NP-complete. There are threefold evidences that GraphIso is unlikely to be NP-complete: equivalence of existence and counting [3,73], GraphIso belongs to coAM, so the polynomial-hierarchy collapses if GraphIso is NP-complete [40,83], and GraphIso can be solved in quasipolynomial time [5]. For a survey, see [4].

1.1 Graph Isomorphism Problem for Restricted Graph Classes and Parameters

The graph isomorphism problem is solved efficiently for various restricted graph classes and parameters, see Fig. 1.

Combinatorial Algorithms. A prime example is the linear-time algorithm for testing graph isomorphism of (rooted) trees. It is a bottom-up procedure comparing subtrees. This algorithm is very robust and captures all possible isomorphisms. For many other graph classes, graph isomorphism reduces to graph isomorphism of labeled trees: for planar graphs [48,47,49], interval graphs [70], circle graphs [52], and permutation graphs [15,85]. Involved combinatorial arguments are used to solve graph isomorphism for bounded genus graphs [66,34,75,55] and bounded treewidth graphs [10,68].

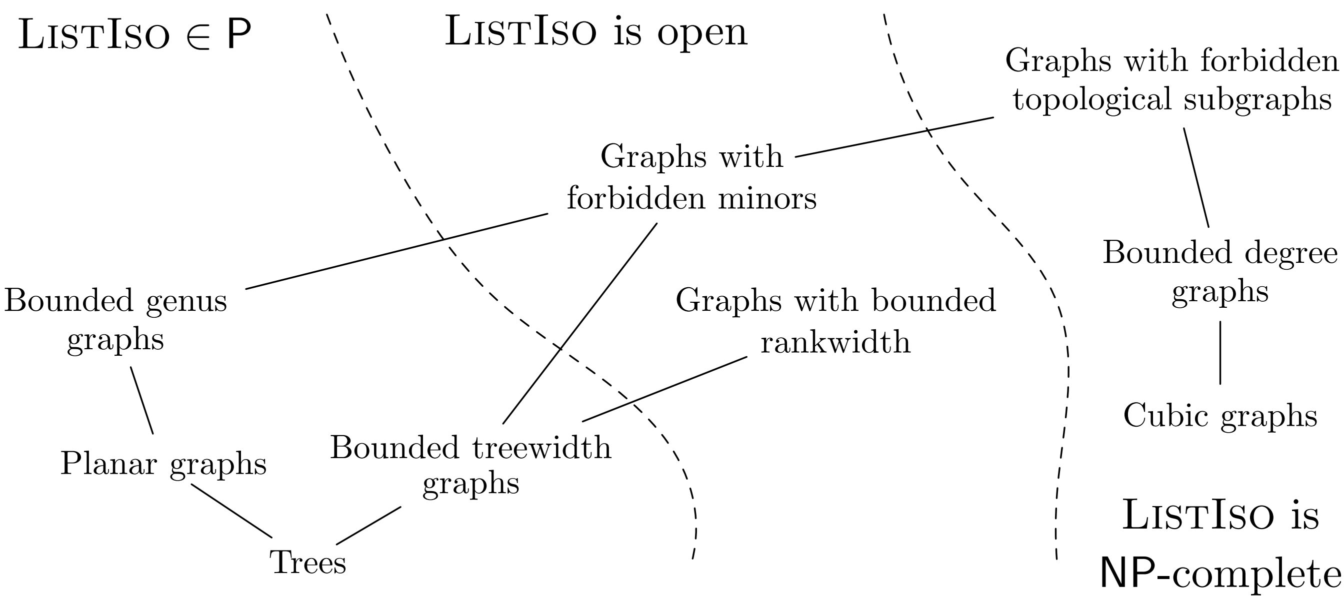


Fig. 1. Important graph classes for which the graph isomorphism problem can be solved in polynomial time. Our complexity results for the list restricted graph isomorphism problem are depicted.

Algorithms Based on Group Theory. The graph isomorphism problem is closely related to group theory, in particular to computing generators of automorphism groups of graphs. Assuming that G and H are connected, we can test $G \cong H$ by computing generators of $\mathrm{Aut}(G \dot{\cup} H)$ and checking whether there exists a generator which swaps G and H. For the converse relation, Mathon [73] proved that generators of the automorphism group can be computed using $\mathcal{O}(n^3)$ instances of graph isomorphism.

Therefore, GraphIso can be attacked by techniques of group theory. A prime example is the seminal result of Luks [71] which uses group theory to solve GraphIso for graphs of bounded degree in polynomial time. If G has bounded degree, its automorphism group $\mathrm{Aut}(G)$ may be arbitrary, but the stabilizer $\mathrm{Aut}_e(G)$ of an edge e is restricted. Luks' algorithm tests GraphIso by an iterative process which determines $\mathrm{Aut}_e(G)$ in steps, by adding layers around e.

Group theory can be used to solve GraphIso of colored graphs with bounded sizes of color classes [37] and of graphs with bounded eigenvalue multiplicity [6,29]. Miller [76] solved GraphIso of k-contractible graphs (which generalize both bounded degree and bounded genus graphs), and his results are used by Ponomarenko [79] to show that GraphIso can be decided in polynomial time for graphs with excluded minors. Luks' algorithm [71] for bounded degree graphs is also used by Grohe and Marx [42] as a subroutine to solve GraphIso on graphs with excluded topological subgraphs. The recent breakthrough of Babai [5] heavily uses group theory to solve the graph isomorphism problem in quasipolynomial time.

Is Group Theory Needed? One of the fundamental problems for understanding the graph isomorphism problem is to understand in which cases group theory is really needed, and in which cases it can be avoided.[3] For instance, for which graph classes can GraphIso be decided by the classical combinatorial algorithm called k-dimensional Weisfieler-Leman refinement (k-WL)? (Described in Conclusions.)

Ponomarenko [79] used group theory to solve GraphIso in polynomial time on graphs with excluded minors. Robertson and Seymour [81] proved that a graph G with an excluded minor can be decomposed into pieces which are "almost embeddable" to a surface of genus g, where g depends on this minor. Recently, Grohe [41] generalized this to show that for G, there exists a treelike decomposition into almost embeddable pieces which is *automorphism-invariant* (every automorphism of G induces an automorphism of the treelike decomposition). Using this decomposition, it is possible to solve graph isomor-

[3] Ilya Ponomarenko in personal communication.

phism in polynomial time and to avoid group theory techniques. In particular, k-WL can decide graph isomorphism on graphs with excluded minors where k depends on the minor.

It is a long-standing open problem whether the graph isomorphism problem for bounded degree graphs, and in particular for cubic graphs, can be solved in polynomial time without group theory. It is known that k-WL, for any fixed k, cannot decide graph isomorphism on cubic graphs [13]. Very recently, fixed parameter tractable algorithms for graphs of bounded treewidth [68] and for graphs of bounded genus [55] were constructed. On the other hand, the best known parameterized algorithm for graphs of bounded degree is the XP algorithm of Luks [71], and it is a major open problem whether an FPT algorithm exists.

In this paper, we propose a different approach to show limitations of techniques used to attack the graph isomorphism problem. We study its generalization called *list restricted graph isomorphism* (ListIso) which is NP-complete for general graphs.

> **Implications for GraphIso.** The study for ListIso allows to classify the results for the graph isomorphism problem. An algorithm for GraphIso is called *robust* if it can be modified to solve ListIso while preserving the complexity. (Say, it remains a polynomial-time algorithm, fixed parameter tractable algorithm, etc.)

We understand that the notions of modified algorithms and of robustness are vague. For instance, if an algorithm B is created from A by completely replacing A with B, is B still a modification of A? At this moment, a precise definition of robustness is unclear, but the reader may understand it intuitively, similarly as a statement: "The proof of the result X is created by a modification of a proof of a result Y." Robustness is not used in any formal statement mentioned in this paper. The purpose of this paper is to give more insight into this notion in the context of the graph isomorphism problem.

We show that many combinatorial algorithms for graph isomorphism are robust. On the other hand, hardness results for ListIso imply non-existence of robust algorithms for GraphIso. In particular, we show that ListIso is NP-complete for cubic graphs, so no robust algorithm for cubic graph isomorphism exists, unless P = NP. Similarly, no robust FPT algorithm for graph isomorphism of graphs of bounded degree exists.

1.2 List Restricted Graph Isomorphism

In 1981, Lubiw [69] introduced the following computational problems. Let G and H be graphs, and the vertices of G be equipped with lists: each vertex

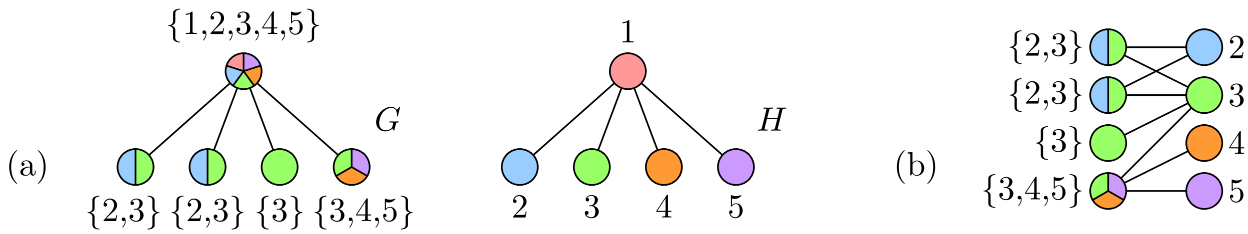


Fig. 2. (a) Two isomorphic graphs G and H with no list-compatible isomorphism. (b) It does not exist because there is no perfect matching between the lists of the leaves of G and the leaves of H.

$u \in V(G)$ has a list $\mathfrak{L}(u) \subseteq V(H)$. We say that an isomorphism $\pi : G \to H$ is *list-compatible* if, for all vertices $u \in V(G)$, we have $\pi(u) \in \mathfrak{L}(u)$; see Fig. 2a. A list-compatible isomorphism $\pi : G \to G$ is called a *list-compatible automorphism*.

Problem: List restricted graph isomorphism – ListIso
Input: Graphs G and H, and the vertices of G are equipped by lists $\mathfrak{L}(u) \subseteq V(H)$.
Output: Is there a list-compatible isomorphism $\pi : G \to H$?

Problem: List restricted graph automorphism – ListAut
Input: A graph G with vertices equipped with lists $\mathfrak{L}(u) \subseteq V(G)$.
Output: Is there a list-compatible automorphism $\pi : G \to G$?

These two problems are polynomially equivalent (see Lemma 3.1). Lubiw [69] proved the following surprising result:

Theorem 1.1 (Lubiw [69]). *The problems* ListIso *and* ListAut *are* NP-*complete.*

Moreover, she proved that finding a fixed-point free involutory automorphism of a graph is NP-complete. Lalonde [65] showed that it is NP-complete to decide whether a bipartite graph has an involutory automorphism exchanging the parts; see [36].

Independently, ListIso was rediscovered in [31,33]. Given two graphs G and H, we say that G *regularly covers* H if there exists a semiregular subgroup $\Gamma \le \mathrm{Aut}(G)$ such that $G/\Gamma \cong H$. The list restricted graph isomorphism problem was used as a subroutine in [31,33] for 3-connected planar and projective graphs to test regular covering when G is a planar graph. The key idea is that a planar graph G can be reduced to a 3-connected planar graph G_r, for which $\mathrm{Aut}(G_r)$ is a spherical group. Therefore, we can compute all regular quotients G_r/Γ_r. Next, we reduce H towards G_r/Γ_r. The problem is that subgraphs of H may correspond to several different parts in G, so we compute lists of all possibilities. One subroutine of the reduction leads to ListIso of 3-connected planar and projective planar graphs, while the other leads to a generalization of bipartite perfect matching [35].

We note that other computational problems restricted by lists are frequently studied. *List coloring*, introduced by Vizing [91], is NP-complete even for planar graphs [63] and interval graphs [9]. *List H-homomorphisms*, having a similar setting as ListIso, were also considered; see [46,21,14].

It was suggested by an anonymous reviewer that unlike for GraphIso, for which the role of graphs G and H is symmetric, the role of G and H in ListIso is asymmetric, and thus the ListIso problem is closer to the H-homomorphism problem. We argue that this is not the case, ListIso is symmetric as well. Given lists $\mathfrak{L}(u) \subseteq V(H)$ for each $u \in V(G)$, we derive the corresponding lists $\mathfrak{L}^{-1}(w) \subseteq V(G)$ for each $w \in V(H)$: $\mathfrak{L}^{-1}(w) = \big\{u : u \in V(G), w \in \mathfrak{L}(u)\big\}$. An isomorphism $\pi : G \to H$ is list-compatible with $\mathfrak{L}$, if and only if the isomorphism $\pi^{-1} : H \to G$ is list-compatible with $\mathfrak{L}^{-1}$. Actually, we could work with both lists simultaneously. The similarity of ListIso and GraphIso also follows from the fact that many combinatorial algorithms for GraphIso can be modified for ListIso without any difficulty.

1.3 Our Results

We revive the study of list restricted graph isomorphism. The goal is to determine which techniques for GraphIso can be modified to ListIso. We believe that ListIso is a very natural computational problem, as evidenced by its application in [31,33]. Further, its hardness results prove non-existence of robust algorithms for the graph isomorphism problem itself. For instance, it is believed that no NP-complete problem can be solved in quasipolynomial time. Therefore, Babai's algorithm [5] cannot be robust, i.e., it cannot be modified to solve ListIso in quasipolynomial time. To solve GraphIso efficiently (say, in polynomial time), one necessarily has to apply some non-robust techniques which does not generalize to ListIso.

The described algorithms for ListIso are a straightforward modification of previously known algorithms for the graph isomorphism problem. The main point of this paper is *not to develop new algorithmic techniques*, but to *classify known techniques for* GraphIso *from a different viewpoint.* This viewpoint is the robustness of algorithms with respect to ListIso, and our results give an insight into its meaning.

We prove the following three *informal results* in this paper; see Fig. 1 for an overview:

Result 1. GI*-completeness results for* GraphIso *with polynomial-time reductions using vertex-gadgets imply* NP*-completeness for* ListIso*.*

For many classes $\mathcal{C}$ of graphs, it is known that GraphIso is equally hard for them as for general graphs, i.e., it is GI-complete. For instance, GraphIso is GI-complete for bipartite graphs, split and chordal graphs [70], chordal bipartite and strongly chordal graphs [90], trapezoid graphs [86], comparability graphs of dimension 4 [57], grid intersection graphs [89], line graphs [95], and self-complementary graphs [16].

The polynomial-time reductions are often done in a way that all graphs are encoded into $\mathcal{C}$, by replacing each vertex with a small vertex-gadget. (The constructions are quite simple, and the non-trivial part is to prove that the constructed graph belongs to $\mathcal{C}$.) As we prove in Theorem 4.1, such reductions using vertex-gadgets also can be modified to solve ListIso: they imply NP-completeness of ListIso for $\mathcal{C}$. For instance, ListIso is NP-complete for all graph classes mentioned above (Corollary 4.3).

Result 2. *The problem* ListIso *can be solved in polynomial-time for trees, planar graphs, interval graphs, circle graphs, permutation graphs, bounded genus graphs and bounded treewidth graphs.*

These algorithms are modifications of known combinatorial techniques for GraphIso. As a by-product, our paper gives an overview of these techiques which are often robust and can be modified to solve ListIso in a straightforward way. Moreover, we can describe them more naturally with lists.

For example, the bottom-up linear-time algorithm for testing graph isomorphism of (rooted) trees can be modified to solve ListIso in Theorem 6.1, since it captures all possible isomorphisms. The key difference is that the algorithm for ListIso finds perfect matchings in bipartite graphs, in order to decide whether lists of several subtrees are simultaneously compatible; see Fig. 2b. We use the algorithm of Hopcroft and Karp [50], running in time $\mathcal{O}(\sqrt{n}m)$.

The algorithms for graph isomorphism of planar, interval, permutation and circle graphs based on tree decompositions and can be modified to solve ListIso, as we show in Theorems 8.3, 7.1, 7.2, and 7.3. Even more involved algorithms for graphs isomorphism of bounded genus and bounded treewidth graphs can be modified to solve ListIso in Theorems 9.1 and 10.5. The complexity for graphs with bounded rankwidth and graphs with excluded minors remains open, see Conclusions for details.

Result 3. *The problem* ListIso *is* NP*-complete for 3-regular colored graphs with all color classes of size at most 8 and with all lists of size at most 3.*

This result contrasts with two fundamental results using group theory techniques to solve graph isomorphism in polynomial time for graphs of bounded degrees [71] and bounded color classes [37]. Therefore, no robust algorithm solving graph isomorphism for these graph classes exists. In general, our impression is that group theory techniques cannot be modified to solve ListIso since list-compatible automorphisms of a graph G do not form a subgroup of $\mathrm{Aut}(G)$. In Theorem 5.3, we prove Result 3 by describing a non-trivial modification of the original NP-hardness reduction of Lubiw [69].

2 Preliminaries and Outline

Let G be an input graph of ListIso or ListAut. We denote the set of its vertices by $V(G)$ and the set of its edges by $E(G)$. Let $n = |V(G)|$, $m = |E(G)|$ and ℓ be the total size of all lists. To make the problem non-trivial, we can assume that $\ell \geq n$.

Bipartite Perfect Matchings. As a subroutine, we frequently solve *bipartite perfect matching*:

Lemma 2.1 (Hopcroft and Karp [50]). *The bipartite perfect matching problem can be solved in time $\mathcal{O}(\sqrt{n}m)$, where n is the number of vertices and m is the number of edges.*

We repeatedly use this subroutine to solve ListIso for many graph classes. Therefore, the running time of many of our algorithms $\mathcal{O}(\sqrt{n}\ell)$ while the input size is $\Omega(n + \ell)$. This cannot be avoided for the following reason.

Lemma 2.2. *There exists a linear-time reduction from the bipartite perfect matching problem for n vertices and m edges to* ListIso *of two independent sets with n vertices and $\ell = m$.*

Proof. We have a bipartite graph B. One part X is represented by $V(G)$ and the other part Y by $V(H)$. For every $u \in X$, we put $\mathfrak{L}(u) = \{v : v \in Y, uv \in E(B)\}$. $\square$

Similar reductions work for trees, etc. Therefore, finding bipartite perfect matchings is the bottleneck in many of our algorithms and cannot be avoided: if it cannot be solved in linear time, ListIso for many graph classes cannot be solved in linear time as well.

Outline: Main Points of This Paper. In Section 3, we prove some basic results for ListIso such as polynomial-time equivalence of ListIso and ListAut and polynomial-time algorithms when maximum degree is 2 or all lists are of size at most 2.

In Section 4, we give a formal description of Result 1. We study polynomial-time reductions ψ for GraphIso from a graph class $\mathcal{C}$ to another graph class $\mathcal{C}'$: for each graph $G \in \mathcal{C}$, the reduction ψ produces another graph $G' \in \mathcal{C}'$ such that $G \cong H$ if and only if $G' \cong H'$. When ψ uses vertex-gadgets, it can be modified to a polynomial-time reduction for ListIso from $\mathcal{C}$ to $\mathcal{C}'$. The vertex-gadget assumption means the following: in G', each vertex $V(G)$ is replaced by a small vertex-gadget while all automorphisms of G' preserve vertex-gadgets and automorphisms of G induce automorphisms of G' and vice versa.

In Section 5, we prove Result 3. We modify the reduction of Lubiw [69] in two steps. First, we reduce the problem from (positive) 1-in-3 SAT instead of 3-SAT. Therefore, only positive literals appear and we reduce the sizes of lists from 7 to 3. Second, we modify variable gadgets to make the graph 3-regular.

In the remaining sections, we give a formal description of Result 2. In Section 6, we modify the basic algorithm for graph isomorphism of (rooted) trees. To deal with lists, we solve several bipartite perfect matching subroutines to test whether subtrees are simultaneously list-compatible. The idea of this algorithm for ListIso of trees is used in some other combinatorial algorithms.

In Section 8, we describe that every planar graph can be decomposed into a tree of its 3-connected components. Since ListIso can be easily solved on 3-connected planar graphs (using geometry and uniqueness of embedding), we apply dynamic programming on this tree and solve ListIso for general planar graphs as well.

In Section 7, we describe how to modify the algorithms for graph isomorphism of interval, permutation and circle graphs. Similarly, they can be represented by MPQ-trees, modular trees and split trees. Since we can solve ListIso on the graphs induced by nodes, we apply dynamic programming on these trees and solve ListIso on interval, permutation and circle graphs as well.

In Section 9, we modify the algorithm of Kawarabayashi [55] to solve ListIso on graphs of bounded genus. This algorithm either uses a small number of possible embeddings (which can be modified to solve ListIso), or finds a small cut of size at most 4 which is canonical, splits the graph and test graph isomorphism of both pieces (which again can be modified to solve ListIso).

In Section 10, we modify Bodlaender's XP algorithm [10] for graph isomorphism of graphs of bounded treewidth. The problem is non-trivial since tree decomposition is not canonical. Therefore, it is a dynamic algorithm running over all potential bags, and it can be easily modified with lists. Lokshtanov et al. [68] obtain an FPT running time by computing a smaller set of potential bags which is canonical.

In Section 11, we conclude this paper with a group reformulation, related results and open problems.

3 Basic Results

In this section, we prove some basic results concerning the complexity of ListIso and ListAut.

Lemma 3.1. *Both problems* ListAut *and* ListIso *are polynomially equivalent.*

Proof. To see that ListAut is polynomially reducible to ListIso just set H to be a copy of G and keep the lists for all vertices of G. It is straightforward to check that these two instances are equivalent. For the other direction, we build an instance G' and $\mathfrak{L}'$ of ListAut as follows. Let G' be a disjoint union of G and H. And let $\mathfrak{L}'(v) = \mathfrak{L}(v)$ for all $v \in V(G)$ and set $\mathfrak{L}'(w) = V(G)$ for all $w \in V(H)$. It is easy to see that there exists list-compatible isomorphism from G to H, if and only if there exists a list-compatible automorphism of G'. $\square$

Lemma 3.2. *The problem* ListIso *can be solved in time $\mathcal{O}(n+m)$ when all lists are of size at most two.*

Proof. We construct a list-compatible isomorphism $\pi : G \to H$ by solving a 2-Sat formula which can be done in linear time [30,2]. When $w \in \mathfrak{L}(v)$, we assume that $\deg(v) = \deg(w)$, otherwise we remove w from $\mathfrak{L}(v)$. Notice that if $\mathfrak{L}(u) = \{w\}$, we can set $\pi(u) = w$ and for every $v \in N(u)$, we modify $\mathfrak{L}(v) := L(v) \cap N(w)$. Now, for every vertex u_i with $\mathfrak{L}(u_i) = \{w_i^0, w_i^1\}$, we introduce a variable x_i such that $\pi(u_i) = w_i^{x_i}$. Clearly, the mapping π is compatible with the lists.

We construct a 2-SAT formula such that there exists a list-compatible isomorphism if and only if it is satisfiable. First, if $\mathfrak{L}(u_i) \cap \mathfrak{L}(u_j) \neq \emptyset$, we add implications for x_i and x_j such that $\pi(u_i) \neq \pi(u_j)$. Next, when $\pi(u_i) = w_i^j$, we add implications that every $u_j \in N(u_i)$ is mapped to $N(w_i^j)$. If $\mathfrak{L}(u_j) \cap N(w_i^j) \neq \emptyset$, otherwise u_i cannot be mapped to w_i^j and $x_i \neq j$. Therefore, π obtained from a satisfiable assignment maps $N[u]$ bijectively to $N[\pi(u)]$ and it is an isomorphism. The total number of variables in n, and the total number of clauses is $\mathcal{O}(n+m)$, so the running time is $\mathcal{O}(n+m)$. $\square$

Lemma 3.3. *Let $G_1, \dots, G_k$ be the components of G and $H_1, \dots, H_k$ be the components of H. If we can decide* ListIso *in polynomial time for all pairs G_i and H_j, then we can solve* ListIso *for G and H in polynomial time.*

Proof. Let $G_1, \dots, G_k$ be the components of G and $H_1, \dots, H_k$ be the components of H. For each component G_i, we find all components H_j such that there exists a list-compatible isomorphism from G_i to H_j. Notice that a necessary condition is that every vertex in G_i contains one vertex of H_j in its list. So we can go through all lists of G_i and find all candidates H_j, in total time $\mathcal{O}(\ell)$ for all components $G_1, \dots, G_k$. Let $n' = |V(G_i)|$, $m' = |E(G_i)|$, and ℓ' be the total size of lists of G_i restricted to H_j. We test existence of a list-compatible isomorphism in time $\varphi(n', m', \ell')$. Then we form the bipartite graph B between $G_1, \dots, G_k$ and $H_1, \dots, H_k$ such that $G_i H_j \in E(B)$ if and only if there exists a list-compatible isomorphism from G_i to H_j. There exists a list-compatible isomorphism from G to H, if and only if there exists a perfect matching in B. Using Lemma 2.1, this can be tested in time $\mathcal{O}(\sqrt{k}\ell)$. The total running time depends on the running time of testing ListIso of the components, and we note that the sum of the lengths of lists in these test is at most ℓ. $\square$

Lemma 3.4. *The problem* ListIso *can be solved for cycles in time $\mathcal{O}(\ell)$.*

Proof. We may assume that $|V(G)| = |V(H)|$. Let $u \in V(G)$ be a vertex with a smallest list and let $k = |\mathfrak{L}(u)|$. Since $\ell = \mathcal{O}(kn)$, it suffices to show that we can find a list-compatible isomorphism in time $\mathcal{O}(kn)$. We test all the k possible mappings $\pi \colon G \to H$ with $\pi(u) \in \mathfrak{L}(u)$. For $u \in V(G)$ and $v \in \mathfrak{L}(u)$, there are at most two possible isomorphisms that map u to v. For each of these isomorphism, we test whether they are list-compatible. $\square$

Lemma 3.5. *The problem* ListIso *can be solved for graphs of maximum degree 2 in time $\mathcal{O}(\sqrt{n}\ell)$.*

Proof. Both graphs G and H are disjoint unions of paths and cycles of various lengths. For each two connected components, we can decide in time $\mathcal{O}(\ell')$ whether there exists a list-compatible isomorphism between them, where ℓ' is the total size of lists when restricted to these components: for paths trivially, and for cycles by Lemma 3.4. The rest follows from Lemma 3.3, where the running time is of each test in $\mathcal{O}(\ell')$ where ℓ' is the total length of lists restricted to two components. $\square$

4 GI-completeness Implies NP-completeness

When that graph isomorphism is GI-complete for some class of graphs $\mathcal{C}'$. We want to show that in most cases, the reductions can be modified to show NP-completeness of ListIso for $\mathcal{C}'$.

Vertex-gadget Reductions. Suppose that GraphIso is GI-complete for a class $\mathcal{C}$. To show that GraphIso is GI-complete for another class $\mathcal{C}'$, one builds a polynomial-time reduction ψ from GraphIso of $\mathcal{C}$: given graphs $G, H \in \mathcal{C}$, we construct graphs $G', H' \in \mathcal{C}'$ in polynomial time such that $G \cong H$ if and only if $G' \cong H'$. Such reductions were described for certain graph classes (e.g., chordal graphs [70]) and they were systematically studied in [17].

We say that ψ uses *vertex-gadgets*, if to every vertex $u \in V(G)$ (resp. $u \in V(H)$), it assigns a *vertex-gadget* $\mathcal{V}_u$, and these gadgets are subgraphs of G' (resp. of H'), and satisfy the following two conditions:

1. Every isomorphism $\pi : G \to H$ induces an isomorphism $\pi' : G' \to H'$ such that $\pi(u) = v$ implies $\pi'(\mathcal{V}_u) = \mathcal{V}_v$.
2. Every isomorphism $\pi' : G' \to H'$ maps vertex-gadgets to vertex-gadgets and induces an isomorphism $\pi : G \to H$ such that $\pi'(\mathcal{V}_u) = \mathcal{V}_v$ implies $\pi(u) = v$.

Theorem 4.1. *Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of graphs. Suppose that there exists a polynomial-time reduction ψ using vertex-gadgets from* GraphIso *of $\mathcal{C}$ to* GraphIso *of $\mathcal{C}'$. Then there exists a polynomial-time reduction from* ListIso *of $\mathcal{C}$ to* ListIso *of $\mathcal{C}'$.*

Proof. Let $G, H \in \mathcal{C}$ be an instance of ListIso. Using the reduction ψ, we construct the corresponding graphs $G', H' \in \mathcal{C}'$ with vertex-gadgets. We need to add lists for $V(G')$, we initiate them empty. Let $u \in V(G)$. To all vertices w of $\mathcal{V}_u$, we add $\bigcup_{v \in \mathfrak{L}(u)} V(\mathcal{V}_v)$ to $\mathfrak{L}(w)$. For the vertices of G' outside vertex-gadgets, we set the lists equal to the union of all remaining vertices of H'.

We want to argue that there exists a list-compatible isomorphism $\pi' : G' \to H'$, if and only if there exists a list-compatible isomorphism $\pi : G \to H$. If π exists, by the first assumption of the reduction, it induces π' which is list-compatible by our construction of lists. On the other hand, suppose that there exists a list-compatible isomorphism π'. By the second assumption, π' maps vertex-gadgets to vertex-gadgets and induces an isomorphism $\pi : G \to H$ which is list-compatible by our construction. $\square$

Corollary 4.2. *Let $\mathcal{C}$ be a class of graphs with* NP*-complete* ListIso*. Suppose that there exists a reduction ψ using vertex-gadgets from* GraphIso *of $\mathcal{C}$ to* GraphIso *of $\mathcal{C}'$. Then* ListIso *is* NP*-complete for $\mathcal{C}'$.* $\square$

Among others, this implies NP-completeness of ListIso for the following graph classes:

Corollary 4.3. *The problem* ListIso *is* NP*-complete for bipartite graphs, split and chordal graphs, chordal bipartite and strongly chordal graphs, trapezoid graphs, comparability graphs of dimension 4, grid intersection graphs, line graphs, and self-complementary graphs.*

Proof. We use Corollary 4.2 together with Theorem 1.1. We briefly describe GI-hardness reductions for every mentioned class. It is easy to check that, except for line graphs and self-complementary graphs, all these reductions use vertex-gadgets, where $\mathcal{V}_u = \{u\}$ for every $u \in V(G) \cup V(H)$.

Bipartite graphs. Assuming the graphs are not cycles, we subdivide every edge in the input graphs G and H.

Split and chordal graphs [70]. We subdivide every edge in G and H and add the complete graphs on the original vertices.

Chordal bipartite and strongly chordal graphs [90]. For bipartite graphs G and H, we subdivide all edges e_i twice, by adding vertices a_i and b_i, we add paths of length three from a_i to b_i, and we add the complete bipartite graph between a_i's and b_i's.

Trapezoid graphs [86]. For bipartite graphs G and H, we subdivide every edge and add the complete bipartite graph on the original vertices.

Comparability graphs of dimension at most 4 [58]. Assuming the graphs are not cycles, we replace every edge in G and H by a path of length 8.

Grid intersection graphs [89]. For bipartite graphs G and H, we subdivide every edge twice and add the complete bipartite graph on the original vertices.

Line graphs [95,45]. Assuming the graphs are not K_3 and $K_{1,3}$, we consider G' and H' being the line graphs of G and H. For every $u \in V(G)$, we put $\mathcal{V}_u = \{e : e \in E(G), u \in e\}$, and similarly for $u \in V(H)$. By Whitney Theorem [95], $G \cong H$ if and only if $G' \cong H'$, and it is easy to observe that it is a reduction using vertex-gadgets.

Self-complementary graphs [16]. A graph H is self-complementary if and only if $H \cong \overline{H}$ where $\overline{H}$ is the complement of H. Notice that the path of length 3 is self-complementary. We first describe a polynomial-time reduction from GraphIso of general graphs to GraphIso of self-complementary graphs.

For an arbitrary graph G, let $G_1, \dots, G_4$ be four copies of G. We construct G' as depicted in Fig. 3 as the disjoint union of G_1, $\overline{G_2}$, $\overline{G_3}$, and G_4. Further, we connect all vertices in $V(G_i)$ with $V(G_{i+1})$. The graph G' is self-complementary; see Fig. 3. All vertices of G_1 and G_4 have degrees at most $2n-1$ in G' and all vertices of $\overline{G_2}$ and $\overline{G_3}$ have degrees at least $2n$. Since all vertices of G_1 have common neighbors in $\overline{G_2}$, but there are no edges between $V(\overline{G_2})$ and $V(G_4)$, we can find these four copies of G in G'. Therefore, $G \cong H$ if and only if $G' \cong H'$. The reduction is clearly polynomial.

It remains to define vertex-gadgets. For every $u \in V(G)$, we put $\mathcal{V}_u = \{u_1\}$, where $u_1 \in V(G_1)$ is the copy of u. This reduction clearly uses vertex-gadgets. □

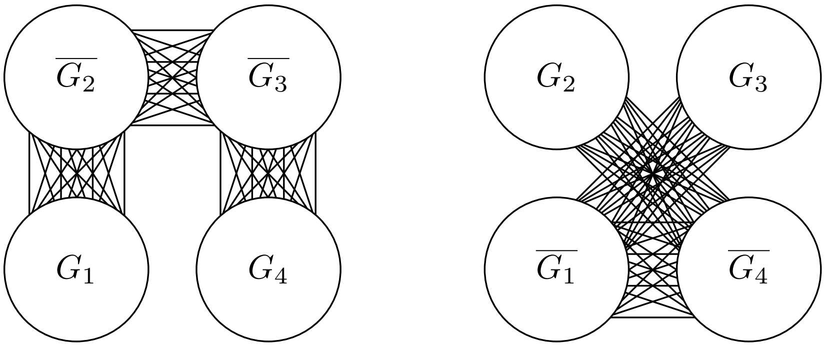


Fig. 3. On the left, the construction of G' from four copies of G. On the right, $\overline{G'}$ is depicting, showing that G' is a self-complementary graph.

We are not aware of any polynomial-time reduction for graph isomorphism used in the literature which cannot be easily modified to use vertex-gadgets. The reason is that most of the reductions use the following operations:

- Taking the complement of the graph.
- Replacing all vertex by small disjoint isomorphic gadgets.
- Replacing all edge by small disjoint isomorphic gadgets.
- Taking disjoint copies of the graph or its complement. (We can set vertex-gadgets equal to the vertices in one copy only.)
- Adding a universal vertex, adjacent to all vertices.
- Adding a complete subgraph on some vertices or a complete bipartite graph between two sets of vertices.

For instance, all reductions described in [17] can be easily modified to use vertex-gadgets.

5 NP-completeness for 3-regular Colored Graphs

Using group theory techniques, graph isomorphism can be solved in polynomial time for graphs of bounded degree [71] and for colored graphs with color classes of bounded size [37]. In this section, we modify the reduction of Lubiw [69] to show that ListIso remains NP-complete even for 3-regular colored graphs with color classes of size at most 8 and each list of size at most 3.

The reduction of Lubiw [69] is from 3-Sat, but we instead use 1-in-3 Sat which is NP-complete by Schaefer [82]: all literals are positive, each clause is of size 3 and a satisfying assignment has exactly one true literal in each clause. We show that an instance of 1-in-3 Sat can be solved using ListAut. We further simplify the reduction since a fixed-point free automorphism is not required for ListAut.

Variable Gadget. For each variable u_i, we construct the *variable gadget* H_i which consists of two isolated vertices $u_i(0)$ and $u_i(1)$; see Fig. 4a. We assign $\mathfrak{L}(u_i(0)) = \mathfrak{L}(u_i(1)) = \{u_i(0), u_i(1)\}$. There exist two list-compatible

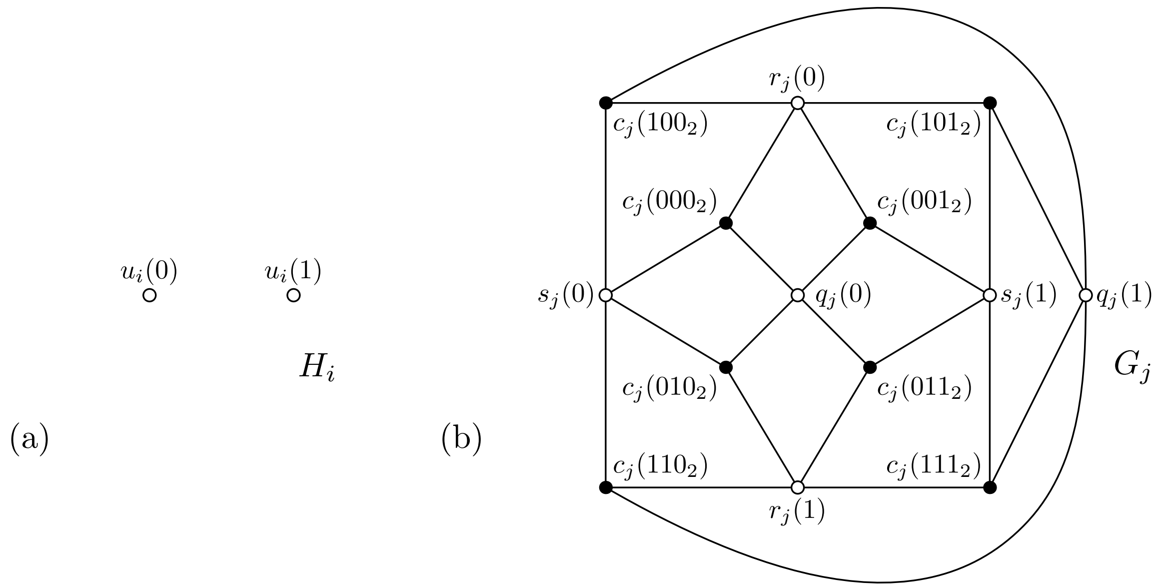


Fig. 4. (a) The variable gadget H_i. (b) The black vertices form the clause gadget G_j, adjacent to white vertices of variable gadgets.

automorphisms of H_i: the transposition α_i swapping $u_i(0)$ and $u_i(1)$ and the identity β_i fixing both $u_i(0)$ and $u_i(1)$.

Clause Gadget. Let c_j be a clause with the literals q_j, r_j, and s_j. For every such clause c_j, the *clause gadget* G_j consists of the isolated vertices $c_j(0), \dots, c_j(7)$. For every $k = 0, \dots, 7$, we consider its binary representation $k = abc_2$, for $a, b, c \in \{0, 1\}$. The vertex $c_j(k)$ has three neighbors $q_j(a)$, $r_j(b)$, and $s_j(c)$ belonging to the variable gadgets of its literals; see Fig. 4b. We assign the list

$$\mathfrak{L}(c_j(k)) = \{c_j(k \oplus 100_2), c_j(k \oplus 010_2), c_j(k \oplus 001_2)\},$$

where $\oplus$ denotes the bitwise XOR; i.e., $\mathfrak{L}(c_j(k))$ contains all $c_j(k')$ in which k' differs from k in exactly one bit. Let G be the resulting graph consisting of all variable and clause gadgets.

Lemma 5.1. *Suppose that π' is a partial automorphism of G obtained by choosing α_i or β_i on each variable gadget H_i. There exists a unique automorphism π extending π' such that $\pi(G_j) = G_j$.*

Proof. Let c_j be a clause with the literals q_j, r_j, and s_j. We claim that $\pi(c_j(k))$ is determined by the images of its neighbors. Recall that β_i preserves the vertices of H_i, but α_i swaps them. Therefore, one neighbor of $\pi(c_j(k))$ is different from the corresponding neighbor of $c_j(k)$ for every application of α_i on q_j, r_j and s_j. Let $p = abc_2$ such that $a = 1$, $b = 1$ and $c = 1$ if and only if α_i is applied on the variable gadget of q_j, r_j, and s_j, respectively. Then $\pi(c_j(k)) = c_j(k \oplus p)$; otherwise π would not be an automorphism. □

Lemma 5.2. *The* 1-IN-3 SAT *formula is satisfiable if and only if there exists a list-compatible automorphism of G.*

Proof. Let T be a truth value assignment satisfying the input formula. We construct a list-compatible automorphism π of G. If $T(u_i) = 1$, we put $\pi|_{H_i} = \alpha_i$, and if $T(u_i) = 0$, we put $\pi|_{H_i} = \beta_i$. By Lemma 5.1, this partial isomorphism has a unique extension to an automorphism π of G. It is list-compatible since T satisfies the 1-in-3 condition, so $\pi(c_j(k)) = c_j(k \oplus p)$, for $p \in \{100_2, 010_2, 001_2\}$.

For the other implication, let π be a list-compatible automorphism. Then $\pi|_{H_i}$ is either equal α_i, or β_i, which gives the values $T(u_i)$. By Lemma 5.1, $\pi(c_j(k)) = c_j(k \oplus p)$ and since π is a list-compatible isomorphism, we have $p \in \{100_2, 010_2, 001_2\}$. Therefore, exactly one literal in each clause is true, so all clauses are satisfied in T. $\square$

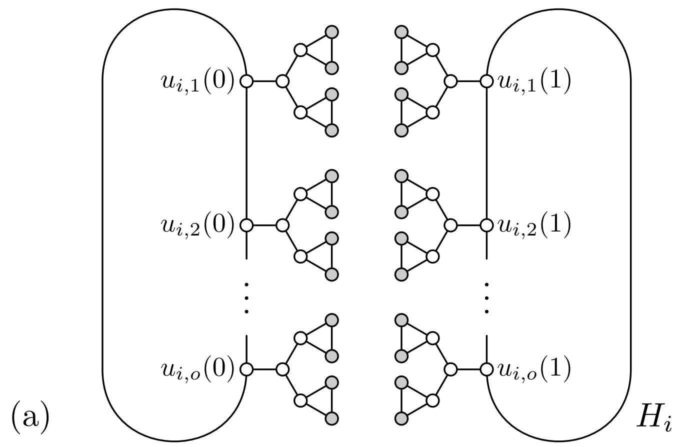
The described reduction clearly runs in polynomial-time, so we have established a proof of Theorem 1.1. For colored graphs, we require that automorphisms preserve colors. By altering the above reduction, we get the following:

Theorem 5.3. *The problem* ListIso *is* NP*-complete for 3-regular colored graphs for which each color class is of size at most 8 and each list is of size at most 3.*

Proof. We modify the graph G to a 3-regular graph. For a clause gadget G_j representing c_j, every vertex $c_j(k)$ already has degree 3. On the other hand, suppose that a variable u_i has o literals in the formula. Then both vertices of H_i have degrees $4o$, so we have to modify the variable gadgets.

We replace H_i by two cycles of length o, consisting of the vertices $u_{i,1}(0), \dots, u_{i,o}(0)$ and $u_{i,1}(1), \dots, u_{i,o}(1)$, respectively. To each of these vertices, we attach a small gadget depicted in Fig. 5a. We have $\mathfrak{L}(u_{i,t}(0)) = \mathfrak{L}(u_{i,t}(1)) = \{u_{i,t}(0), u_{i,t}(1)\}$. Again, there are two list-compatible automorphisms: α_i exchanging these two cycles by swapping $u_{i,t}(0)$ with $u_{i,t}(1)$, and β_i which is the identity fixing all $2o$ vertices. We note that when $o \le 2$, we get parallel edges or loops; if we want to avoid this, we may replace edges of two cycle by some 3-regular subgraphs.

Consider the attached gadgets to the vertices $u_{i,t}(0)$ and $u_{i,t}(1)$ corresponding to one literal of a clause c_j. Each vertex depicted in gray is adjacent to exactly one $c_j(k)$ of G_j, as depicted in Fig. 5b. Each k consists of three bits, denoted x, y and z (in some order). The bit x corresponds to this literal of u_i (i.e, x is the first bit for q_j being a literal of u_i, and so on). The gray vertices of gadgets attached to $u_{i,t}(j)$ are adjacent to $c_j(k)$ with $x = j$. Adjacent pairs of gray vertices are connected to $c_j(k)$ where k differs in the bit y. Non-adjacent pairs of gray vertices in one gadget are connected to $c_j(k)$ where k differs in the bit z.



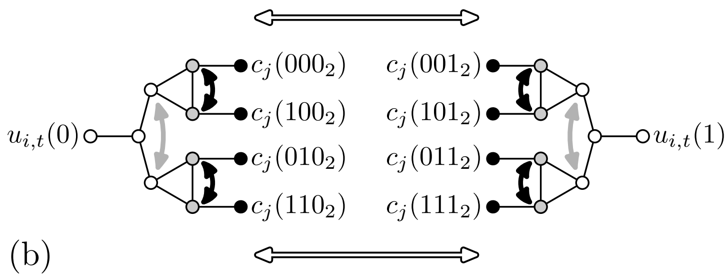


Fig. 5. (a) The variable gadget H_i. (b) The connection between H_i and G_j. Suppose that the variable u_i has a literal in the clause c_j, so $k = yzx_2$. We connect H_i with G_j as depicted. Suppose that an automorphism π maps $c_j(k)$ to $c_j(k \oplus p)$. We show the action of π on the vertices of H_i when $p = 001$ (in white), $p = 010$ (in gray), and $p = 100$ (in black).

In Fig. 5b, the action of $\mathbb{Z}_2^3$ is depicted. Lemma 5.1 translated to the modified definitions of variable gadgets which implies correctness of the reduction. The lists for the vertices of the attached gadgets are created as images of three depicted automorphisms; they clearly are of size at most 3.

The constructed graph G is 3-regular and all lists of G are of size at most 3. We color the vertices by the orbits of all list-compatible automorphisms and their compositions. Notice that each color class is of size at most 8. $\square$

With Lemma 3.5, we get a dichotomy for the maximum degree: ListIso can be solved in time $\mathcal{O}(\sqrt{n}\ell)$ for the maximum degree 2, and it is NP-complete for the maximum degree 3. Similarly, Lemma 3.2 implies a dichotomy for the list sizes: ListIso can be solved in time $\mathcal{O}(n+m)$ where all lists are of size 2, and it is NP-complete for lists of size at most 3. For the last parameter, the maximum size of color classes, there is a gap. Lemma 3.2 implies that ListIso can be solved in time $\mathcal{O}(n+m)$ when all color classes are of size 2 while it is NP-complete for size at most 8.

6 Trees

In this section, we modify the standard algorithm for tree isomorphism to solve list restricted isomorphism of trees. We may assume that both trees G and H

are rooted, otherwise we root them by their centers (and possibly subdivide the central edges). The algorithm for GraphIso process both trees from bottom to the top. Using dynamic programming, it computes for every vertex possible images using possible images of its children. This algorithm can be modified to ListIso.

Theorem 6.1. *The problem* ListIso *can be solved for trees in time* $\mathcal{O}(\sqrt{n}\ell)$.

Proof. We apply the same dynamic algorithm with lists and update these lists as we go from bottom to the top. After processing a vertex u, we compute an updated list $\mathfrak{L}'(u)$ which contains all elements of $\mathfrak{L}(u)$ to which u can be mapped compatibly with its descendants. To initiate, each leaf u of G has $\mathfrak{L}'(u) = \{w : w \text{ is a leaf and } w \in \mathfrak{L}(u)\}$.

Next, we want to compute $\mathfrak{L}'(u)$ and we know $\mathfrak{L}'(u_i)$ of all children $U = \{u_1, \dots, u_k\}$ of u. For each $w \in \mathfrak{L}(u)$ with k children $w_1, \dots, w_k$, we want to decide whether to put $w \in \mathfrak{L}'(u)$. Let $W = \{w_1, \dots, w_k\}$. Each u_i can be mapped to all vertices in $\mathfrak{L}'(u_i) \cap W$. We need to decide whether all u_i's can be mapped simultaneously. Therefore, we form a bipartite graph $B(U,W)$ between U and W: we put an edge $u_i w_j$ if and only if $w_j \in \mathfrak{L}'(u_i)$. Simultaneous mapping is possible if and only if there exists a perfect matching in this bipartite graph.

Let r be the root of G and r' be the root of H. We claim that there is a list-compatible isomorphism $\pi : G \to H$, if and only if $\mathfrak{L}'(r) = \{r'\}$. Suppose that π exists. When $\pi(u) = w$, its children U are mapped to W. Since this mapping is compatible with the lists, $w \in \mathfrak{L}(u)$, and the mapping of $u_1, \dots, u_k$ gives a perfect matching in $B(U,W)$. Therefore, $w \in \mathfrak{L}'(u)$, and by induction $r' \in \mathfrak{L}'(r)$. On the other hand, we can construct π from the top to the bottom. We start by putting $\pi(r) = r'$. When $\pi(u) = w$, we map its children U to W according to some perfect matching in $B(U,W)$ which exists from the fact that $w \in \mathfrak{L}'(u)$.

It remains to argue details of the complexity. We process the tree which takes time $\mathcal{O}(\ell)$ (assuming $n \le \ell$) and we process each list constantly many times which takes $\mathcal{O}(\ell)$. Suppose that we want to compute $\mathfrak{L}'(u)$. We consider all vertices $w^1, \dots, w^p \in \mathfrak{L}(u)$, and let W^j be the children of w^j. We go through all lists of $\mathfrak{L}'(u_1), \dots, \mathfrak{L}'(u_k)$ in linear time, and split them into sublists $\mathfrak{L}'(u_i^j)$ of vertices whose parent is w^j. Only these sublists are used in the construction of the bipartite graph $B(U, W^j)$. Using Lemma 2.1, we decide existence of a perfect matching in time $\mathcal{O}(\sqrt{k}\ell_j)$ which is at most $\mathcal{O}(\sqrt{n}\ell_j)$, where ℓ_j is the total size of all sublists $\mathfrak{L}'(u_i^j)$. When we sum this complexity for all vertices u, we get the total running time $\mathcal{O}(\sqrt{n}\ell)$. □

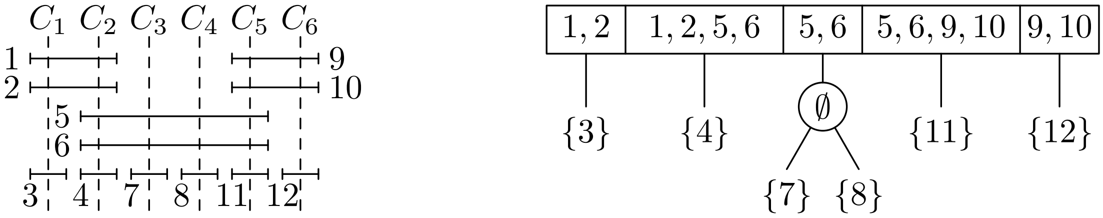


Fig. 6. An ordering of the maximal cliques, and the corresponding MPQ-tree. The P-nodes are denoted by circles, the Q-nodes by rectangles. There are four equivalent MPQ-trees.

7 Interval, Permutation and Circle Graphs

In this section, we prove that the standard algorithms solving GraphIso on interval, circle and permutation graphs can be modified to solve ListIso on them. The key idea is that the structure of these graph classes can be captured by graph-labeled trees which are unique up to isomorphism and which capture the structure of all automorphisms; see [57,58] and the references therein.

For interval graphs, we use MPQ-tree. For circle graphs, we use split trees. For permutation graphs, we use modular trees. On these trees, we apply a bottom-up procedure similarly as in the proof of Theorem 6.1. The key difference is that nodes correspond to either prime, or degenerate graphs. Degenerate graphs are simple and lead to perfect matchings in bipartite graphs. Prime graphs have a small number of automorphisms [57,58], so all of them can be tested.

7.1 Interval Graphs

To each interval graph G, a unique MPQ-tree T_G is assigned. Two interval graphs G and H are isomorphic if and only if T_G and T_H are equivalent, and these trees capture all isomorphisms. Therefore, we apply a bottom-up procedure to test ListIso for MPQ-trees, similarly as in Theorem 6.1.

MPQ-trees. Booth and Lueker [11] invented a data structure called a *PQ-tree* which capture the structure of an interval graph. We use *modified PQ-trees* (MPQ-trees) due to Korte and Möhring [62]. Let G be an interval graph. A rooted tree T is an MPQ-tree if the following holds. It has two types of inner nodes: *P-nodes* and *Q-nodes*. For every inner node, its children are ordered from left to right. Each P-node has at least two children and each Q-node at least three. The leaves of T correspond one-to-one to the maximal cliques in G.

Two MPQ-trees are *equivalent* if one can be obtained from the other by a sequence of two *equivalence transformations*: (i) an arbitrary permutation of the order of the children of a P-node, and (ii) the reversal of the order of the children of a Q-node. Booth and Lueker [11] proved the existence and uniqueness of PQ-trees (up to equivalence transformations); see Fig. 6.

We assign subsets of $V(G)$, called *sections*, to the nodes of T; see Fig. 6. The leaves and the P-nodes have each assigned exactly one section while the Q-nodes have one section per child. We assign these sections as follows:

- For a leaf L, the section $\mathrm{sec}(L)$ contains those vertices that are only in the maximal clique represented by L, and no other maximal clique.
- For a P-node P, the section $\mathrm{sec}(P)$ contains those vertices that are in all maximal cliques of the subtree of P, and no other maximal clique.
- For a Q-node Q and its children $T_1, \dots, T_n$, the section $\mathrm{sec}_i(Q)$ contains those vertices that are in the maximal cliques represented by the leaves of the subtree of T_i and also some other T_j, but not in any other maximal clique outside the subtree of Q. We put $\mathrm{sec}(Q) = \mathrm{sec}_1(Q) \cup \dots \cup \mathrm{sec}_n(Q)$.

Each vertex appears in sections of exactly one node and in the case of a Q-node in consecutive sections. Two vertices are in the same sections if and only if they belong to precisely the same maximal cliques. Figure 6 shows an example. MPQ-tree can be constructed in time [62].

Testing ListIso. Let G and H be two isomorphic interval graphs. From [58, Lemma 4.3], it follows that T_G and T_H are equivalent, and every isomorphism $\pi : G \to H$ is obtained by an equivalence transformation of T_G and some permutation of the vertices in identical sections. Now, we are ready to show ListIso can be solved on interval graphs in time $\mathcal{O}(\sqrt{n}\ell + m)$:

Theorem 7.1. *The problem* ListIso *can be solved for interval graphs in time* $\mathcal{O}(\sqrt{n}\ell + m)$.

Proof. We proceed similarly as in Theorem 6.1. We compute MPQ-trees representing T_G and T_H representing the graphs G and H in linear time [62]. Then we compute lists $\mathfrak{L}(N)$ for every node N of T_G from the bottom. We distinguish three types of nodes.

- *Leaf nodes.* Let L_G be a leaf node in T_G and let L_H be a leaf node in T_H. Then $L_H \in \mathfrak{L}(L_G)$ if there exists a list-compatible isomorphism between the induced complete subgraphs $G[\mathrm{sec}(L_G)]$ and $H[\mathrm{sec}(L_H)]$.
- *P-nodes.* Let N and M be P-nodes of T_G and T_H, respectively. We want to decide whether $M \in \mathfrak{L}(N)$. Let $N_1, \dots, N_k$ be the children of N and let $M_1, \dots, M_k$ be the children of M_k. We construct a bipartite graph similarly as in Theorem 6.1. Then $M \in \mathfrak{L}(N)$ if there exists a perfect matching in the bipartite graph and a perfect matching between the lists of $G[\mathrm{sec}(N)]$ and $H[\mathrm{sec}(M)]$ (which are complete graphs).
- *Q-nodes.* Let N and M be Q-nodes of T_G and T_H and let $N_1, \dots, N_k$ and $M_1, \dots, M_k$ be their children. Here we have at most two possible isomorphisms. In particular, an isomorphism can either map the subtree of N_i on

the subtree of M_i, or in the reversed order, and we can test for both possibilities whether the lists $\mathfrak{L}(N_i)$ are compatible. Moreover, we consider all sets of intervals belonging to exactly the same sections of the Q-node, and we test by perfect matchings between pairs of them whether there exists a list-compatible isomorphism between them.

The MPQ-trees have $\mathcal{O}(n)$ nodes and $\mathcal{O}(n)$ intervals in their sections. For leaf nodes and P-nodes, the analysis is exactly the same as in the proof of Theorem 6.1. For Q-nodes, we just test two possible mappings and bipartite matchings for sections. We get the total running time $\mathcal{O}(\sqrt{n}\ell + m)$. $\square$

7.2 Permutation Graphs

A *module* M of a graph G is a set of vertices such that each $x \in V(G) \setminus M$ is either adjacent to all vertices in M, or to none of them. See Fig. 7a for examples. A module M is called *trivial* if $M = V(G)$ or $|M| = 1$, and *non-trivial* otherwise. If M and M' are two disjoint modules, then either the edges between M and M' form the complete bipartite graph, or there are no edges at all; see Fig. 7a. In the former case, M and M' are called *adjacent*, otherwise they are *non-adjacent*.

Let $\mathcal{P} = \{M_1, \dots, M_k\}$ be a *modular partition* of $V(G)$, i.e., each M_i is a module of G, $M_i \cap M_j = \emptyset$ for every $i \neq j$, and $M_1 \cup \dots \cup M_k = V(G)$. We define the *quotient graph* $G/\mathcal{P}$ with the vertices $m_1, \dots, m_k$ corresponding to $M_1, \dots, M_k$ where $m_i m_j \in E(G/\mathcal{P})$ if and only if M_i and M_j are adjacent. In other words, the quotient graph is obtained by contracting each module M_i into the single vertex m_i; see Fig. 7b.

Modular Decomposition. To decompose G, we find some modular partition $\mathcal{P} = \{M_1, \dots, M_k\}$, compute $G/\mathcal{P}$ and recursively decompose $G/\mathcal{P}$ and each $G[M_i]$. The recursive process terminates on *prime graphs* which are graphs containing only trivial modules. There might be many such decompositions for different choices of $\mathcal{P}$ in each step. In 1960s, Gallai [38] described the *modular decomposition* in which special modular partitions are chosen and which encodes all other decompositions.

The key is the following observation. Let M be a module of G and let $M' \subseteq M$. Then M' is a module of G if and only if it is a module of $G[M]$.

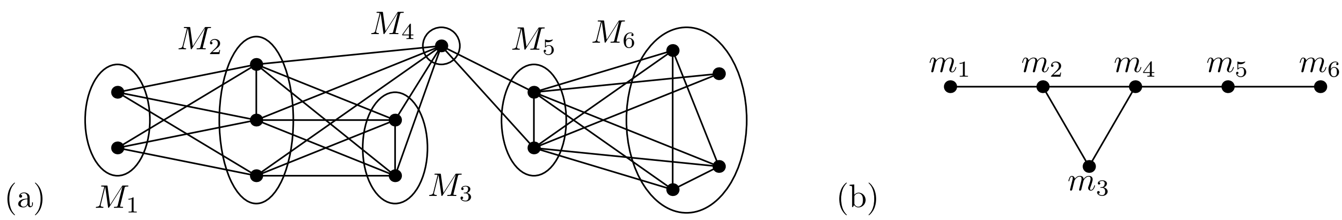
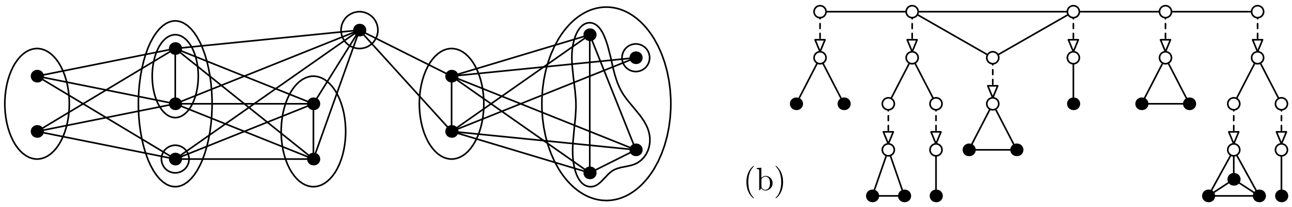


Fig. 7. (a) A graph G with a modular partition $\mathcal{P}$. (b) The quotient graph $G/\mathcal{P}$ is prime.



(a) (b)

Fig. 8. (a) The graph G from Fig. 7 with the modular partitions used in the modular decomposition. (b) The modular tree T of G, the marker vertices are white, the tree edges are dashed.

A graph G is called *degenerate* if it is K_n or $\overline{K}_n$. We construct the modular decomposition of a graph G in the following way, see Fig. 8a for an example:

- If G is a prime or a degenerate graph, then we terminate the modular decomposition on G. We stop on degenerate graphs since every subset of vertices forms a module, so it is not useful to further decompose them.
- Let G and $\overline{G}$ be connected graphs. Gallai [38] shows that the inclusion maximal proper subsets of $V(G)$ which are modules form a modular partition $\mathcal{P}$ of $V(G)$, and the quotient graph $G/\mathcal{P}$ is a prime graph; see Fig. 7. We recursively decompose $G[M]$ for each $M \in \mathcal{P}$.
- If G is disconnected and $\overline{G}$ is connected, then every union of connected components is a module. Therefore the connected components form a modular partition $\mathcal{P}$ of $V(G)$, and the quotient graph $G/\mathcal{P}$ is an independent set. We recursively decompose $G[M]$ for each $M \in \mathcal{P}$.
- If $\overline{G}$ is disconnected and G is connected, then the modular decomposition is defined in the same way on the connected components of $\overline{G}$. They form a modular partition $\mathcal{P}$ and the quotient graph $G/\mathcal{P}$ is a complete graph. We recursively decompose $G[M]$ for each $M \in \mathcal{P}$.

Modular Tree. We encode the modular decomposition by the *modular tree* T. The modular tree T is a graph with two types of vertices (normal and *marker vertices*) and two types of edges (normal and *directed tree edges*). The directed tree edges connect the prime and degenerate graphs encountered in the modular decomposition (as quotients and terminal graphs) into a rooted tree.

We give a recursive definition. Every modular tree has an induced subgraph called *root node*. If G is a prime or a degenerate graph, we define $T = G$ and its root node equals T. Otherwise, let $\mathcal{P} = \{M_1, \dots, M_k\}$ be the used modular partition of G and let $T_1, \dots, T_k$ be the modular trees corresponding to $G[M_1], \dots, G[M_k]$. The modular tree T is the disjoint union of $T_1, \dots, T_k$ and of $G/\mathcal{P}$ with the marker vertices $m_1, \dots, m_k$. To every graph T_i, we add a new marker vertex m'_i such that m'_i is adjacent exactly to the vertices of the root node of T_i. We further add a tree edge oriented from m_i to m'_i. For an example, see Fig. 8b.

The modular tree of G is unique. The graphs encountered in the modular decomposition are called *nodes of* T, or alternatively root nodes of some modular trees in the construction of T. For a node N, its subtree is the modular tree which has N as the root node. *Leaf nodes* correspond to the terminal graphs in the modular decomposition, and *inner nodes* are the quotients in the modular decomposition. All vertices of G are in leaf nodes and all marker vertices correspond to modules of G. *All inner nodes consist of marker vertices.*

Testing ListIso. Now, we are ready to show that the problem ListIso can be decided in $\mathcal{O}(\sqrt{n}\ell + m)$ time for permutation graphs.

Theorem 7.2. *The problem* ListIso *can be solved for permutation graphs in time* $\mathcal{O}(\sqrt{n}\ell + m)$.

Proof. For input graph G and H, we first compute the modular trees T_G and T_H, respectively, in time $\mathcal{O}(n+m)$ [74]. We again apply the idea of Theorem 6.1. We compute the list $\mathfrak{L}(N)$ for every node N of T_G. Note that all inner nodes consist only of marker vertices which have no lists. Therefore, we first compute $\mathfrak{L}(L)$, for every leaf node. A leaf node K is in $\mathfrak{L}(L)$ if every non-marker vertex of L has a non-marker vertex of K in its list. These candidate nodes for $\mathfrak{L}(L)$ can be found in linear time in the total size of lists by iterating through the lists of vertices of L.

Suppose that a node N has the children $N_1, \dots, N_k$ with computed lists $\mathfrak{L}(N_1), \dots, \mathfrak{L}(N_k)$ and M has the children $M_1, \dots, M_k$. There exist a list-compatible isomorphism mapping the subtree of N_i to the subtree of M_j if and only if $M_j \in \mathfrak{L}(N_i)$. Moreover these subtrees have to be compatible with a list isomorphism from N to M. We compute $\mathfrak{L}(N)$ according to the type of N.

- *Degenerate nodes.* For degenerate nodes, we proceed similarly as for trees in Theorem 6.1. We construct a bipartite graph between the nodes nodes $N_1, \dots, N_k$ and $M_1, \dots, M_k$ and test for a perfect matching using Lemma 2.1.
- *Prime nodes.* For prime nodes, there are at most four possible isomorphisms mapping N to M [58, Lemma 6.6]. We test for these four possible isomorphisms π whether $\pi(M_i) \in \mathfrak{L}(M_i)$ for every M_i.

A list compatible isomorphism exists if $M \in \mathfrak{L}(N)$, for the root nodes N and M of T_G and T_H. The correctness of the algorithm follows from the fact that all automorphisms of a permutation graph are captured by the modular tree [58]. A similar argument as in the proofs of Theorems 6.1 and 7.1 gives the running time. □

7.3 Circle Graphs

For a given circle graph, we define the split tree which captures its automorphism group. A *split* is a partition (A, B, A', B') of $V(G)$ such that:

- For every $a \in A$ and $b \in B$, we have $ab \in E(G)$.
- There are no edges between A' and $B \cup B'$, and between B' and $A \cup A'$.
- Both sides have at least two vertices: $|A \cup A'| \geq 2$ and $|B \cup B'| \geq 2$.

The split decomposition of G is constructed by taking a split of G and replacing G by the graphs G_A and G_B defined as follows. The graph G_A is created from $G[A \cup A']$ together with a new *marker vertex* m_A adjacent exactly to the vertices in A. The graph G_B is defined analogously for B, B' and m_B; see Fig. 9a. The decomposition is then applied recursively on G_A and G_B. Graphs containing no splits are called *prime graphs.* We stop the split decomposition also on *degenerate graphs* which are complete graphs K_n and stars $K_{1,n}$. A split decomposition is called *minimal* if it is constructed by the least number of splits. Cunningham [20] proved that the minimal split decomposition of a connected graph is unique.

Split tree. The *split tree* T representing a graph G encodes the minimal split decomposition. A split tree is a graph with two types of vertices (normal and marker vertices) and two types of edges (normal and tree edges). We initially put $T = G$ and modify it according to the minimal split decomposition. If the minimal decomposition contains a split (A, B, A', B') in G, then we replace G in T by the graphs G_A and G_B, and connect the marker vertices m_A and m_B by a *tree edge* (see Fig. 9a). We repeat this recursively on G_A and G_B; see Fig. 9b. Each prime and degenerate graph is a *node* of the split tree. A node that is incident with exactly one tree edge is called a *leaf node.*

Since the minimal split decomposition is unique, we also have that the split tree is unique. Further, each automorphism π of G induces an automorphism π' of the split tree T representing G. Similarly as for trees, there exists a *center* of T which is either a tree edge, or a prime or degenerate node. The automorphism π' preserves the center, so we can regard T as rooted by the

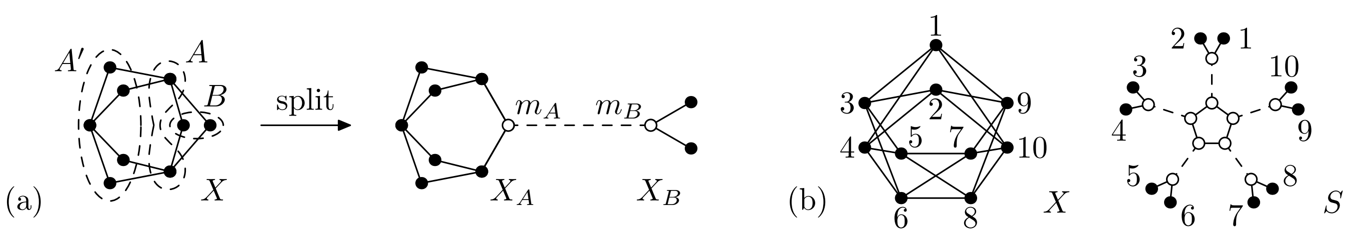


Fig. 9. (a) An example of a split of the graph G. The marker vertices are depicted in white. The tree edge is depicted by a dashed line. (b) The split tree S of the graph G. We have that $\mathrm{Aut}(S) \cong \mathbb{Z}_2^5 \rtimes \mathbb{D}_5$.

center. Every automorphism of G can be reconstructed from the root of T to the bottom.

Testing ListIso. Next, we show that the problem ListIso can be solved on circle graphs in time $\mathcal{O}(\sqrt{n}\ell + m \cdot \alpha(m)$.

Theorem 7.3. *The problem* ListIso *can be solved for circle graphs in time* $\mathcal{O}(\sqrt{n}\ell + m \cdot \alpha(m))$*, where* α *is the inverse Ackermann function.*

Proof. For input graph G and H, we first compute the split trees T_G and T_H, in time $\mathcal{O}((n+m) \cdot \alpha(n+m))$ [39]. We assume that the trees T_G and T_H are rooted and we can also assume that the roots are prime or degenerate nodes. We again apply the idea of Theorem 6.1.

We compute the list $\mathfrak{L}(N)$ for every node N of T_G. Let M be a leaf node of T_H and let $m_N \in V(N)$ and $m_M \in V(M)$ be the marker vertices incident to a tree edge closer to the root. Then M is in $\mathfrak{L}(N)$ if there is a list-compatible isomorphism from N to M which maps m_N to m_M.

Suppose that a node N has the children $N_1, \dots, N_k$ with computed lists $\mathfrak{L}(N_1), \dots, \mathfrak{L}(N_k)$ and M has the children $M_1, \dots, M_k$. There exist a list-compatible isomorphism mapping the subtree of N_i to the subtree of M_j if and only if $M_j \in \mathfrak{L}(N_i)$. Moreover these subtrees have to be compatible with an isomorphism from N to M. We compute $\mathfrak{L}(N)$ according to the type of N.

- *Degenerate nodes.* For degenerate nodes, we proceed similarly as for trees in Theorem 6.1. We construct a bipartite graph between the nodes nodes $N_1, \dots, N_k$ and $M_1, \dots, M_k$ and test for a perfect matching using Lemma 2.1.
- *Prime nodes.* For prime nodes, there are at most four possible isomorphisms mapping m_N to m_M [57, Lemma 5.6]. We test those four possible isomorphisms, construct four bipartite graphs and test existence of perfect matchings.
- *The root node.* If it is degenerate, we proceed as above. If it is prime, then its automorphism groups is a subgroup of a dihedral group [57, Lemma 5.5]; essentially it behaves as a cycle. Therefore, we approach it similarly as in Lemma 3.4.

A list compatible isomorphism exists if $M \in \mathfrak{L}(N)$, for the root nodes N and M of T_G and T_H.

The correctness of the algorithm follows from the fact that all automorphisms of a circle graph are captured by the split tree [57]. The running time can be argued as in Theorems 6.1, 7.1, and 7.2. □

8 Planar Graphs

In this section, we describe how to solve ListIso on planar graphs.

For the purpose of this section, we need to consider a more general definition of a graph. We work with multigraphs and we admit pendant edges with free ends (which are edges attached to single vertices). Also, each edge uv gives rise to two incident darts,[4] one attached to u, the other to v. Every isomorphism maps vertices and darts while preserving incidencies. We consider the problem ListIso with lists on both vertices and darts.

3-connected Planar Graphs. We have a unique embedding into the sphere (up to the reflection). This embeddings can be described in the language of flags, which are pairs (d, f) where d is a dart and f is an incident face. Every automorphism of G corresponds either to a direct map automorphism, or to a indirect map automorphism (composed with a reflection). In particular, $\mathrm{Aut}(G) \cong \mathrm{Aut}(\mathcal{M})$ acts semiregularly on the set of flags of $\mathcal{M}$. See [56] for more details and references. Therefore, if the images of two consecutive darts in the rotational scheme are set, the entire mapping is determined and we just need to check whether it is an isomorphism.

Lemma 8.1. *The problem* ListIso *(with lists on both vertices and darts) can be solved for 3-connected planar graphs in time $\mathcal{O}(\ell)$.*

Proof. We start by computing embeddings of both G and H, in time $\mathcal{O}(n)$. It remains to decide whether there exists a list-compatible isomorphism which has to be a map isomorphism. By Euler Theorem, we know that the average degree is less than six. Consider all vertices of degree at most 5, let u be such a vertex with a smallest list, and let $k = |\mathfrak{L}(u)|$. We have $\ell = \Omega(kn)$ and we show that we can decide existence of a list-compatible isomorphism in time $\mathcal{O}(kn)$.

We test all possible mappings $\pi : G \to H$ having $\pi(u) \in \mathfrak{L}(u)$. For each, we have at most 10 possible ways how to extend this mapping on the neighbors of u, and the rest of the mapping is uniquely determined by the embeddings and can be computed in time $\mathcal{O}(n)$. In the end, we test whether the constructed mapping π is an isomorphism and whether it is list-compatible. $\square$

3-connected Reduction. Seminal papers by Mac Lane [72] and Trakhtenbrot [87] introduced reduction which decomposes a graph into its *3-connected*

[4] In the standard definition of graphs, the primary objects are vertices and the secondary objects are edges. The definition via darts, from algebraic and topological graph theory, makes edges (or more precisely their halves) the primary objects, while the vertices are secondary objects. It is important because we need to distinguish between an isomorphism which maps an edge and which also reflects it.

components. This idea was further extended in [88,51,48,19,92,8], also known in the literature under the name of SPQR trees [25,26,27,44]. We use an augmentation described in [31,32,56] which behaves well with respect to automorphism groups.

The reduction is constructed by replacing atoms by colored possibly directed edges. *Atoms* are subgraphs of the following three types (for precise definitions, see [32]):

- *Block atom.* Either a pendant star, or a pendant block with attached single pendant edges.
- *Proper atom.* Inclusion minimal subgraphs separated by a 2-cut.
- *Dipoles.* They are two vertices together with all (at least two) parallel edges between them.

Further, each atom A has the *boundary* ∂A (of size at most 2) and the *interior* $\mathring{A}$. A graph is called *essentially 3-connected* if it is a 3-connected graph with attached single pendant edges attached. Similarly, a graph is called *essentially a cycle* if it is a cycle with attached single pendant edges. It follows from [32] that each block atom is either a star, or essentially a cycle, or essentially 3-connected, or K_2 with a single pendant edge attached. For a proper atom A with $\partial A = \{u, v\}$, we denote by A^+ the graph with the added edge uv. The graph A^+ is always either essentially a cycle, or essentially 3-connected.

A proper atom or a dipole A is called *symmetric* if there exists an automorphism in $\mathrm{Aut}(A)$ exchanging ∂A, and *asymmetric* otherwise. Every block atom is symmetric by the definition. The reduction is done by finding all atoms in G (by [32], they have disjoint interiors) and replacing their interiors by edges. Further, we color these edges to code isomorphism types of atoms, and we use directed edges for asymmetric atoms. Block atoms are replaced by pendant edges with free ends.

We repeat this reductions over and over, which gives a sequence of graphs $G = G_0, \dots, G_r$ where G_r is called *primitive* and contains no atoms. By [32], it is either essentially 3-connected, essentially a cycle, K_2 possibly with attached single pendant edges, or K_1 with an attached single pendant edge with a free end. Further, this reduction process can be encoded by the *reduction tree* T_G; see Fig. 10 for an example. It is a rooted tree, where each node is labeled by a graph. The root of T_G is the primitive graph G_r. The other nodes correspond to atoms obtained in the reductions. When the interior of an atom A is replaced by an edge e, we attach the node representing to A to the edge e.

It easily follows that the reduction tree is unique and canonical. Further each automorphism π of G induces automorphisms of $G_1, \dots, G_r$ by permuting edges exactly as atoms. Therefore, it induces an automorphism π' of T_G which

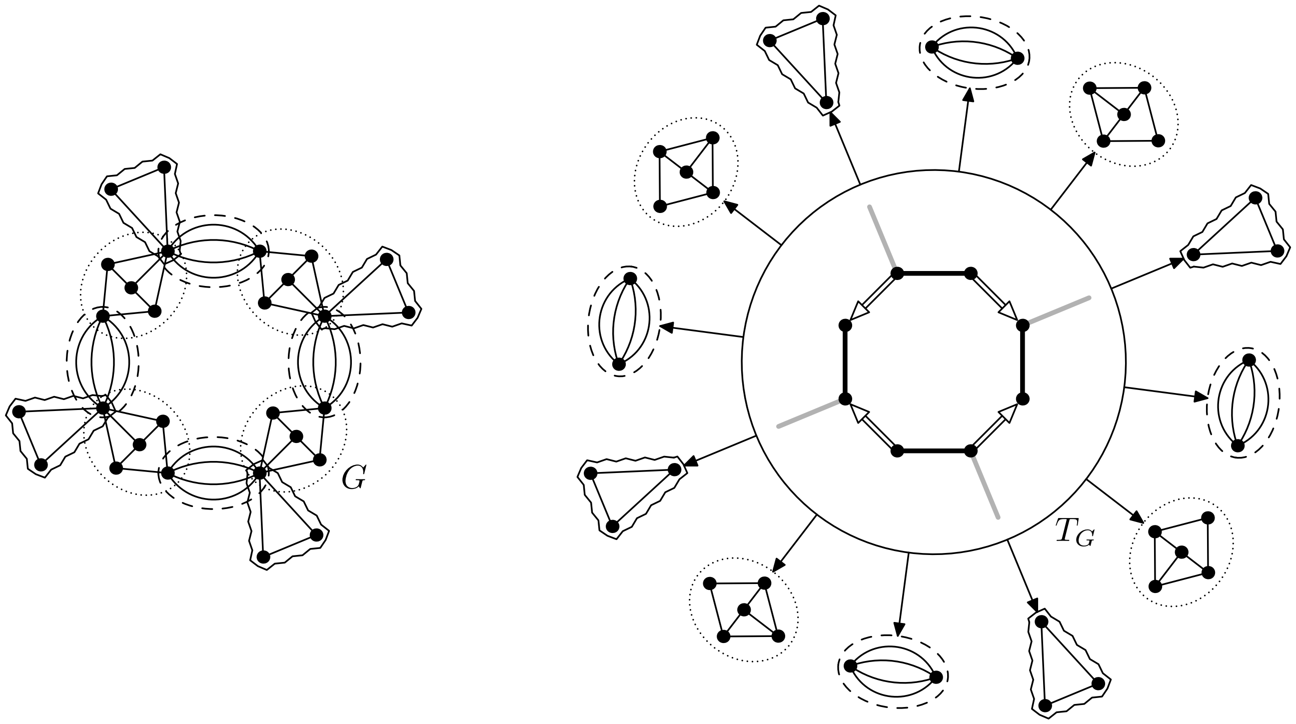


Fig. 10. A graph G together with its reduction tree T_G.

permutes the nodes of isomorphic graphs, and when it maps a colored edge e to a colored edge e', it maps the subtree attached to e to the isomorphic subtree attached to e'. And every automorphism of G can be constructed in this way, from the root of T_G to the bottom. We can use this to solve LISTISO.

Theorem 8.2. *Let $\mathcal{C}$ be a class of connected graphs closed under contractions and taking connected subgraphs. Suppose that* LISTISO *with lists on both vertices and darts can be solved for 3-connected graphs in $\mathcal{C}$ in time $\varphi(n, m, \ell)$. We can solve* LISTISO *on $\mathcal{C}$ in time $\mathcal{O}(\sqrt{m}\ell + m + \varphi(n, m, \ell))$.*

Proof. We compute reduction trees T_G and T_H for both G and H in time $\mathcal{O}(n+m)$. We apply the idea of Theorem 6.1 to test list-compatible isomorphism of T_G and T_H. We compute the lists $\mathfrak{L}(N)$ for the nodes N of T_G, from the bottom to the root. A node $M \in \mathfrak{L}(N)$ if there exists a list-compatible isomorphism from N to M mapping ∂N to ∂M and there exists list-compatible isomorphism between attached subtrees. (Further, if $|\partial N| = |\partial M| = 2$, we remember in $\mathfrak{L}(N)$ which of both possible mappings of ∂N to ∂M can be extended as list-compatible isomorphisms.)

Suppose that N has the children $N_1, \dots, N_k$ with computed lists and M has the children $M_1, \dots, M_k$. There exists a list-compatible isomorphism mapping the subtree of N_i to the subtree of M_j, if and only if $M_j \in \mathfrak{L}(N_i)$. The difference from Theorem 6.1 is these subtrees have to be compatible with a list-isomorphism from N to M; so it depends on the structure of the nodes N and M.

There, we compute $\mathfrak{L}(N)$ differently according to the type of N:

- *Star block atoms or dipoles.* For star block atoms, similarly as in Theorem 6.1, we construct a bipartite graph between $N_1, \dots, N_k$ and $M_1, \dots, M_k$ and test existence of a perfect matching using Lemma 2.1. For dipoles, we test two possible isomorphisms, construct two bipartite graph and test existence of perfect matchings.
- *Non-star block or proper atoms.* We modify the lists of ∂N to the vertices of ∂M only. (When they are proper atoms, we run this in two different ways.) We encode the lists $\mathfrak{L}(N_1), \dots, \mathfrak{L}(N_k)$ by lists on the corresponding darts of N (depending on which of two possible list-isomorphisms of ∂N_i are possible), and we remove single pendant edges, and intersect their lists with the lists of the incident vertices. For a proper atom, we further consider N^+ and M^+ with added edges e and f such that $\mathfrak{L}(e) = \{f\}$. If the nodes are K_2 or cycles, and we can test existence of a list-compatible isomorphism using Lemma 3.4. If both are 3-connected, we can test it by our assumption in time $\varphi(n', m', \ell')$. If this list-compatible isomorphism exists, we add M to $\mathfrak{L}(N)$.
- *The root primitive graphs.* We use the same approach as above, ignoring the part about ∂N and ∂M.

A list-compatible isomorphism from G to H exists, if and only if $M \in \mathfrak{L}(N)$ for the root nodes N and M of T_G and T_H.

The correctness of the algorithm can be argued from the fact that all automorphisms are captured by the reduction trees [32], inductively from the top to the bottom as in Theorem 6.1. It remains to discuss the running time. The reduction trees can be computed in linear time [48]. When computing $\mathfrak{L}(N)$, we first consider the lists of all vertices and edges of N. A node M is a candidate for $\mathfrak{L}(N)$, if every vertex and every edge of N has a vertex/edge of M in its list. Therefore, we can find all these candidate nodes by iterating these lists, in linear time with respect to their total size. Let M be one of them, and let $n' = |V(N)|$, $m' = |E(N)|$ and ℓ' be the total size of lists of the vertices and edges of N when restricted only to the vertices and edges of M. Either we construct a bipartite graph and test existence of a perfect matching in time $\mathcal{O}(\sqrt{m}'\ell')$, or we test existence of a list-compatible isomorphism in time $\varphi(n', m', \ell')$. The total running time spent on the tree is $\mathcal{O}(\ell)$, the total running time spent testing perfect matchings is $\mathcal{O}(\sqrt{m}\ell)$, and the total running time testing list-compatible isomorphisms of 3-connected graphs is $\mathcal{O}(\varphi(n, m, \ell))$.

□

General Planar Graphs. By putting both results together, we get the following:

Theorem 8.3. *The problem* ListIso *can be solved for planar graphs in time* $\mathcal{O}(\sqrt{n}\ell)$.

Proof. If G and H are connected, we use Theorem 8.2. By Lemma 8.1, the function $\varphi(n, m, \ell)$ is $\mathcal{O}(\ell)$. If G and H are disconnected, we apply Lemma 3.3 on all connected components of G and H, and by analysing the proof, the total running time is $\mathcal{O}(\sqrt{n}\ell)$. □

9 Bounded Genus Graphs

In this section, we describe an FPT algorithm solving ListIso when parameterized by the Euler genus g. We modify the recent paper of Kawarabayashi [55] solving graph isomorphism in linear time for a fixed genus g. The harder part of this paper are structural results, described below, which transfer to list-compatible isomorphisms without any change. Using these structural results, we can build our algorithm.

Theorem 9.1. *For every integer g, the problem* ListIso *can be solved on graphs of Euler genus at most g in time $\mathcal{O}(\sqrt{n}\ell)$.*

Proof. See [55, p. 14] for overview of the main steps. We show that these steps can be modified to deal with lists. We prove this result by induction on g, where the base case for $g = 0$ is Theorem 8.3. Next, we assume that both graphs G and H are 3-connected, otherwise we apply Theorem 8.2. By [55, Theorem 1.2], if G and H have no polyhedral embeddings, then the face-width is at most two.

Case 1: G and H have polyhedral embeddings. Following [55, Theorem 1.2], we have at most $f(g)$ possible embeddings of G and H. We choose one embedding of G and we test all embeddings of H. It is known that the average degree is $\mathcal{O}(g)$. Therefore, we can apply the same idea as in the proof of Lemma 8.1 and test isomorphism of all these embeddings in time $\mathcal{O}(\ell)$.

Case 2: G and H have no polyhedral embedding, but have embeddings of face-width exactly two. Then we split G into a pair of graphs (G', L). The graph L are called *cylinders* and the graph G' correspond to the remainder of G. The following properties hold [55, p. 5]:

- We have $G = G' \cup L$ and for $\partial L = V(G' \cap L)$, we have $|\partial L| = 4$.
- The graph G' can be embedded to a surface of genus at most $g - 1$, and L is planar [55, p. 4].
- This pair (G', L) is canonical, i.e., every isomorphism from G to H maps (G', L) to another pair (H', L') in H.

It is proved [55, Theorem 5.1] that there exists some function $q'(g)$ bounding the number of these pairs both in G and H, and can be found in time $\mathcal{O}(n)$. We fix a pair (G', L) in G and iterate over all pairs (H'_i, L'_i) in H. Following [55, p. 36], we get that $G \cong H$, if and only if there exists a pair (H'_i, L'_i) in H such that $G' \cong H'_i$, $L \cong L'_i$, and $G' \cap L$ is mapped to $H'_i \cap L'_i$. To test this, we run at most $2q'(g)$ instances of ListIso on smaller graphs with modified lists.

Suppose that we want to test whether $G' \cong H'_i$ and $L \cong L'_i$. First, we modify the lists: for $u \in V(G')$, put $\mathfrak{L}'(u) = \mathfrak{L}(u) \cap H'_i$, and for $v \in V(L)$, put $\mathfrak{L}'(v) = \mathfrak{L}(v) \cap L'_i$, and similarly for lists of darts. Further, for all vertices $u \in \partial L$ in both G' and L, we put $\mathfrak{L}'(u) = \mathfrak{L}(u) \cap \partial L$. We test existence of list-compatible isomorphisms from G' to H'_i and from L to L'_i. There exists a list-compatible isomorphism from G to H, if and only if these list-compatible isomorphisms exist at least for one pair (H'_i, L'_i).

We note that when $g = 2$, a special case is described in [55, Theorem 5.3], which is slightly easier and can be modified similarly.

Case 3: G and H have no polyhedral embedding and have only embeddings of face-width one. Let V be the set of vertices in G such that for each $u \in V$, there exists a non-contractible curve passing only through u. By [55, Lemma 6.3], $|V| \le q(g)$ for some function q. For u, the non-contractible curve divides its edges to two sides, so we can cut G at u, and split the incident edges. We obtain a graph G' which can be embedded to a surface of genus at most $g-1$.

By [55, Lemma 6.3], we can find all these vertices V and V' in G and H in time $\mathcal{O}(n)$. We choose $u \in V$ arbitrarily, and we test all possible vertices $v \in V'$. Let G' be constructed from G by splitting u into new vertices u' and u'', and similarly H' be constructed from H by splitting v into new vertices v' and v''. In [55, p. 36], it is stated that $G \cong H$, if and only if there exists a choice of $v \in V'$ such that $G' \cong H'$ and $\{u', u''\}$ is mapped to $\{v', v''\}$. Therefore, we run at most $q(g)$ instances of ListIso on smaller graphs with modified lists.

If $v \notin L(u)$, clearly a list-compatible isomorphism is not possible for this choice of $v \in V'$. If $v \in L(u)$, we put $L'(u') = L'(u'') = \{v', v''\}$. Then there exists a list-compatible isomorphism from G to H, if and only if there exists a list-compatible isomorphism from G' to H'.

The correctness of our algorithm follows from [55]. It remains to argue the complexity. Throughout the algorithm, we produce at most $w(g)$ subgraphs of G and H, for some function w, for which we test list-compatible isomorphisms. Assuming the induction hypothesis, the reduction of graphs to 3-connected graphs can be done in time $\mathcal{O}(\sqrt{n}\ell)$. Case 1 can be solved in time $\mathcal{O}(\ell)$. Case 2 can be solved in time $\mathcal{O}(\sqrt{n}\ell)$. Case 3 can be solved in time $\mathcal{O}(\sqrt{n}\ell)$. $\square$

10 Bounded Treewidth Graphs

In this section, we prove that ListIso can be solved in FPT with respect to the parameter treewidth $\mathrm{tw}(G)$. Unlike in Sections 8 and 7, the difficulty of graph isomorphism on bounded treewidth graphs raises from the fact that tree decomposition is not uniquely determined. We follow the approach of Bodlaender [10] which describes an XP algorithm for GraphIso of bounded treewidth graphs, running in time $n^{\mathcal{O}(\mathrm{tw}(G))}$. Then we show that the FPT-algorithm for GraphIso by Lokshtanov et al. [68] can be modified as well.

Definition 10.1. *A* tree decomposition *of a graph G is a pair $\mathcal{T} = (\{B_i \colon i \in I\}, T = (I, F))$, where T is a rooted tree and $\{B_i \colon i \in I\}$ is a family of subsets of V, such that*

1. *for each $v \in V(G)$ there exists an $i \in I$ such that $v \in B_i$,*
2. *for each $e \in E(G)$ there exists an $i \in I$ such that $e \subseteq B_i$,*
3. *for each $v \in V(G), I_v = \{i \in I \colon v \in B_i\}$ induces a subtree of T.*

We call the elements B_i the nodes*, and the elements of the set F the decomposition edges.*

We define the width of a tree decomposition $\mathcal{T} = (\{B_i \colon i \in I\}, T)$ as $\max_{i \in I} |B_i| - 1$ and the *treewidth* $\mathrm{tw}(G)$ of a graph G as the minimum width of a tree decomposition of the graph G.

Nice Tree Decompositions. It is common to define a *nice tree decomposition* of the graph [59]. We naturally orient the decomposition edges towards the root and for an oriented decomposition edge (B_j, B_i) from B_j to B_i we call B_i the *parent* of B_j and B_j a *child* of B_i. If there is an oriented path from B_j to B_i we say that B_j is a *descendant* of B_i.

We also adjust a tree decomposition such that for each decomposition edge (B_i, B_j) it holds that $\big||B_i| - |B_j|\big| \le 1$ (i.e. it joins nodes that differ in at most one vertex). The in-degree of each node is at most 2 and if the in-degree of the node B_k is 2 then for its children B_i, B_j holds that $B_i = B_j = B_k$ (i.e. they represent the same vertex set).

We classify the nodes of a nice decomposition into four classes—namely *introduce nodes*, *forget nodes*, *join nodes* and *leaf nodes*. We call the node B_i an introduce node of the vertex v, if it has a single child B_j and $B_i \setminus B_j = \{v\}$. We call the node B_i a forget node of the vertex v, if it has a single child B_j and $B_j \setminus B_i = \{v\}$. If the node B_k has two children, we call it a join node (of nodes B_i and B_j). Finally we call a node B_i a leaf node, if it has no child.

Bodlaender's Algorithm. A graph G has treewidth at most k if either $|V(G)| \le k$, or there exists a cut set $U \subseteq V(G)$ such that $|U| \le k$ and each component of $G \setminus U$ together with U has treewidth at most k. The set U corresponds to a bag in some tree decomposition of G. Bodlaender's algorithm [10] enumerates all possible cut sets U of size at most k in G (resp. H), we denote these C_i (resp. D_i). Furthermore, it enumerates all connected components of $G \setminus C_i$ as C_i^j (resp. of $H \setminus D_i$ as D_i^j). We denote by $G[U, W]$ the graph induced by $U \dot\cup W$. The set W is either a connected component or a collection of connected components. We call U the *border set.*

Lemma 10.2 ([1,10]). *A graph $G[U, W]$ with at least k vertices has a treewidth at most k with the border set U if and only if there exists a vertex $v \in W$ such that for each connected component A of $G[W \setminus v]$, there is a k-vertex cut $C_s \subseteq U \cup \{v\}$ such that no vertex in A is adjacent to the (unique) vertex in $(U \cup \{v\}) \setminus C_s$, and $G[C_s, A]$ has treewidth at most k.*

Lemma 10.3. *The problem* ListIso *can be solved in* XP *with respect to the parameter treewidth.*

Proof. We modify the algorithm of Bodlaender [10]. Let $k = \mathrm{tw}(G) = \mathrm{tw}(H)$. We compute the sets C_i, C_i^j for G and the sets $D_{i'}, D_{i'}^{j'}$ for H; there are $n^{\mathcal{O}(k)}$ pairs (C_i, C_i^j). The pair (C_i, C_i^j) is *compatible* if C_i^j is a connected component of $G' \setminus C_i$ for some $G' \subseteq G$ that arises during the recursive definition of treewidth. Let $f \colon C_i \to D_{i'}$ be an isomorphism. We say that $(C_i, C_i^j) \equiv_f (D_{i'}, D_{i'}^{j'})$ if and only if there exists an isomorphism $\varphi \colon C_i \cup C_i^j \to D_{i'} \cup D_{i'}^{j'}$ such that $\varphi|_{C_i} = f$. In other words, φ is a partial isomorphism from G to H. The change for ListIso is that we also require that both f and φ are list-compatible.

The algorithm resolves $(C_i, C_i^j) \equiv_f (D_{i'}, D_{i'}^{j'})$ by the dynamic programming, according to the size of $D_{i'}^{j'}$. If $|C_i^j| = |D_{i'}^{j'}| \le 1$, we can check it trivially in time $k^{\mathcal{O}(k)}$. Otherwise, suppose that $|C_i^j| = |D_{i'}^{j'}| > 1$, and let m be the number of components of C_i^j (and thus $D_{i'}^{j'}$). We test whether $f : C_i \to D_{i'}$ is a list-compatible isomorphism. Let $v \in C_i^j$ be a vertex given by Lemma 10.2 (with $U = C_i$ and $W = C_i^j$) and let C_s be the corresponding extension of v to a cut set. We compute for all $w \in D_{i'}^{j'} \cap \mathfrak{L}(v)$ all connected components B_q. From the dynamic programming, we know for all possible extensions D' of w to a cut set whether $(C_m, A_p) \equiv_{f'} (D', B_q)$ with $f'(x) = f(x)$ for $x \in C_i$ and $f'(v) = w$. Finally, we decide whether there exists a perfect matching in the bipartite graph between (C_m, A_p)'s and (D', B_q)'s where the edges are according to the equivalence. $\square$

Reducing The Number of Possible Bags. Otachi and Schweitzer [78] proposed the idea of pruning the family of potential bags which finally led to an FPT algorithm [68]. A family $\mathcal{B}(G)$, whose definition depends on the graph, is called *isomorphism-invariant* if for an isomorphism $\phi : G \to G'$, we get $\mathcal{B}(G') = \phi(\mathcal{B}(G))$, where $\phi(\mathcal{B}(G))$ denotes the family $\mathcal{B}(G)$ with all the vertices of G replaced by their images under ϕ.

For a graph G, a pair (A, B) with $A \cup B = V$ is called a *separation* if there are no edges between $A \setminus B$ and $B \setminus A$ in G. The order of (A, B) is $|A \cap B|$. For two vertices $u, v \in V(G)$, by $\mu(u, v)$ we denote the minimum order of separation (A, B) with $u \in A \setminus B$ and $v \in B \setminus A$. We say a graph G is *k-complemented* if $\mu_G(u, v) \geq k) \implies uv \in E(G)$ holds for every two vertices $u, v \in V$. We may canonically modify the input graphs G and H ListIso, by adding these additional edges and making them k-complemented.

Theorem 10.4 ([68], Theorem 5.5). *Let k be a positive integer, and let G be a graph on n vertices that is connected and k-complemented. There exists an algorithm that computes in time $2^{\mathcal{O}(k^5 \log k)} \cdot n^3$ an isomorphism-invariant family of bags $\mathcal{B}$ with the following properties:*
1. *$|B| \leq \mathcal{O}(k^4)$ for each $B \in \mathcal{B}$,*
2. *$|\mathcal{B}| \leq 2^{\mathcal{O}(k^5 \log k)} \cdot n^2$,*
3. *Assuming $\mathrm{tw}(G) < k$, the family $\mathcal{B}$ captures some tree decomposition of G that has width $\mathcal{O}(k^4)$.*
4. *The family $\mathcal{B}$ is closed under taking subsets.*

Theorem 10.5. *The problem* ListIso *can be solved in* FPT *time $2^{\mathcal{O}(k^5 \log k)} n^5$ where $k = \mathrm{tw}(G)$.*

Proof. We use the algorithm of Lemma 10.3, where C_i's and D_i's are from the collection $\mathcal{B}$ of Theorem 10.4. The total number of pairs (C_i, C_i^j) and $(D_{i'}, D_{i'}^{j'})$ is bounded by $2^{\mathcal{O}(k^5 \log k)} n^3$ [68, p. 20]. The dynamic programming in [68, Theorem 6.2] is done according to the potential function $\Phi(D_{i'}, D_{i'}^{j'}) = 2|D_{i'}^{j'}| + |D_{i'}|$. We use nice tree decompositions, so in each step, the dynamic programming either introduces a new node into the bag $D_{i'}$, or moves a node from the bag $D_{i'}$ to $D_{i'}^{j'}$, or joins several pairs with the same bag $D_{i'}$. In all these operations, we check existence of a list-compatible isomorphism, using dynamic programming, exactly as in Lemma 10.3. ☐

11 Conclusions

We conclude this paper with description of related results and open problems.

Forbidden Images. We note that Lubiw [69] used a different definition of ListIso: for every vertex $u \in V(G)$, we are given a *list of forbidden images* $\mathcal{F}(u) \subseteq V(H)$ and we want to find an isomorphism $\pi : G \to H$ such that $\pi(u) \notin \mathcal{F}(u)$. The advantage of forbidden lists is that we can express GraphIso in space $\mathcal{O}(n+m)$, but the input for ListIso is of size $\mathcal{O}(n^2)$. On the other hand, we consider lists of allowed images more natural (for instance, list coloring is defined similarly) and also such a definition appears naturally in [31]. Both statements are clearly polynomially equivalent, and the main focus of our paper is to distinguish between tractable and intractable cases for ListIso.

Group Reformulation. Luks [71] described the following group problem which generalizes computing automorphism groups of graphs. Let Ω be a ground set and let Γ be a group acting on Ω. Further, let Ω be colored. We want to compute the subgroup of Γ which is color preserving. When Γ is the symmetric group acting on all pairs of vertices $V(G)$ which are colored by two colors (corresponding to edges and non-edges in G), then the computed subgroup is $\mathrm{Aut}(G)$. To generalize graph isomorphism in this language, we have two colorings and we want to find a color-preserving permutation $g \in \Gamma$. We note that Babai [5] calls these generalizations *string automorphism/isomorphism problems.*

A similar generalization of ListIso was suggested to us by Ponomarenko. We are given a group Γ acting on a ground set Ω and for every $x \in \Omega$, we have a list $\mathfrak{L}(x) \subseteq \Omega$. We ask whether there exists a permutation $g \in \Gamma$ such that $g(x) \in \mathfrak{L}(x)$ for every $x \in \Omega$. We obtain ListIso either similarly as above, or when $\Omega = V(G)$ and $\Gamma = \mathrm{Aut}(G)$.

We may interpret our results for ListIso using this group reformulation. The robust combinatorial algorithms work because the groups Γ are highly restricted. In particular, for trees, Jordan [54] proved that $\mathrm{Aut}(G)$ is formed by a series of direct products and wreath products with symmetric groups, to it has a tree structure. Therefore, the algorithm of Theorem 6.1 solves ListIso on $\mathrm{Aut}(G)$ by a bottom-up dynamic algorithm. Similar characterizations were recently proved for interval, permutation and circle graphs [57,58] and for planar graphs [56], and are used in the algorithms of Theorems 8.3, 7.1, 7.2, and 7.3. For graphs of bounded genus or bounded treewidth, no such detailed description of the automorphism groups is yet known, but they are likely restricted as well. On the other hand, for cubic graphs, the automorphism groups $\mathrm{Aut}(G)$ may be arbitrary, so this approach fails, and actually ListIso is NP-complete by Theorem 5.3.

To attack ListIso from the point of this group reformulation, instead of different graph classes, we may study it for different combinations of Γ and the lists $\mathfrak{L}$. First, for which groups Γ, it can be solved efficiently for all possible lists $\mathfrak{L}$? Second, for which lists $\mathfrak{L}$, the problem can be solved efficiently? We did not try to attack the problem much in the second direction, aside Lemma 3.2 and Theorem 5.3. For instance, when $\mathfrak{L}$ is a partitioning of Ω, the problem is easy since we get the usual color-preserving isomorphism problem.

k-dimensional Weisfieler-Leman refinement (k-WL). The classical 2-WL [93,94] colors vertices of a graph and it initiates with different colors for each degree. In each step, it takes vertices of one color class and partitions them by different numbers of neighbors of other color classes. It stops when no partitioning longer occurs. Its generalization k-WL [3,53] colors and partitions $(k-1)$-tuples according to their adjacencies.

Certainly, when G and H are isomorphic graphs, they are partitioned and colored the same. So when $G \not\cong H$, k-WL distinguishes G and H, for a suitable value of k. If we prove for a graph class that k is small enough, we obtain a combinatorial algorithm for GraphIso. For instance, Grohe [41] proves that for every graph X, there exists a value k such that two X-minor free graphs are either isomorphic, or distinguished by k-WL. This does cannot be modified to solve ListIso since k-WL applied on G only estimates the orbits of $\mathrm{Aut}(G)$. When $G \cong H$, and we may test this, assuming that GraphIso can be decided efficiently for G and H, we obtain two identical partitions. They may be used to reduce sizes of the lists, but we still end up with the question whether there exists a list-compatible isomorphism. In Section 5, we show that it is NP-complete to decide ListIso even when sizes of all lists are bounded by 3.

Excluded Minors. Another major open problem for ListIso is its complexity for graphs with excluded minors. As described in Introduction, the original polynomial-time algorithm of Ponomarenko [79] for GraphIso is based heavily on group theory, and his technique probably cannot be modified to solve ListIso. But it seems doubtful that the problem will be NP-complete, since new combinatorial structural and algorithmic results may be applied.

Robertson and Seymour [81] proved that every graph G with an excluded minor can be decomposed into pieces which are "almost embeddable" to a surface of genus g, where g depends on the minor. The recent book of Grohe [41] describes the seminal idea of automorphism-invariant treelike decompositions. A *treelike decomposition* generalizes a classical tree decomposition by replacing a tree of bags by a directed acyclic graph of bags. Unlike tree decompositions, a treelike decomposition D of G can be constructed with two additional properties. Firstly, it is *automorphism-invariant*, meaning that every automorphism of G induces an automorphism of D. Secondly, it is *canonical*, meaning that for two isomorphic graphs G and G', isomorphic treelike decompositions D and D' are constructed.

The main structural result of Grohe [41] is that every graph G with an excluded minor has a canonical automorphism-invariant treelike decomposition for which the graphs induced by bags called *torsos* are "almost embeddable" to a surface of genus g, where g depends on the minor. Therefore, to solve ListIso on graphs with excluded minors, we need to prove the following:

1. We need to show that ListIso can be solved on almost embeddable graphs in polynomial time. We may use the results of Theorems 9.1 and 10.5 to do so.
2. We need to prove Lifting Lemma for ListIso, stating the following: If we can compute a canonical automorphism-invariant treelike decomposition of a class $\mathcal{C}$ in polynomial time and we can solve ListIso for its torsos $\mathcal{A}$ in polynomial time, then we can solve ListIso for $\mathcal{C}$ in polynomial time as well. Here, we might modify the algorithm for lifting of canonization by Grohe [41].

We note that it is quite difficult to understand and describe everything in detail. The book of Grohe [41] is very extensive (almost 500 pages) and described in the language of graph logics. Unfortunately, no purely combinatorial description of the results is available, and we believe that such a description of a combinatorial algorithm for solving graph isomorphism of graphs with excluded minors would be desirable. A combinatorial description of treelike decompositions is described by Grohe and Marx [42], but an algorithm for the graph isomorphism problem of graphs with excluded minors is only used as a black box.

Bounded Rankwidth. Rankwidth generalizes treewidth in the way that bounded treewidth implies bounded rankwidth, but not the opposite. Very recently, the first XP algorithm for graph isomorphism of graphs of bounded rankwidth was described [43]. The approach is by computing automorphism-invariant tree decomposition (which probably can be modified for ListIso), but then further group theory is applied to test whether the decompositions are isomorphic. It is a very interesting question whether group theory can be avoided and the problem can be solved in a purely combinatoric way. Therefore, determining the complexity of ListIso for graphs of bounded rankwidth is one of major open problems and it might give an insight into this question as well.

Also, rankwidth is closely related to cliquewidth: when one parameter is bounded, the other is bounded as well. The graphs of cliquewidth at most 2 are called *cographs* and can be represented by *cotrees*. Their isomorphism can therefore be reduced to isomorphism of cotrees and solved in polynomial time [18], and this approach probably can be modified for ListIso. Very recently, a combinatorial polynomial-time algorithm for graph isomorphism of graphs of cliquewidth at most 3 was described [22] which probably can be modified for ListIso as well.

Bounded Eigenvalue Multiplicity. The polynomial-time algorithms for GraphIso of graphs of bounded eigenvalue multiplicity [6,29] are heavily based on group theory. Actually, already the case of multiplicity one is non-trivial. It seems unlikely that these results can be modified to solve ListIso, but constructing an NP-hardness reduction with bounded eigenvalue multiplicity seems non-trivial.

Forbidden Subgraphs, Induced Subgraphs and Induced Minors. There are several papers dealing with GraphIso for classes of graphs with excluded subgraphs, induced subgraphs and induced minors, and again the question is which results can be modified to solve ListIso. Otachi and Schweitzer [77] prove a dichotomy for excluded subgraphs. The GI-complete cases can be modified by Theorem 4.1, but the polynomial cases follow from [42] which probably cannot be modified. More complicated characterizations are known for forbidden induced subgraphs [12,64,84]. Belmonte et al. [7] describe dichotomy for forbidden induced minors.

Logspace Results. For some graph classes, GraphIso is known to be solvable in LogSpace. It is a natural question to ask whether these results can be modified to solve ListIso. For instance, graph isomorphism of trees [67] can be solved in LogSpace, with a similar bottom-up procedure as in the proof of Theorem 6.1. The celebrated result of Reingold [80], stating that undirected reachability can be solved in LogSpace, allowed many other graph algorithms to be solved in LogSpace. In particular, GraphIso is known to be solvable in LogSpace for planar graphs [23,24], k-trees [60], interval graphs [61], and bounded treewidth graphs [28].

Acknowledgment. We want to thank to annonymous reviewers for pointing out the papers [16,17] and for a suggestion to simplify the reduction of Section 5 which reduced sizes of color classes from 16 to 8.

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, April 1987.
2. B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.
3. L. Babai. On the isomorphism problem. 1977.
4. L. Babai. Automorphism groups, isomorphism, reconstruction. In *Handbook of combinatorics (vol. 2)*, pages 1447–1540. MIT Press, 1996.
5. L. Babai. Graph isomorphism in quasipolynomial time. In *STOC*, 2016.

6. L. Babai, D.Y. Grigoryev, and D.M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 310–324. ACM, 1982.
7. R. Belmonte, Y. Otachi, and P. Schweitzer. Induced minor free graphs: Isomorphism and clique-width. *CoRR*, abs:1605.08540, 2016.
8. D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17(1):53–76, 1988.
9. M. Biró, M. Hujter, and Zs. Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992.
10. H. L Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.
11. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.
12. K.S. Booth and C.J. Colbourn. Problems polynomially equivalent to graph isomorphism. *Technical Report CS-77-04, Computer Science Department, University of Waterloo*, 1979.
13. J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
14. R. Chitnis, L. Egri, and D. Marx. List h-coloring a graph by removing few vertices. In *Algorithms – ESA 2013: 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, LNCS, pages 313–324, 2013.
15. C. J. Colbourn. On testing isomorphism of permutation graphs. *Networks*, 11(1):13–21, 1981.
16. C. J. Colbourn and M. J. Colbourn. Isomorphism problems involving self-complementary graphs and tournaments. In *Proceedings of the Eighth Manitoba Conference on Numerical Mathematics and Computing (Univ. Manitoba, Winnipeg, Man., 1978), Congr. Numer*, volume 22, pages 153–164, 1978.
17. D. G. Corneil and D. G. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM Journal on Computing*, 9(2):281–297, 1980.
18. D.G. Corneil, H. Lerchs, and L.S. Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981.
19. W.H. Cuningham and J. Edmonds. A combinatorial decomposition theory. *Canad. J. Math.*, 32:734–765, 1980.
20. W.H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3:214–228, 1982.
21. V. Dalmau, L. Egri, P. Hell, B. Larose, and A. Rafiey. Descriptive complexity of list h-coloring problems in logspace: A refined dichotomy. In *Logic in Computer Science (LICS), 2015 30th Annual ACM/IEEE Symposium on*, pages 487–498, 2015.
22. B. Das, M. K. Enduri, and I. V. Reddy. Polynomial-time algorithm for isomorphism of graphs with clique-width at most three. *CoRR*, 1506.01695, 2015.
23. S. Datta, N. Limaye, and P. Nimbhorkar. 3-connected planar graph isomorphism is in log-space. In *FSTTCS*, 2008.
24. S. Datta, N. Limaye, P. Nimbhorkar, T. Thierauf, and F. Wagner. Planar graph isomorphism is in logspace. In *Proc. 24th IEEE CCC*, pages 203–214, 2009.
25. G. Di Battista and R. Tamassia. Incremental planarity testing. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 436–441. IEEE, 1989.
26. G. Di Battista and R. Tamassia. On-line graph algorithms with spqr-trees. In *International Colloquium on Automata, Languages, and Programming*, pages 598–611. Springer, 1990.
27. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.
28. M. Elberfeld and P. Schweitzer. Canonizing Graphs of Bounded Tree Width in Logspace. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:14, 2016.
29. S. Evdokimov and I.N. Ponomarenko. Isomorphism of coloured graphs with slowly increasing multiplicity of jordan blocks. *Combinatorica*, 19(3):321–333, 1999.

30. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691, 1976.
31. J. Fiala, P. Klavík, J. Kratochvíl, and R. Nedela. Algorithmic aspects of regular graph covers with applications to planar graphs. In *ICALP 2014*, volume 8572 of *LNCS*, pages 489–501, 2014.
32. J. Fiala, P. Klavík, J. Kratochvíl, and R. Nedela. 3-connected reduction for regular graph covers. *CoRR*, abs/1503.06556, 2016.
33. J. Fiala, P. Klavík, J. Kratochvíl, and R. Nedela. Algorithmic aspects of regular graph covers. *CoRR*, abs/1609.03013, 2016.
34. L.S. Filotti and J.N. Mayer. A polynomial time algorithm for determining the isomorphism of graphs of fixed genus. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, STOC '80, pages 236–243, 1980.
35. L. Folwarczný and D. Knop. IV-matching is strongly NP-hard. *CoRR*, abs/1506.08388, 2015.
36. 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.
37. M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *FOCS*, pages 36–41. IEEE, 1980.
38. T. Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1):25–66, 1967.
39. E. Gioan, C. Paul, M. Tedder, and D. Corneil. Practical and efficient circle graph recognition. *Algorithmica*, pages 1–30, 2013.
40. O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, volume 86, pages 174–187, 1986.
41. M. Grohe. Descriptive complexity, canonisation, and definable graph structure theory. *Manuscript*, 2012.
42. M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 173–192, 2012.
43. M. Grohe and P. Schweitzer. Isomorphism testing for graphs of bounded rank width. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1010–1029, 2015.
44. C. Gutwenger and P. Mutzel. A linear time implementation of spqr-trees. In *International Symposium on Graph Drawing*, pages 77–90. Springer, 2000.
45. F. Harary. *Graph theory*. Addison-Wesley, Reading, MA, 1969.
46. P. Hell and J. Nešetril. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2004.
47. J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In *Complexity of computer computations*, pages 131–152. Springer, 1972.
48. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
49. J. E. Hopcroft and J. Wong. Linear time algorithm for isomorphism of planar graphs. In *STOC*, pages 172–184. ACM, 1974.
50. J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
51. J.E. Hopcroft and R.E. Tarjan. A V^2 algorithm for determining isomorphism of planar graphs. *Information Processing Letters*, 1(1):32–34, 1971.
52. W. L. Hsu. $\mathcal{O}(M \cdot N)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal on Computing*, 24(3):411–439, 1995.
53. N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pages 59–81. Springer, 1990.
54. C. Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

55. K. Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. *CoRR*, abs/1511.02460, 2015.
56. P. Klavík, R. Nedela, and P. Zeman. Constructive approach to automorphism groups of planar graphs. *CoRR*, abs/1506.06488, 2016.
57. P. Klavík and P. Zeman. Automorphism groups of geometrically represented graphs. In *STACS 2015*, volume 30 of *LIPIcs*, pages 540–553, 2015.
58. P. Klavík and P. Zeman. Automorphism groups of geometrically represented graphs. *CoRR*, abs/1407.2136, 2015.
59. T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
60. J. Köbler and S. Kuhnert. The isomorphism problem of k-trees is complete for logspace. In *Proc. 34th MFCS*, pages 537–448, 2009.
61. Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representations in logspace. *SIAM Journal on Computing*, 40(5):1292–1315, 2011.
62. N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.
63. J. Kratochvíl and Zs. Tuza. Algorithmic complexity of list colorings. *Discrete Applied Mathematics*, 50(3):297–302, 1994.
64. S. Kratsch and P. Schweitzer. Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. In *WG 2012*, volume 7551 of *Lecture Notes in Computer Science*, pages 34–45, 2012.
65. F. Lalonde. Le problème d'étoiles pour graphes est np-complet. *Discrete Mathematics*, 33(3):271–280, 1981.
66. D. Lichtenstein. Isomorphism for graphs embeddable on the projective plane. In *ACM Symposium on Theory of Computing*, STOC '80, pages 218–224, 1980.
67. S. Lindell. A logspace algorithm for tree canonization. In *Proc. 24th ACM STOC*, pages 400–404, 1992.
68. D. Lokshtanov, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. In *FOCS*, pages 186–195, 2014.
69. A. Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981.
70. G. S. Lueker and K. S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM)*, 26(2):183–195, 1979.
71. E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.
72. S. Mac Lane. A structural characterization of planar combinatorial graphs. *Duke Mathematical Journal*, 3(3):460–472, 1937.
73. R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.
74. R.M. McConnell and J.P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1–3):189–241, 1999.
75. G. Miller. Isomorphism testing for graphs of bounded genus. In *ACM Symposium on Theory of Computing*, STOC '80, pages 225–235, 1980.
76. G.L. Miller. Isomorphism testing and canonical forms for k-contractable graphs (a generalization of bounded valence and bounded genus). In *International Conference on Fundamentals of Computation Theory*, pages 310–327. Springer, 1983.
77. Y. Otachi and P. Schweitzer. Isomorphism on subgraph-closed graph classes: A complexity dichotomy and intermediate graph classes. In *International Symposium on Algorithms and Computation*, pages 111–118. Springer, 2013.
78. Y. Otachi and P. Schweitzer. Reduction techniques for graph isomorphism in the context of width parameters. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 368–379, 2014.

79. I.N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, 1991.
80. O. Reingold. Undirected connectivity in logspace. *Journ. of ACM*, 55(4), 2008.
81. N. Robertson and P. Seymour. Graph minors xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 77:1–27, 1999.
82. T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226. ACM, 1978.
83. U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
84. P. Schweitzer. Towards an isomorphism dichotomy for hereditary graph classes. In *STACS 2016*, volume 30 of *LIPIcs*, 2015.
85. J. Spinrad. On comparability and permutation graphs. *SIAM J. Comput.*, 14(3):658–670, 1985.
86. A. Takaoka. Graph isomorphism completeness for trapezoid graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 98(8):1838–1840, 2015.
87. B.A. Trakhtenbrot. Towards a theory of non-repeating contact schemes. *Trudi Mat. Inst. Akad. Nauk SSSR*, 51:226–269, 1958.
88. W. T. Tutte. *Connectivity in graphs*, volume 15. University of Toronto Press, 1966.
89. R. Uehara. Tractabilities and intractabilities on geometric intersection graphs. *Algorithms*, 6(1):60–83, 2013.
90. 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.
91. V. G. Vizing. Vertex colorings with given colors. *Metody Diskret. Analiz.*, 29:3–10, 1976.
92. T.R.S. Walsh. Counting unlabeled three-connected and homeomorphically irreducible two-connected graphs. *J. Combin. Theory B*, 32:12–32, 1982.
93. B. Weisfeiler and A.A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 9:12–16, 1968.
94. B. Weisfeiler (ed.). *On Construction and Identification of Graphs*, volume 558 of *Lect. Notes in Math.* Springer, 1976.
95. H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.