Recent Advances on the Graph Isomorphism Problem

Martin Grohe and Daniel Neuen

Abstract

We give an overview of recent advances on the graph isomorphism problem. Our main focus will be on Babai's quasi-polynomial time isomorphism test and subsequent developments that led to the design of isomorphism algorithms with a quasi-polynomial parameterized running time of the form $n^{\mathrm{polylog}(k)}$, where k is a graph parameter such as the maximum degree. A second focus will be the combinatorial Weisfeiler-Leman algorithm.

1 Introduction

Determining the computational complexity of the Graph Isomorphism Problem (GI) is regarded a major open problem in computer science. Already stated as an open problem in Karp's seminal paper on the NP-completeness of combinatorial problems in 1972 [51], GI remains one of the few natural problems in NP that is neither known to be NP-complete nor known to be polynomial-time solvable. In a breakthrough five years ago, Babai [6] proved that GI is solvable in *quasi-polynomial time* $n^{\mathrm{polylog}(n)} := n^{(\log n)^{\mathcal{O}(1)}}$, where n denotes the number of vertices of the input graphs. This was the first improvement on the worst-case running time of a general graph isomorphism algorithm over the $2^{\mathcal{O}(\sqrt{n \log n})}$ bound established by Luks in 1983 (see [10]).

Research on GI as a computational problem can be traced back to work on chemical information systems in the 1950s (e.g. [77]). The first papers in the computer science literature, proposing various heuristic approaches, appeared in the 1960s (e.g. [85]). An early breakthrough was Hopcroft and Tarjan's $\mathcal{O}(n \log n)$ isomorphism test for planar graphs [48]. Isomorphism algorithms for specific graph classes have been an active research strand ever since (e.g. [31, 39, 57, 61, 66, 76]).

Significant progress on GI was made in the early 1980s when Babai and Luks introduced non-trivial group-theoretic methods to the field [4, 11, 61]. In particular, Luks [61] introduced a general divide-and-conquer framework based on the structure of the permutation groups involved. It is the basis for most subsequent developments. Luks used it to design an isomorphism test running in time $n^{\mathcal{O}(d)}$, where d is the maximum degree of the input graphs. The divide-and-conquer framework is also the foundation of Babai's quasi-polynomial time algorithm. Primitive permutation groups constitute the difficult bottleneck of Luks's strategy. Even though we know a lot about the structure of primitive permutation groups (going back to [20] and the classification of finite simple groups), for a long a time it was not clear how we can exploit this structure algorithmically. It required several groundbreaking ideas by Babai, both on the combinatorial and the group-theoretic side, to obtain a quasi-polynomial time algorithm. Starting from Luks's divide-and-conquer framework, we will describe the group-theoretic approach and (some of) Babai's new techniques in Section 4.

In subsequent work, jointly with our close collaborators Pascal Schweitzer and Daniel Wiebking, we have designed a number of *quasi-polynomial parameterized*

algorithms where we confine the poly-logarithmic factor in the exponent to some parameter that can be much smaller than the number n of vertices of the input graph. The first result [41] in this direction was an isomorphism test running in time $n^{\text{polylog}(d)}$ which improves both Luks's and Babai's results (recall that d denotes the maximum degree of the input graph). After similar quasi-polynomial algorithms parameterized by tree width [87] and genus [71], the work culminated in an isomorphism test running in time $n^{\text{polylog}(k)}$ for graphs excluding some k-vertex graph as a minor [43]. In Section 5, we will present these results in a unified framework that is based on a new combinatorial closure operator and the notion of *t-CR-bounded* graphs, introduced by the second author in [71] (some ideas can be traced back as far as [67, 75]).

The combinatorial algorithm underlying this approach is a very simple partition refinement algorithm known as the Color Refinement algorithm (first proposed in [68]), which is also an important subroutine of all practical graph isomorphism algorithms. The Color Refinement algorithm can be viewed as the 1-dimensional version of a powerful combinatorial algorithm known as the Weisfeiler-Leman algorithm (going back to [86]). The strength of the Weisfeiler-Leman algorithm and its multiple, sometimes surprising connections with other areas, have been a recently very active research direction that we survey in Section 3.

It is not the purpose of this paper to cover all aspects of GI. In fact, there are many research strands that we completely ignore, most notably the work on practical isomorphism algorithms. Instead we want to give the reader at least an impression of the recent technical developments that were initiated by Babai's breakthrough. For a broader and much less technical survey, we refer the reader to [45].

2 Preliminaries

Let us review some basic notation. By $\mathbb{N}$, we denote the natural numbers (including 0), and for every positive $k \in \mathbb{N}$ we let $[k] := \{1, \ldots, k\}$. Graphs are always finite and, unless explicitly stated otherwise, undirected. We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively. For (undirected) edges, we write vw instead of $\{v, w\}$. The *order* of a graph G is $|G| := |V(G)|$.

A *subgraph* of a graph G is a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a set $W \subseteq V(G)$, the *induced subgraph* of G with vertex set W is the graph $G[W]$ with $V(G[W]) = W$ and $E(G[W]) := \{vw \in E(G) \mid v, w \in W\}$. Moreover, $G - W := G[V(G) \setminus W]$. The set of *neighbors* of a vertex v is $N_G(v) := \{w \mid vw \in E(G)\}$, and the *degree* of v is $\deg_G(v) := |N_G(v)|$. If G is clear from the context, we omit the subscript $_G$. The *maximum degree* of G is $\Delta(G) := \max\{\deg(v) \mid v \in V(G)\}$.

A *homomorphism* from a graph G to a graph H is a mapping $h : V(G) \to V(H)$ that preserves adjacencies, that is, if $vw \in V(G)$ then $h(v)h(w) \in V(H)$. An isomorphism from G to H is a bijective mapping $\varphi : V(G) \to V(H)$ that preserves adjacencies and non-adjacencies, that is, $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$. We write $\varphi : G \cong H$ to indicate that φ is an isomorphism from G to H.

Sometimes, it will be convenient to work with *vertex- and arc-colored graphs*, or just *colored graphs*, $G = (V(G), E(G), \chi_V^G, \chi_E^G)$, where $(V(G), E(G))$ is a graph, $\chi_V^G : V(G) \to C_V$ is a vertex coloring, and $\chi_E^G : \{(v, w) \mid vw \in E(G)\} \to C_E$ an arc coloring. We speak of an *arc coloring* rather than edge coloring, because we color or-
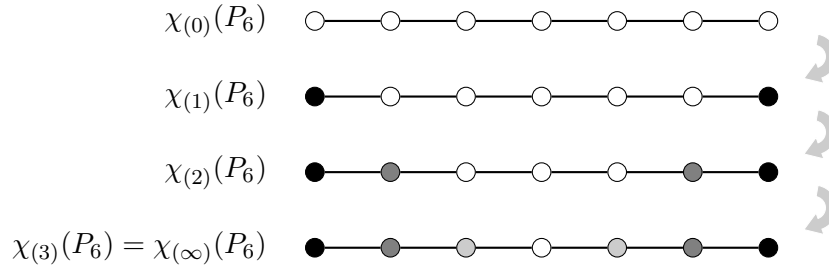
$$\chi_{(0)}(P_6)$$
$$\chi_{(1)}(P_6)$$
$$\chi_{(2)}(P_6)$$
$$\chi_{(3)}(P_6) = \chi_{(\infty)}(P_6)$$

Figure 1: The iterations of Color Refinement on a path P_6 of length 6.

dered pairs (v, w) and not undirected edges vw. Homomorphisms and isomorphisms of colored graphs must respect colors.

3   The Weisfeiler-Leman Algorithm

3.1   The Color Refinement Algorithm

One of the simplest combinatorial procedures to tackle GI is the *Color Refinement algorithm*. The basic idea is to label vertices of the graph with their iterated degree sequence. More precisely, an initially uniform coloring is repeatedly refined by counting, for each color, the number of neighbors of that color.

Let $\chi_1, \chi_2 \colon V(G) \to C$ be colorings of vertices of a graph G, where C is some finite set of colors. The coloring χ_1 *refines* χ_2, denoted $\chi_1 \preceq \chi_2$, if $\chi_1(v) = \chi_1(w)$ implies $\chi_2(v) = \chi_2(w)$ for all $v, w \in V(G)$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

The Color Refinement algorithm takes as input a vertex- and arc-colored graph (G, χ_V, χ_E) and outputs an isomorphism-invariant coloring of the vertices. Initially, the algorithm sets $\chi_{(0)}(G) \coloneqq \chi_V$. Then the coloring is iteratively refined as follows. For $i > 0$ we let $\chi_{(i)}(G)(v) \coloneqq \big(\chi_{(i-1)}(G)(v), \mathcal{M}_i(v)\big)$, where

$$\mathcal{M}_i(v) \coloneqq \left\{\!\!\left\{ \big(\chi_{(i-1)}(G)(w), \chi_E(v, w), \chi_E(w, v)\big) \ \Big|\ w \in N_G(v) \right\}\!\!\right\}$$

is the multiset of colors for the neighbors computed in the previous round.

In each round, the algorithm computes a coloring that is finer than the one computed in the previous round, i.e., $\chi_{(i)}(G) \preceq \chi_{(i-1)}(G)$. At some point this procedure stabilizes, meaning the coloring does not become strictly finer anymore. In other words, there is an $i_\infty < n$ such that $\chi_{(i_\infty)}(G) \equiv \chi_{(i_\infty+1)}(G)$. We call $\chi_{(i_\infty)}(G)$ the *stable coloring* and denote it by $\chi_{(\infty)}(G)$.

Example 3.1 The sequence of colorings for a path of length 6 is shown in Figure 1.

The Color Refinement algorithm is very efficient, and it is used as an important subroutine in all practical isomorphism tools [22, 50, 59, 64, 65]. The stable coloring of a graph can be computed in time $\mathcal{O}((n + m) \log n)$, where n denotes the number

of vertices and m the number of edges of the input graph [21] (also see [74])[1]. For a natural class of partitioning algorithms, this is best-possible [13].

3.2 Higher Dimensions

The *k-dimensional Weisfeiler-Leman algorithm (k-WL)* is a natural generalization of the Color Refinement algorithm where, instead of vertices, we color k-tuples of vertices of a graph. The 2-dimensional version, also known as the *classical* Weisfeiler-Leman algorithm, has been introduced by Weisfeiler and Leman [86], and the k-dimensional generalization goes back to Babai and Mathon (see [19]).

To introduce k-WL, we need the notion of the *atomic type* $\mathrm{atp}(G, \bar{v})$ of a tuple $\bar{v} = (v_1, \dots, v_\ell)$ of vertices in a graph G. It describes the isomorphism type of the labeled induced subgraph $G[\{v_1, \dots, v_\ell\}]$, that is, for graphs G, H and tuples $\bar{v} = (v_1, \dots, v_\ell) \in V(G)^\ell, \bar{w} = (w_1, \dots, w_\ell) \in V(H)^\ell$ we have $\mathrm{atp}(G, \bar{v}) = \mathrm{atp}(H, \bar{w})$ if and only if the mapping $v_i \mapsto w_i$ is an isomorphism from $G[v_1, \dots, v_k]$ to $H[w_1, \dots, w_k]$. Formally, we can view $\mathrm{atp}(G, v_1, \dots, v_k)$ as a pair of Boolean $(k \times k)$-matrices, one describing equalities between the v_i, and one describing adjacencies. If G is a colored graph, we need one matrix for each arc color and one diagonal matrix for each vertex color.

Then k-WL computes a sequence of colorings $\chi^k_{(i)}(G)$ of $V(G)^k$ as follows. Initially, each tuple is colored by its atomic type, that is, $\chi^k_{(0)}(G)(\bar{v}) \coloneqq \mathrm{atp}(G, \bar{v})$. For $i > 0$, we define

$$\chi^k_{(i)}(G)(\bar{v}) \coloneqq \left(\chi^k_{(i-1)}(G)(\bar{v}), \mathcal{M}^k_i(\bar{v}) \right),$$

where

$$\mathcal{M}^k_i(\bar{v}) \coloneqq \left\{\!\!\left\{ \left(\mathrm{atp}(G, \bar{v}v), \chi^k_{(i-1)}(G)(\bar{v}[v/1]), \dots, \chi^k_{(i-1)}(G)(\bar{v}[v/k]) \right) \;\middle|\; v \in V(G) \right\}\!\!\right\}.$$

Here, for $\bar{v} = (v_1, \dots, v_k) \in V(G)$ and $v \in V(G)$, by $\bar{v}v$ we denote the $(k+1)$-tuple $(v_1, \dots, v_k, v)$ and, for $i \in [k]$, by $\bar{v}[v/i]$ the k-tuple $(v_1, \dots, v_{i-1}, v, v_{i+1}, \dots, v_k)$. Then there is an $i_\infty < n^k$ such that $\chi^k_{(i_\infty)}(G) \equiv \chi^k_{(i_\infty+1)}(G)$, and we call $\chi^k_{(\infty)}(G) \coloneqq \chi^k_{(i_\infty)}(G)$ the *k-stable coloring*. The k-stable coloring of an n-vertex graph can be computed in time $\mathcal{O}(n^k \log n)$ [49].

It is easy to see that 1-WL is essentially just the Color Refinement algorithm, that is, for all graphs G, H, vertices $v \in V(G), w \in V(H)$, and $i \geq 0$ it holds that

$$\chi_{(i)}(G)(v) = \chi_{(i)}(H)(w) \iff \chi^1_{(i)}(G)(v) = \chi^1_{(i)}(H)(w). \tag{3.1}$$

In the following, we drop the distinction between the Color Refinement algorithm and 1-WL.

Example 3.2 The sequence of colorings of 2-WL on a cycle C_7 of length 7 is shown in Figure 2. In this example, the coloring is symmetric, that is, $\chi^2_{(i)}(C_7)(v, v') = \chi^2_{(i)}(C_7)(v', v)$ for all $i \geq 0$ and $v, v' \in V(C_7)$. This allows us to show the colors as undirected edges.

---

[1]To be precise, the algorithms compute the partition of the vertex set corresponding to the stable coloring, not the actual colors viewed as multisets.

$$\chi^2_{(0)}(C_7) \qquad\qquad \chi^2_{(1)}(C_7) = \chi^2_{(\infty)}(C_7)$$
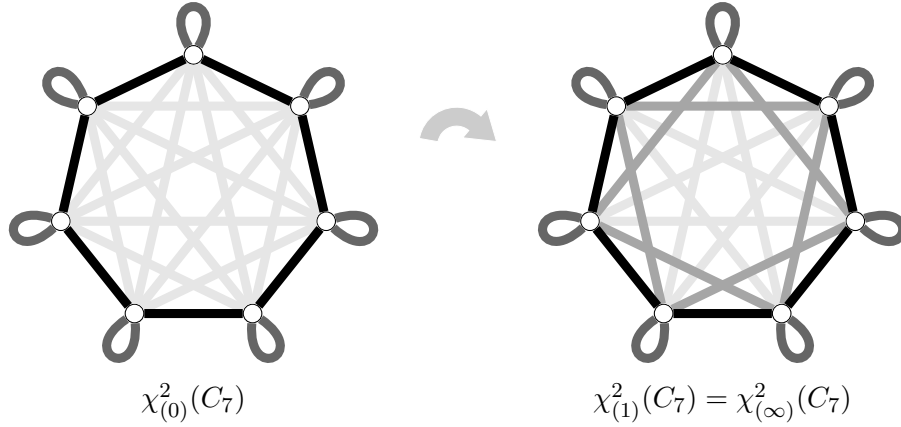
Figure 2: The iterations of 2-WL on a cycle C_7

3.3 The Power of WL and the WL Dimension

To use the Weisfeiler-Leman algorithm as an isomorphism test, we can compare the color patterns of two given graphs. We say that k-WL *distinguishes* graphs G and H if there is a color $c \in \mathrm{rg}(\chi^k_{(\infty)}(G)) \cup \mathrm{rg}(\chi^k_{(\infty)}(H))$ such that

$$\left|\left\{\bar{v} \in V(G)^k \mid \chi^k_{(\infty)}(G)(\bar{v}) = c\right\}\right| \neq \left|\left\{\bar{w} \in V(H)^k \mid \chi^k_{(\infty)}(H)(\bar{w}) = c\right\}\right|.$$

Note that k-WL is an *incomplete* isomorphism test: if it distinguishes two graphs, we know that they are non-isomorphic, but as we will see, there are non-isomorphic graphs that k-WL does not distinguish. Let us call two graphs that are not distinguished by the k-WL k-*indistinguishable*.

Example 3.3 Any two d-regular graphs are 1-indistinguishable. The simplest example are a cycle of length 6 and the disjoint union of two triangles.

Similarly, any two strongly regular graphs with the same parameters are 2-indistinguishable. Figure 3 shows two non-isomorphic strongly regular graphs with parameters $(16, 6, 2, 2)$, which means that they have 16 vertices, each vertex has 6 neighbors, each pair of adjacent vertices has two common neighbors, and each pair of non-adjacent vertices has 2 common neighbors. The first of the graphs is the line graph of the complete bipartite graph $K_{4,4}$. The second graph is the Shrikhande graph, which can be viewed as the undirected graph underlying the Cayley graph for the group $\mathbb{Z}_4 \times \mathbb{Z}_4$ with generators $(0,1), (1,0), (1,1)$. To see that they are non-isomorphic, note that the neighborhood of each vertex in the Shrikhande graph is a cycle of length 6, and the neighborhood of the each vertex in the line graph of $K_{4,4}$ is the disjoint union of two triangles.

It is not easy to find two non-isomorphic graphs that are k-indistinguishable for some $k \geq 3$. In a seminal paper, Cai, Fürer, and Immerman constructed such graphs for each k.

Theorem 3.4 (Cai, Fürer and Immerman [19]) *For every $k \geq 1$ there exist non-isomorphic 3-regular graphs G_k, H_k of order $|G_k| = |H_k| = \mathcal{O}(k)$ that are k-indistinguishable.*
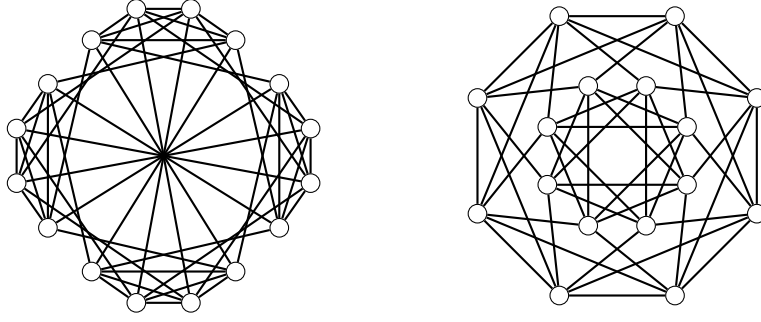
Figure 3: Two non-isomorphic strongly regular graphs with parameters $(16, 6, 2, 2)$: the line graph of $K_{4,4}$ (left) and the Shrikhande Graph (right).

The Weisfeiler-Leman algorithm colors tuples of vertices based on the isomorphism type of these tuples and the overlap with other tuples. To distinguish the so called *CFI graphs* G_k, H_k of Theorem 3.4, we need to color tuples of length linear in the order of the graphs. As the number of such tuples is $\Omega(k^k)$, this is prohibitive both in terms of running time and memory space. However, a closer analysis reveals that it suffices to color only some long tuples, to be precise, an isomorphism-invariant set of tuples of size polynomial in k (this follows from an argument due to Dawar, Richerby and Rossman [24]). In [46], the first author jointly with Schweitzer and Wiebking proposed an algorithmic framework called DeepWL for combinatorial "coloring" algorithms that allows it to selectively color tuples within isomorphism-invariant families and thereby color tuples of length linear in the input graphs. DeepWL can distinguish the CFI graphs in polynomial time.

While Theorem 3.4 clearly shows its limitations, the Weisfeiler-Leman algorithm still remains quite powerful. In particular, it does yield complete isomorphism tests for many natural graph classes. Let us say that k-WL *identifies* a graph G if it distinguishes G from all graphs H that are not isomorphic to G. It is long known [9] that even 1-WL (that is, the Color Refinement algorithm) identifies almost all graphs, in the sense that in the limit, the fraction of n-vertex graphs identified by 1-WL is 1 as n goes to infinity. Similarly, 2-WL identifies almost all d-regular graphs [16].

The *WL dimension* of a graph G is the least $k \geq 1$ such that k-WL identifies G. Trivially, the WL dimension of an n-vertex graph is at most n (in fact, $n - 1$ for $n \geq 2$). The *WL dimension* of a class $\mathcal{C}$ of graphs is the maximum of the WL dimensions of the graphs in $\mathcal{C}$ if this maximum exists, or ∞ otherwise. It is easy to see that the WL dimension of the class of forests is 1. Building on earlier results for graphs of bounded genus and bounded tree width [33, 34, 38], the first author proved the following far-reaching result.

Theorem 3.5 (Grohe [35]) *All graph classes excluding a fixed graph as a minor have bounded WL dimension.*

Explicit bounds on the WL dimension are known for specific graph classes. Most notably, Kiefer, Ponomarenko, and Schweitzer [54] proved that the WL dimension of planar graphs is at most 3. The example of two triangles vs a cycle of length

6 shows that it is at least 2. More generally, the WL dimension of the class of all graphs embeddable in a surface of Euler genus g is at most $4g + 3$ [37]; a lower bound linear in g follows from Theorem 3.4. The second author jointly with Kiefer [53] proved that the WL dimension of the class of graphs of tree width k is at least $\lceil k/2 \rceil - 3$ and at most k.

There are also natural graph classes that do not exclude a fixed graph as a minor (and that are therefore not covered by Theorem 3.5). The WL dimension of the class of interval graphs is 2 [30] and the WL dimension of the class of graphs of rank width at most k is at most $3k+4$ [40]. Contrasting all these results, Theorem 3.4 shows that the WL dimension of the class of 3-regular graphs is ∞, even though isomorphism of 3-regular graphs can be decided in polynomial time [61] (cf. Theorem 5.4). Much more on this topic can be found in Kiefer's recent survey [52].

3.4 Characterisations of WL

The relevance of the Weisfeiler-Leman algorithm is not restricted to GI. It has found applications in linear optimisation [36] and probabilistic inferences [1], and because of relations to graph kernels [83] and graph neural networks [69, 88], received attention in machine learning recently. On a theoretical level, connections to other areas are induced by several seemingly unrelated characterizations of k-indistinguishability in terms of logic, combinatorics, and algebra.

We start with a logical characterization going back to [19, 49]. Let C be the extension of first-order predicate logic using counting quantifiers of the form $\exists^{\geq \ell} x$ for $\ell \geq 1$, where $\exists^{\geq \ell} x\, \varphi(x)$ means that there are at least ℓ elements x satisfying φ. By C^k we denote the fragment of C consisting of formulas with at most k variables. For example, the following C^2-sentence φ says that all vertices of a graph have fewer than 3 neighbors of degree at least 10:

$$\varphi := \forall x \neg \exists^{\geq 3} y \big(E(x,y) \wedge \exists^{\geq 10} x\, E(y,x) \big).$$

Note that we reuse variable x for the innermost quantification to keep the number of distinct variables down to two.

Theorem 3.6 (Cai, Fürer and Immerman [19]) *Let $k \geq 1$. Then two graphs are k-indistinguishable if and only if they are C^{k+1}-equivalent, that is, they satisfy the same sentences of the logic C^{k+1}.*

This logical characterization of k-indistinguishability plays a crucial role in the proof of Theorem 3.5 and, via a game characterization of C^k-equivalence, in the proof of Theorem 3.4.

The next characterization we will discuss is combinatorial. Recall that a graph homomorphism is a mapping that preserves adjacency, and let $\hom(G, H)$ denote the number of homomorphism from G to H. An old theorem due to Lovász [60] states that two graphs G, H are isomorphic if and only if $\hom(F, G) = \hom(F, H)$ for all graphs F. If, instead of all graphs F, we only count homomorphisms from graphs of tree width k, we obtain a characterization of k-indistinguishability.

Theorem 3.7 (Dvorák [28]) *Let $k \geq 1$. Then two graphs G, H are k-indistinguishable if and only if $\hom(F, G) = \hom(F, H)$ for all graphs F of tree width at most k.*

Maybe the most surprising characterization of k-indistinguishability is algebraic. For simplicity, we focus on $k = 1$ here. Let G, H be graphs with vertex sets $V \coloneqq V(G), W \coloneqq V(H)$, and let $A \in \mathbb{R}^{V \times V}, B \in \mathbb{R}^{W \times W}$ be their adjacency matrices. Then G and H are isomorphic if and only if there is a permutation matrix X such that $X^{-1}AX = B$, or equivalently, $AX = XB$. We can write this as a system of linear (in)equalities:

$$\sum_{v' \in V} A_{vv'} X_{v'w} = \sum_{w' \in W} X_{vw'} B_{w'w}, \tag{3.2}$$

$$\sum_{w' \in W} X_{vw'} = \sum_{v' \in V} X_{v'w} = 1, \tag{3.3}$$

$$X_{vw} \geq 0, \qquad\qquad \text{for all } v \in V, w \in W. \tag{3.4}$$

Integer solutions to this system are precisely permutation matrices $X = (X_{vw})$ satisfying $AX = XB$. Thus the system (3.2)–(3.4) has an integer solution if and only if G and H are isomorphic. Now let us drop the integrality constraints and consider rational solutions to the system, which can be viewed as doubly stochastic matrices X satisfying $AX = XB$. Let us call such solutions *fractional isomorphisms*.

Theorem 3.8 (Tinhofer [84]) *Two graphs G, H are 1-distinguishable if and only if they are fractionally isomorphic, that is, the system (3.2)–(3.4) has a rational solution.*

Atserias and Maneva [2] showed that by considering the Sherali-Adams hierarchy over the integer linear program (3.2)–(3.4), we obtain a characterization of k-indistinguishability. To be precise, solvability of the level-k Sherali-Adams (in)equalities yields an equivalence relation strictly between k-indistinguishability and $(k + 1)$-indistinguishability. By combining the equations of levels $(k − 1)$ and k, we obtain an exact characterization of k-indistinguishability [44, 62]. A similar, albeit looser correspondence to k-indistinguishability has been obtained for the Lassere hierarchy of semi-definite relaxations of the integer linear program (3.2)–(3.4) [3, 73] and in terms of polynomial equations and the so-called polynomial calculus [3, 14, 32].

4   The Group-Theoretic Graph Isomorphism Machinery

In this section we present recent advances on the group-theoretic graph isomorphism machinery. After giving some basics on permutation groups we first discuss Luks's algorithm for testing isomorphism of graphs of bounded degree [61]. This algorithm forms the foundation of the group-theoretic graph isomorphism machinery. Then, we give a very brief overview on Babai's quasi-polynomial-time isomorphism test [6] which also builds on Luks's algorithm and attacks the obstacle cases where the recursion performed by Luks's algorithm does not give the desired running time. Finally, we present recent extensions of Babai's techniques resulting in more efficient isomorphism tests for bounded-degree graphs [41] and hypergraphs [71].

4.1 Basics

We introduce the group-theoretic notions required in this work. For a general background on group theory we refer to [80]; background on permutation groups can be found in [26].

Permutation Groups. A *permutation group* acting on a set Ω is a subgroup $\Gamma \leq \mathrm{Sym}(\Omega)$ of the symmetric group. The size of the permutation domain Ω is called the *degree* of Γ. Throughout this section, the degree is denoted by $n := |\Omega|$. If $\Omega = [n]$, then we also write S_n instead of $\mathrm{Sym}(\Omega)$. The *alternating group* on a set Ω is denoted by $\mathrm{Alt}(\Omega)$. As before, we write A_n instead of $\mathrm{Alt}(\Omega)$ if $\Omega = [n]$. For $\gamma \in \Gamma$ and $\alpha \in \Omega$ we denote by α^γ the image of α under the permutation γ. The set $\alpha^\Gamma := \{\alpha^\gamma \mid \gamma \in \Gamma\}$ is the *orbit* of α. The group Γ is *transitive* if $\alpha^\Gamma = \Omega$ for some (and therefore every) $\alpha \in \Omega$.

For $\alpha \in \Omega$ the group $\Gamma_\alpha := \{\gamma \in \Gamma \mid \alpha^\gamma = \alpha\} \leq \Gamma$ is the *stabilizer* of α in Γ. The group Γ is *semi-regular* if $\Gamma_\alpha = \{\mathrm{id}\}$ for every $\alpha \in \Omega$ (where id denotes the identity element of the group). The *pointwise stabilizer* of a set $A \subseteq \Omega$ is the subgroup $\Gamma_{(A)} := \{\gamma \in \Gamma \mid \forall \alpha \in A\colon \alpha^\gamma = \alpha\}$. For $A \subseteq \Omega$ and $\gamma \in \Gamma$ let $A^\gamma := \{\alpha^\gamma \mid \alpha \in A\}$. The set A is Γ-*invariant* if $A^\gamma = A$ for all $\gamma \in \Gamma$. The *setwise stabilizer* of a set $A \subseteq \Omega$ is the subgroup $\Gamma_A := \{\gamma \in \Gamma \mid A^\gamma = A\}$.

For $A \subseteq \Omega$ and a bijection $\theta\colon \Omega \to \Omega'$ we denote by $\theta[A]$ the restriction of θ to the domain A. For a Γ-invariant set $A \subseteq \Omega$, we denote by $\Gamma[A] := \{\gamma[A] \mid \gamma \in \Gamma\}$ the induced action of Γ on A, i.e., the group obtained from Γ by restricting all permutations to A.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a transitive group. A *block* of Γ is a nonempty subset $B \subseteq \Omega$ such that $B^\gamma = B$ or $B^\gamma \cap B = \emptyset$ for all $\gamma \in \Gamma$. The trivial blocks are Ω and the singletons $\{\alpha\}$ for $\alpha \in \Omega$. The group Γ is called *primitive* if there are no non-trivial blocks. If $B \subseteq \Omega$ is a block of Γ then $\mathfrak{B} := \{B^\gamma \mid \gamma \in \Gamma\}$ builds a *block system* of Γ. Note that $\mathfrak{B}$ is an equipartition of Ω. The group $\Gamma_{(\mathfrak{B})} := \{\gamma \in \Gamma \mid \forall B \in \mathfrak{B}\colon B^\gamma = B\}$ denotes the subgroup stabilizing each block $B \in \mathfrak{B}$ setwise. Moreover, the natural action of Γ on the block system $\mathfrak{B}$ is denoted by $\Gamma[\mathfrak{B}] \leq \mathrm{Sym}(\mathfrak{B})$. Let $\mathfrak{B}, \mathfrak{B}'$ be two partitions of Ω. We say that $\mathfrak{B}$ *refines* $\mathfrak{B}'$, denoted $\mathfrak{B} \preceq \mathfrak{B}'$, if for every $B \in \mathfrak{B}$ there is some $B' \in \mathfrak{B}'$ such that $B \subseteq B'$. If additionally $\mathfrak{B} \neq \mathfrak{B}'$ we write $\mathfrak{B} \prec \mathfrak{B}'$. A block system $\mathfrak{B}$ is *minimal* if there is no non-trivial block system $\mathfrak{B}'$ such that $\mathfrak{B} \prec \mathfrak{B}'$. A block system $\mathfrak{B}$ is minimal if and only if $\Gamma[\mathfrak{B}]$ is primitive.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ and $\Gamma' \leq \mathrm{Sym}(\Omega')$. A *homomorphism* is a mapping $\varphi\colon \Gamma \to \Gamma'$ such that $\varphi(\gamma)\varphi(\delta) = \varphi(\gamma\delta)$ for all $\gamma, \delta \in \Gamma$. For $\gamma \in \Gamma$ we denote by γ^φ the φ-image of γ. Similarly, for $\Delta \leq \Gamma$ we denote by Δ^φ the φ-image of Δ (note that Δ^φ is a subgroup of Γ').

Algorithms for Permutation Groups. Next, we review some basic facts about algorithms for permutation groups. For detailed information we refer to [82].

In order to handle permutation groups computationally it is essential to represent groups in a compact way. Indeed, the size of a permutation group is typically exponential in the degree of the group which means it is not possible to store the whole group in memory. In order to allow for efficient computation, permutation groups are represented by generating sets. By Lagrange's Theorem, for each permutation

group $\Gamma \leq \mathrm{Sym}(\Omega)$, there is a generating set of size $\log |\Gamma| \leq n \log n$ (recall that $n := |\Omega|$ denotes the size of the permutation domain). Actually, most algorithms are based on so-called *strong generating sets*, which can be chosen of size quadratic in the degree of the group and can be computed in polynomial time given an arbitrary generating set (see, e.g., [82]).

Theorem 4.1 (cf. [82]) *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ and let S be a generating set for Γ. Then the following tasks can be performed in time polynomial in n and $|S|$:*

1. *compute the order of Γ,*

2. *given $\gamma \in \mathrm{Sym}(\Omega)$, test whether $\gamma \in \Gamma$,*

3. *compute the orbits of Γ,*

4. *compute a minimal block system of Γ (if Γ is transitive), and*

5. *given $A \subseteq \Omega$, compute a generating set for $\Gamma_{(A)}$.*

Let $\Delta \leq \mathrm{Sym}(\Omega')$ be a second group of degree $n' = |\Omega'|$. Also let $\varphi: \Gamma \to \Delta$ be a homomorphism given by a list of images for $s \in S$. Then the following tasks can be solved in time polynomial in n, n' and $|S|$:

6. *compute a generating set for $\ker(\varphi) := \{\gamma \in \Gamma \mid \gamma^\varphi = \mathrm{id}\}$, and*

7. *given $\delta \in \Delta$, find $\gamma \in \Gamma$ such that $\gamma^\varphi = \delta$ (if it exists).*

Observe that, by always maintaining generating sets of size at most quadratic in the degree, we can concatenate a polynomial number of subroutines from the theorem while keeping the polynomial-time bound. For the remainder of this work, we will typically ignore the role of generating sets and simply refer to groups being the input/output of an algorithm. This always means the algorithm performs computations on generating sets of size polynomial in the degree of the group.

4.2 Luks's Algorithm

The String Isomorphism Problem. In order to apply group-theoretic techniques to the Graph Isomorphism Problem, Luks [61] introduced a more general problem that allows to build recursive algorithms along the structure of the permutation groups involved. Here, we follow the notation and terminology used by Babai [6] for describing his quasi-polynomial-time algorithm for the Graph Isomorphism Problem that also employs this recursive strategy.

A *string* is a mapping $\mathfrak{x}: \Omega \to \Sigma$ where Ω is a finite set and Σ is also a finite set called the *alphabet*. Let $\gamma \in \mathrm{Sym}(\Omega)$ be a permutation. The permutation γ can be applied to the string $\mathfrak{x}$ by defining

$$\mathfrak{x}^\gamma: \Omega \to \Sigma: \alpha \mapsto \mathfrak{x}\left(\alpha^{\gamma^{-1}}\right). \tag{4.1}$$

Let $\mathfrak{y}: \Omega \to \Sigma$ be a second string. The permutation γ is an isomorphism from $\mathfrak{x}$ to $\mathfrak{y}$, denoted $\gamma: \mathfrak{x} \cong \mathfrak{y}$, if $\mathfrak{x}^\gamma = \mathfrak{y}$. Let $\Gamma \leq \mathrm{Sym}(\Omega)$. A Γ-*isomorphism* from $\mathfrak{x}$ to $\mathfrak{y}$ is a permutation $\gamma \in \Gamma$ such that $\gamma: \mathfrak{x} \cong \mathfrak{y}$. The strings $\mathfrak{x}$ and $\mathfrak{y}$ are Γ-*isomorphic*, denoted
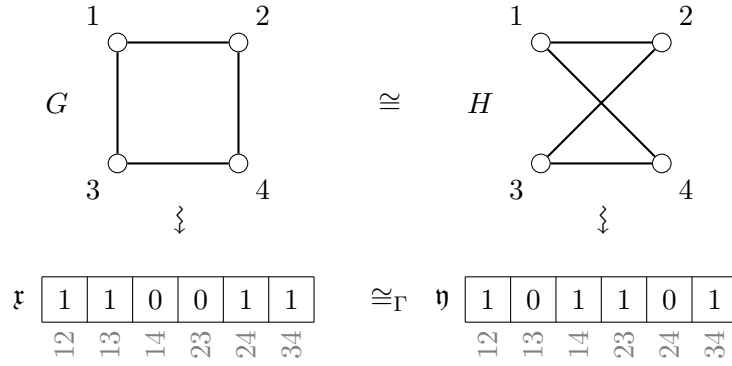
Figure 4: Reduction from the Graph Isomorphism Problem to the String Isomorphism Problem. The group Γ is the natural induced action of S_4 on $\binom{[4]}{2}$.

$\mathfrak{x} \cong_\Gamma \mathfrak{y}$, if there is a Γ-isomorphism from $\mathfrak{x}$ to $\mathfrak{y}$. The *String Isomorphism Problem* asks, given two strings $\mathfrak{x}, \mathfrak{y} \colon \Omega \to \Sigma$ and a generating set for a group $\Gamma \leq \mathrm{Sym}(\Omega)$, whether $\mathfrak{x}$ and $\mathfrak{y}$ are Γ-isomorphic. The set of Γ-isomorphisms from $\mathfrak{x}$ to $\mathfrak{y}$ is denoted by

$$\mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y}) := \{\gamma \in \Gamma \mid \mathfrak{x}^\gamma = \mathfrak{y}\}. \tag{4.2}$$

The set of Γ-*automorphisms* of $\mathfrak{x}$ is $\mathrm{Aut}_\Gamma(\mathfrak{x}) := \mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{x})$. Observe that $\mathrm{Aut}_\Gamma(\mathfrak{x})$ is a subgroup of Γ and $\mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y}) = \mathrm{Aut}_\Gamma(\mathfrak{x})\gamma := \{\delta\gamma \mid \delta \in \mathrm{Aut}_\Gamma(\mathfrak{x})\}$ for an arbitrary $\gamma \in \mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y})$.

Theorem 4.2 *There is a polynomial-time many-one reduction from the Graph Isomorphism Problem to the String Isomorphism Problem.*

Proof Let G and H be two graphs and assume without loss of generality $V(G) = V(H) =: V$. Let $\Omega := \binom{V}{2}$ and $\Gamma \leq \mathrm{Sym}(\Omega)$ be the natural action of $\mathrm{Sym}(V)$ on the two-element subsets of V. Let $\mathfrak{x} \colon \Omega \to \{0, 1\}$ with $\mathfrak{x}(vw) = 1$ if and only if $vw \in E(G)$. Similarly define $\mathfrak{y} \colon \Omega \to \{0, 1\}$ with $\mathfrak{y}(vw) = 1$ if and only if $vw \in E(H)$ (see Figure 4). Let $\varphi \colon \mathrm{Sym}(V) \to \Gamma$ be the natural homomorphism. Then $\gamma \colon G \cong H$ if and only if $\gamma^\varphi \colon \mathfrak{x} \cong \mathfrak{y}$. $\qquad\square$

The main advantage of the String Isomorphism Problem is that it naturally allows for algorithmic approaches based on group-theoretic techniques.

Recursion Mechanisms. The foundation for the group-theoretic approaches lays in two recursion mechanisms first exploited by Luks [61]. As before, let $\mathfrak{x}, \mathfrak{y} \colon \Omega \to \Sigma$ be two strings. For a set of permutations $K \subseteq \mathrm{Sym}(\Omega)$ and a *window* $W \subseteq \Omega$ define

$$\mathrm{Iso}_K^W(\mathfrak{x}, \mathfrak{y}) := \{\gamma \in K \mid \forall \alpha \in W \colon \mathfrak{x}(\alpha) = \mathfrak{y}(\alpha^\gamma)\}. \tag{4.3}$$

In the following, the set K is always a *coset*, i.e., $K = \Gamma\gamma := \{\gamma'\gamma \mid \gamma' \in \Gamma\}$ for some group $\Gamma \leq \mathrm{Sym}(\Omega)$ and a permutation $\gamma \in \mathrm{Sym}(\Omega)$, and the set W is Γ-invariant. In this case it can be shown that $\mathrm{Iso}_K^W(\mathfrak{x}, \mathfrak{y})$ is either empty or a coset of the group $\mathrm{Aut}_\Gamma^W(\mathfrak{x}) := \mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{x})$. Hence, the set $\mathrm{Iso}_K^W(\mathfrak{x}, \mathfrak{y})$ can be represented by a generating

set for $\mathrm{Aut}_\Gamma^W(\mathfrak{x})$ and a single permutation $\gamma \in \mathrm{Iso}_K^W(\mathfrak{x}, \mathfrak{y})$. Moreover, for $K = \Gamma\gamma$, it holds that

$$\mathrm{Iso}_{\Gamma\gamma}^W(\mathfrak{x}, \mathfrak{y}) = \mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{y}^{\gamma^{-1}})\gamma. \tag{4.4}$$

Using this identity, it is possible to restrict to the case where K is actually a group.

With these definitions we can now formulate the two recursion mechanisms introduced by Luks [61]. For the first type of recursion suppose $K = \Gamma \leq \mathrm{Sym}(\Omega)$ is not transitive on W and let $W_1, \ldots, W_\ell$ be the orbits of $\Gamma[W]$. Then the strings can be processed orbit by orbit as follows. We iterate over all $i \in [\ell]$ and update $K := \mathrm{Iso}_K^{W_i}(\mathfrak{x}, \mathfrak{y})$ in each iteration. In the end, $K = \mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{y})$. This recursion mechanism is referred to as *orbit-by-orbit processing*.

The set $\mathrm{Iso}_K^{W_i}(\mathfrak{x}, \mathfrak{y})$ can be computed by making one recursive call to the String Isomorphism Problem over domain size $n_i := |W_i|$. Indeed, using Equation (4.4), it can be assumed that K is a group and W_i is K-invariant. Then

$$\mathrm{Iso}_K^{W_i}(\mathfrak{x}, \mathfrak{y}) = \{\gamma \in K \mid \gamma[W_i] \in \mathrm{Iso}_{K[W_i]}(\mathfrak{x}[W_i], \mathfrak{y}[W_i])\} \tag{4.5}$$

where $\mathfrak{x}[W_i]$ denotes the induced substring of $\mathfrak{x}$ on the set W_i, i.e., $\mathfrak{x}[W_i]: W_i \to \Sigma: \alpha \mapsto \mathfrak{x}(\alpha)$ (the string $\mathfrak{y}[W_i]$ is defined analogously). So overall, if Γ is not transitive, the set $\mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{y})$ can be computed by making ℓ recursive calls over window size $n_i = |W_i|$, $i \in [\ell]$.

For the second recursion mechanism let $\Delta \leq \Gamma$ and let T be a *transversal*[2] for Δ in Γ. Then

$$\mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{y}) = \bigcup_{\delta \in T} \mathrm{Iso}_{\Delta\delta}^W(\mathfrak{x}, \mathfrak{y}) = \bigcup_{\delta \in T} \mathrm{Iso}_\Delta^W(\mathfrak{x}, \mathfrak{y}^{\delta^{-1}})\delta. \tag{4.6}$$

Luks applied this type of recursion when Γ is transitive (on the window W), $\mathfrak{B}$ is a minimal block system for Γ, and $\Delta = \Gamma_{(\mathfrak{B})}$. In this case $\Gamma[\mathfrak{B}]$ is a primitive group. Let $t = |\Gamma[\mathfrak{B}]|$ be the size of a transversal for Δ in Γ. Note that Δ is not transitive (on the window W). Indeed, each orbit of Δ has size at most n/b where $b = |\mathfrak{B}|$. So by combining both types of recursion the computation of $\mathrm{Iso}_\Gamma^W(\mathfrak{x}, \mathfrak{y})$ is reduced to $t \cdot b$ many instances of the String Isomorphism Problem over window size $|W|/b$. This specific combination of types of recursion is referred to as *standard Luks reduction*. Observe that the time complexity of standard Luks reduction is determined by the size of the primitive group $\Gamma[\mathfrak{B}]$.

Overall, Luks's algorithm is formulated in Algorithm 1. The running time of this algorithm heavily depends on the size of the primitive groups involved in the computation.

The Role of Primitive Groups. In order to analyse the running time of Luks's algorithm (on a specific input group Γ) the crucial step is to understand which primitive groups appear along the execution of the algorithm and to bound their size as a function of their degree. Let us first analyse the situation for GI for graphs of maximum degree d. The crucial observation is that there is a polynomial-time Turing reduction from this problem to the String Isomorphism Problem where the input group Γ is contained the class $\widehat{\Gamma}_d$ [61, 11].

---

[2] A set $T \subseteq \Gamma$ is a *transversal* for Δ in Γ if $|T| = |\Gamma|/|\Delta|$ and $\{\Delta\delta \mid \delta \in T\} = \{\Delta\delta \mid \delta \in \Gamma\}$ (such a set always exists).

Algorithm 1: Luks's Algorithm: $\mathsf{StringIso}(\mathfrak{x}, \mathfrak{y}, \Gamma, \gamma, W)$

Input : Strings $\mathfrak{x}, \mathfrak{y} \colon \Omega \to \Sigma$, a group $\Gamma \leq \mathrm{Sym}(\Omega)$, a permutation
$\gamma \in \mathrm{Sym}(\Omega)$, and a Γ-invariant set $W \subseteq \Omega$

Output: $\mathrm{Iso}_{\Gamma\gamma}^{W}(\mathfrak{x}, \mathfrak{y})$

1 if $\gamma \neq 1$ **then**
2 $\quad$ **return** $\mathsf{StringIso}(\mathfrak{x}, \mathfrak{y}^{\gamma^{-1}}, \Gamma, 1, W)\gamma$ $\qquad\qquad\qquad$ /* Equation (4.4) */
3 end
4 if $|W| = 1$ **then**
5 $\quad$ **if** $\mathfrak{x}(\alpha) = \mathfrak{y}(\alpha)$ *where* $W = \{\alpha\}$ **then**
6 $\quad\quad$ **return** Γ
7 $\quad$ **else**
8 $\quad\quad$ **return** $\emptyset$
9 $\quad$ **end**
10 end
11 if Γ *is not transitive on* W **then**
12 $\quad$ compute orbit $W' \subseteq W$
13 $\quad$ **return** $\mathsf{StringIso}(\mathfrak{x}, \mathfrak{y}, \mathsf{StringIso}(\mathfrak{x}, \mathfrak{y}, \Gamma, 1, W'), W \setminus W')$
14 end
15 compute minimal block system $\mathfrak{B}$ of the action of Γ on W
16 compute $\Delta := \Gamma_{(\mathfrak{B})}$
17 compute transversal T of Δ in Γ
18 return $\bigcup_{\delta \in T} \mathsf{StringIso}(\mathfrak{x}, \mathfrak{y}, \Delta, \delta, W)$

Let Γ be a group. A *subnormal series* is a sequence of subgroups $\Gamma = \Gamma_0 \trianglerighteq \Gamma_1 \trianglerighteq \cdots \trianglerighteq \Gamma_k = \{\mathrm{id}\}$. The length of the series is k and the groups Γ_{i-1}/Γ_i are the factor groups of the series, $i \in [k]$. A *composition series* is a strictly decreasing subnormal series of maximal length. For every finite group Γ all composition series have the same family (considered as a multiset) of factor groups (cf. [80]). A *composition factor* of a finite group Γ is a factor group of a composition series of Γ.

Definition 4.3 For $d \geq 2$ let $\widehat{\Gamma}_d$ denote the class of all groups Γ for which every composition factor of Γ is isomorphic to a subgroup of S_d.

We want to stress the fact that there are two similar classes of groups that have been used in the literature both typically denoted by Γ_d. One of these is the class introduced by Luks [61] that we denote by $\widehat{\Gamma}_d$, while the other one used in [7] in particular allows composition factors that are simple groups of Lie type of bounded dimension.

Lemma 4.4 (Luks [61]) *Let $\Gamma \in \widehat{\Gamma}_d$. Then*

1. *$\Delta \in \widehat{\Gamma}_d$ for every subgroup $\Delta \leq \Gamma$, and*

2. *$\Gamma^\varphi \in \widehat{\Gamma}_d$ for every homomorphism $\varphi \colon \Gamma \to \Delta$.*

Now, the crucial advantage of $\widehat{\Gamma}_d$-groups with respect to Luks's algorithm is that the size of primitive $\widehat{\Gamma}_d$-groups is bounded polynomially in the degree for every fixed d.

Theorem 4.5 *There is a function $f \colon \mathbb{N} \to \mathbb{N}$ such that for every primitive permutation group $\Gamma \in \widehat{\Gamma}_d$ it holds that $|\Gamma| \leq n^{f(d)}$. Moreover, the function f can be chosen such that $f(d) = \mathcal{O}(d)$.*

The first part of this theorem on the existence of the function f was first proved by Babai, Cameron and Pálfy [7] implying that Luks's algorithm runs in polynomial time for every fixed number $d \in \mathbb{N}$. Later, it was observed that the function f can actually be chosen to be linear in d (see, e.g., [56]).

Theorem 4.5 allows the analysis of Luks's algorithm when the input group comes from the class $\widehat{\Gamma}_d$. Note that the class $\widehat{\Gamma}_d$ is closed under subgroups and homomorphic images by Lemma 4.4 which implies that all groups encountered during Luks's algorithm are from the class $\widehat{\Gamma}_d$ in case the input group is a $\widehat{\Gamma}_d$-group.

Corollary 4.6 *Luks's Algorithm (Algorithm 1) for groups $\Gamma \in \widehat{\Gamma}_d$ runs in time $n^{\mathcal{O}(d)}$.*

Proof Sketch   In order to analyze the running time of Algorithm 1 let $f(n)$ denote the maximal number of leaves in a recursion tree for a $\widehat{\Gamma}_d$-group where $n := |W|$ denotes the window size. For $n = 1$ it is easy to see that $f(n) = 1$. Suppose Γ is not transitive on W and let $n_1 := |W'|$ be the size of an orbit $W' \subseteq W$. Then

$$f(n) \leq f(n_1) + f(n - n_1). \tag{4.7}$$

Finally, if Γ is transitive on W, the algorithm computes a minimal block system $\mathfrak{B}$ of $\Gamma[W]$ and performs standard Luks reduction. In this case the algorithm performs $t \cdot b$ recursive calls over window size n/b where $t = |\Gamma[\mathfrak{B}]|$. Hence,

$$f(n) \leq t \cdot b \cdot f(n/b). \tag{4.8}$$

Since $|\Gamma[\mathfrak{B}]|$ is a primitive $\widehat{\Gamma}_d$-group it holds that $t = b^{\mathcal{O}(d)}$ by Theorem 4.5. Combining the bound on t with Equation (4.7) and (4.8) gives $f(n) = n^{\mathcal{O}(d)}$. Also, each node of the recursion tree only requires computation time polynomial in n and the number of children in the recursion tree (cf. Theorem 4.1). Overall, this gives the desired bound on the running time. $\qquad\square$

In combination with the polynomial-time Turing reduction from the Graph Isomorphism Problem for graphs of maximum degree d to the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups [61, 11] the former problem can be solved in the same running time $n^{\mathcal{O}(d)}$.

Unfortunately, for the general String Isomorphism Problem, the situation is more complicated. As a very simple example, symmetric and alternating groups S_n and A_n are primitive groups of exponential size (which means that Luks's algorithm performs a brute-force search over all elements of the group). However, even in the case of arbitrary input groups, the recursion techniques of Luks are still a powerful tool. The main reason is that large primitive groups, which build the bottleneck cases for Luks's algorithm, are quite rare and well understood.

Let $m \in \mathbb{N}$, $t \leq \frac{m}{2}$ and denote $\binom{[m]}{t}$ to be the set of all t-element subsets of $[m]$. Let $S_m^{(t)} \leq \mathrm{Sym}(\binom{[m]}{t})$ denote the natural induced action of S_m on the set $\binom{[m]}{t}$. Similarly, let $A_m^{(t)} \leq \mathrm{Sym}(\binom{[m]}{t})$ denote the natural induced action of A_m on the set $\binom{[m]}{t}$. Following Babai [6], we refer to these groups as the *Johnson groups*.

The classification of large primitive groups by Cameron [20] states that a primitive permutation group of order $|\Gamma| \geq n^{1+\log n}$ is necessarily a so-called *Cameron group* which involves a large Johnson group. Here, we only state the following slightly weaker result which is sufficient for most algorithmic purposes.

Theorem 4.7 *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a primitive group of order $|\Gamma| \geq n^{1+\log n}$ where n is greater than some absolute constant. Then there is a polynomial-time algorithm computing a normal subgroup $N \trianglelefteq \Gamma$ of index $|\Gamma : N| \leq n$ and an N-invariant equipartition $\mathfrak{B}$ such that $N[\mathfrak{B}]$ is permutationally equivalent to $A_m^{(t)}$ for some $m \geq \log n$.*

The mathematical part of this theorem follows from [20, 63] whereas the algorithmic part is resolved in [12] (see also [5, Theorem 3.2.1]). The theorem exactly characterizes the obstacle cases of Luks's algorithm and can be seen as the starting point for Babai's algorithm solving the String Isomorphism Problem in quasi-polynomial time [6].

4.3 Babai's Algorithm

Next, we give a brief overview on the main ideas of Babai's quasi-polynomial-time algorithm for the String Isomorphism Problem. By Theorem 4.2 this also gives

an algorithm for the Graph Isomorphism Problem with essentially the same running time.

As already indicated above the basic strategy of Babai's algorithm is to follow standard Luks recursion until the algorithm encounters an obstacle group which, by Theorem 4.7, may be assumed to be a Johnson group. In order to algorithmically handle the case of Johnson groups Babai's algorithm utilizes several subroutines based on both group-theoretic techniques and combinatorial approaches like the Weisfeiler-Leman algorithm.

Let us start with discussing the group-theoretic subroutines, specifically the *Local Certificates Routine* which is based on two group-theoretic statements, the *Unaffected Stabilizers Theorem* and the *Affected Orbit Lemma*.

Recall that for a set M we denote by $\mathrm{Alt}(M)$ the alternating group acting with its standard action on the set M. Moreover, following Babai [6], we refer to the groups $\mathrm{Alt}(M)$ and $\mathrm{Sym}(M)$ as the *giants* where M is an arbitrary finite set. Let $\Gamma \leq \mathrm{Sym}(\Omega)$. A *giant representation* is a homomorphism $\varphi \colon \Gamma \to S_k$ such that $\Gamma^\varphi \geq A_k$. Observe that Johnson groups naturally admit giant representations and thus, the obstacle cases from Theorem 4.7 also have a giant representation (this is a main feature of an obstacle exploited algorithmically).

Given a string $\mathfrak{x} \colon \Omega \to \Sigma$, a group $\Gamma \leq \mathrm{Sym}(\Omega)$ and a giant representation $\varphi \colon \Gamma \to S_k$, the aim of the Local Certificates Routine is to determine whether $(\mathrm{Aut}_\Gamma(\mathfrak{x})^\varphi) \geq A_k$ and to compute a meaningful certificate in either case. To achieve this goal the central tool is to split the set Ω into *affected* and *non-affected* points.

Definition 4.8 (Affected Points, Babai [6]) Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a group and $\varphi \colon \Gamma \to S_k$ a giant representation. Then an element $\alpha \in \Omega$ is *affected by* φ if $\Gamma_\alpha^\varphi \not\geq A_k$.

We remark that, if $\alpha \in \Omega$ is affected by φ, then every element in the orbit α^Γ is affected by φ. Hence, the set α^Γ is called an *affected orbit* (with respect to φ).

Let $\Delta \leq \Gamma$ be a subgroup. We define $\mathrm{Aff}(\Delta, \varphi) := \{\alpha \in \Omega \mid \Delta_\alpha^\varphi \not\geq A_k\}$. Observe that, for $\Delta_1 \leq \Delta_2 \leq \Gamma$ it holds that $\mathrm{Aff}(\Delta_1, \varphi) \supseteq \mathrm{Aff}(\Delta_2, \varphi)$.

The Local Certificates Routine is described in Algorithm 2. The algorithm computes a sequence of groups $\Gamma = \Gamma_0 \geq \Gamma_1 \geq \cdots \geq \Gamma_i$ as well as a sequence of windows $\emptyset = W_0 \subseteq W_1 \subseteq \cdots \subseteq W_i$. Throughout, the algorithm maintains the property that $\Gamma_i = \mathrm{Aut}_\Gamma^{W_i}(\mathfrak{x})$, i.e., the algorithm tries to "approximate" the automorphism group $\mathrm{Aut}_\Gamma(\mathfrak{x})$ taking larger and larger substrings into account. Observe that W_{i+1} is Γ_i-invariant and thus, $\mathrm{Aut}_\Gamma(\mathfrak{x}) \leq \Gamma_{i+1}$.

When growing the window W_i the algorithm adds those points $\alpha \in \Omega$ that are affected (with respect to the current group Γ_i) and computes the group Γ_{i+1} using recursion. The while-loop is terminated as soon as $\Gamma_i^\varphi \not\geq A_k$ or the window W_i stops growing. In the first case the algorithm returns $\Lambda := \Gamma_i^\varphi$ which clearly gives the desired outcome. In the second case the algorithm returns $\Delta := (\Gamma_i)_{(\Omega \setminus W_i)}$. It is easy to verify that $\Delta \leq \mathrm{Aut}_\Gamma(\mathfrak{x})$ since all points inside the window W_i are taken into account by the definition of group Γ_i and all points outside of W_i are fixed. The key insight is that Δ contains a large number of automorphisms, more specifically, $\Delta^\varphi \geq A_k$. This is guaranteed by the *Unaffected Stabilizers Theorem*, one of the main conceptual contributions of Babai's algorithm.

Algorithm 2: LocalCertificates($\mathfrak{x}, \Gamma, \varphi$)

> **Input** : A string $\mathfrak{x} \colon \Omega \to \Sigma$, a group $\Gamma \leq \mathrm{Sym}(\Omega)$, and a giant
> representation $\varphi \colon \Gamma \to S_k$ such that $k \geq \max\{8, 2 + \log_2 n\}$
> **Output:** A non-giant $\Lambda \leq S_k$ with $(\mathrm{Aut}_\Gamma(\mathfrak{x}))^\varphi \leq \Lambda$ or $\Delta \leq \mathrm{Aut}_\Gamma(\mathfrak{x})$ with
> $\Delta^\varphi \geq A_k$.

1 $W_0 := \emptyset$
2 $\Gamma_0 := \Gamma$
3 $i := 0$
4 **while** $\Gamma_i^\varphi \geq A_k$ **and** $W_i \neq \mathrm{Aff}(\Gamma_i, \varphi)$ **do**
5 $\quad$ $W_{i+1} := \mathrm{Aff}(\Gamma_i, \varphi)$
6 $\quad$ $N := \ker(\varphi|_{\Gamma_i})$
7 $\quad$ $\Gamma_{i+1} := \emptyset$
8 $\quad$ **for** $\gamma \in \Gamma_i^\varphi$ **do**
9 $\quad\quad$ compute $\bar{\gamma} \in \varphi^{-1}(\gamma)$
10 $\quad\quad$ $\Gamma_{i+1} := \Gamma_{i+1} \cup \mathrm{Aut}_{N\bar{\gamma}}^{W_{i+1}}(\mathfrak{x})$
11 $\quad$ **end**
12 $\quad$ $i := i + 1$
13 **end**
14 **if** $\Gamma_i^\varphi \not\geq A_k$ **then**
15 $\quad$ **return** Γ_i^φ
16 **else**
17 $\quad$ **return** $(\Gamma_i)_{(\Omega \setminus W_i)}$
18 **end**

Theorem 4.9 (Unaffected Stabilizers Theorem, Babai [6]) *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group of degree n and let $\varphi \colon \Gamma \to S_k$ be a giant representation such that $k > \max\{8, 2 + \log n\}$. Let $D \subseteq \Omega$ be the set of elements not affected by φ.*

Then $(\Gamma_{(D)})^\varphi \geq A_k$. In particular $D \neq \Omega$, that is, at least one point is affected by φ.

Overall, this gives the correctness of the Local Certificates Routine. So let us analyse the running time. Here, the crucial property is that the orbits of $N[W]$ are small which guarantees efficient standard Luks reduction when computing Γ_{i+1}.

Lemma 4.10 (Affected Orbit Lemma, Babai [6]) *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group and suppose $\varphi \colon \Gamma \to S_k$ is a giant representation for $k \geq 5$. Suppose $A \subseteq \Omega$ is an affected orbit of Γ (with respect to φ). Then every orbit of $\ker(\varphi)$ in A has size at most $|A|/k$.*

Overall, this means that the Local Certificates Routine runs in time $k!n^{\mathcal{O}(1)}$ and performs at most $k!n$ many recursive calls to the String Isomorphism Problem over domain size at most n/k. Unfortunately, this not fast enough if k is significantly larger than $\log n$. The solution to this problem is to not run the Local Certificates Routine for the entire set $[k]$, but only for *test sets* $T \subseteq [k]$ of size $t = \mathcal{O}(\log n)$.

Let $T \subseteq [k]$ be a test set of size $|T| = t$. We extend the notion of point- and setwise stabilizers to the image of φ and define $\Gamma_{(T)} := \varphi^{-1}((\Gamma^\varphi)_{(T)})$ and $\Gamma_T := \varphi^{-1}((\Gamma^\varphi)_T)$. The test set T is called *full* if $((\mathrm{Aut}_{\Gamma_T}(\mathfrak{x}))^\varphi)[T] \geq \mathrm{Alt}(T)$. A *certificate of fullness* is a group $\Delta \leq \mathrm{Aut}_{\Gamma_T}(\mathfrak{x})$ such that $(\Delta^\varphi)[T] \geq \mathrm{Alt}(T)$. A *certificate of non-fullness* is a non-giant group $\Lambda \leq \mathrm{Sym}(T)$ such that $((\mathrm{Aut}_{\Gamma_T}(\mathfrak{x}))^\varphi)[T] \leq \Lambda$. Given a test set $T \subseteq [k]$ we can use the Local Certificates Routine to determine whether T is full and, depending on the outcome, compute a certificate of fullness or non-fullness (simply run the Local Certificates Routine with input $(\mathfrak{x}, \Gamma_T, \varphi_T)$ where $\varphi_T(\gamma) = (\gamma^\varphi)[T]$ for all $\gamma \in \Gamma_T$). Observe that, for $t = \mathcal{O}(\log n)$, the recursion performed by the Local Certificate Routine for test sets of size t only results in quasi-polynomial running time.

Going back to the original problem of testing isomorphism of strings, the Local Certificates Routine is applied for all test sets of size t as well as both input strings $\mathfrak{x}, \mathfrak{y}$. Based on the results, one can achieve one of the following two outcomes:

1. subsets $M_{\mathfrak{x}}, M_{\mathfrak{y}} \subseteq [k]$ of size at least $\frac{3}{4}k$ and a group $\Delta \leq \mathrm{Aut}_{\Gamma_{M_{\mathfrak{x}}}}(\mathfrak{x})$ such that $\Delta^\varphi[M_{\mathfrak{x}}] \geq \mathrm{Alt}(M_{\mathfrak{x}})$ and $(M_{\mathfrak{x}})^\varphi(\gamma) = M_{\mathfrak{y}}$ for all $\gamma \in \mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y})$, or

2. two families of $r = k^{\mathcal{O}(1)}$ many t-ary relational structures $(\mathfrak{A}_{\mathfrak{x},j})_{j \in [r]}$ and $(\mathfrak{A}_{\mathfrak{y},j})_{j \in [r]}$, where each relational structure has domain $[k]$ and only few symmetries, such that

$$\{\mathfrak{A}_{\mathfrak{x},1}, \ldots, \mathfrak{A}_{\mathfrak{x},r}\}^{\varphi(\gamma)} = \{\mathfrak{A}_{\mathfrak{y},1}, \ldots, \mathfrak{A}_{\mathfrak{y},r}\}$$

   for all $\gamma \in \mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y})$.

Here, we shall not formally specify what it means for a structure to have few symmetries, but roughly speaking, this implies that the automorphism group is exponentially smaller than the full symmetric group.

Intuitively speaking, the first option can be achieved if there are many test sets which are full. In this case, the certificates of fullness (for the string $\mathfrak{x}$) can be combined to obtain the group Δ. Otherwise, there are many certificates of non-fullness which can be combined into the relational structures.

Now suppose the first option is satisfied. For simplicity assume that $M_{\mathfrak{x}} = M_{\mathfrak{y}} = [k]$ and $\Delta^{\varphi} = S_k$. Then $\mathrm{Iso}_{\Gamma}(\mathfrak{x}, \mathfrak{y}) \neq \emptyset$ if and only if $\mathrm{Iso}_{\ker(\varphi)}(\mathfrak{x}, \mathfrak{y}) \neq \emptyset$. Hence, it suffices to recursively determine whether $\mathrm{Iso}_{\ker(\varphi)}(\mathfrak{x}, \mathfrak{y}) \neq \emptyset$. Since $\ker(\varphi)$ is significantly smaller than Γ this gives sufficient progress for the recursion to obtain the desired running time.

In the other case the situation is far more involved and here, Babai's algorithm builds on combinatorial methods to achieve further progress.

For simplicity, suppose the algorithm is given a single pair of isomorphism-invariant relational structures $\mathfrak{A} \coloneqq \mathfrak{A}_{\mathfrak{x},1}$ and $\mathfrak{B} \coloneqq \mathfrak{A}_{\mathfrak{y},1}$ (the general case reduces to this case by *individualizing* one of the relational structures). Recall that $\mathfrak{A}$ and $\mathfrak{B}$ only have few symmetries. On a high level, the aim of the combinatorial sub-routines is to compute "simpler" isomorphism-invariant relational structures $\mathfrak{A}^*$ and $\mathfrak{B}^*$ such that $\mathfrak{A}^*$ and $\mathfrak{B}^*$ still have few symmetries and all isomorphisms between $\mathfrak{A}^*$ and $\mathfrak{B}^*$ can be computed in polynomial time. Roughly speaking, this allows the algorithm to make sufficient progress by computing $\Lambda\lambda \coloneqq \mathrm{Iso}(\mathfrak{A}^*, \mathfrak{B}^*)$ and updating $\Gamma\gamma \coloneqq \varphi^{-1}(\Lambda\lambda)$, i.e., the algorithm continues to compute $\mathrm{Iso}_{\Gamma\gamma}(\mathfrak{x}, \mathfrak{y})$. Actually, as before, we compute small families of relational structures $\mathfrak{A}_1^*, \ldots, \mathfrak{A}_m^*$ and $\mathfrak{B}_1^*, \ldots, \mathfrak{B}_m^*$ for some number m which is quasi-polynomial in k. This does not impose any additional problems since, once again, it is possible to individualize a single relational structure.

To achieve the goal, Babai's algorithm builds on two subroutines, the *Design Lemma* and the *Split-or-Johnson Routine*. The *Design Lemma* first turns the t-ary relational structures into (a small family of) graphs (without changing the domain). For this task, the *Design Lemma* mainly builds on the t-dimensional Weisfeiler-Leman algorithm. The output is passed to the *Split-or-Johnson Routine* which produces one of the following:

1. an isomorphism-invariant coloring where each color class C has size $|C| \leq \frac{3}{4}k$, or

2. an isomorphism-invariant non-trivial equipartition of a subset $M \subseteq [k]$ of size $|M| \geq \frac{3}{4}k$, or

3. an isomorphism-invariant non-trivial Johnson graph defined on a subset $M \subseteq [k]$ of size $|M| \geq \frac{3}{4}k$.

Here, the Johnson graph $J(m, t)$ is the graph with vertex set $V(J(m, t)) \coloneqq \binom{[m]}{t}$ and edge set $E(J(m, t)) \coloneqq \{XY \mid |X \setminus Y| = 1\}$. We remark that $\mathrm{Aut}(J(m, t)) = S_m^{(t)}$.

Once again, the *Split-or-Johnson Routine* actually produces a family of such objects. Note that, for each possible outcome, isomorphisms between objects can be computed efficiently and the number of isomorphisms is exponentially smaller than the entire symmetric group. This provides the desired progress for the recursive algorithm.

The *Split-or-Johnson Routine* again builds on the Weisfeiler-Leman algorithm as well as further combinatorial tools. We omit any details here and refer the interested reader to [5, 47]. Overall, Babai's algorithm results in the following theorem.

Theorem 4.11 (Babai [6]) *The String Isomorphism Problem can be solved in time* $n^{\text{polylog}(n)}$.

4.4 Faster Certificates for Groups with Restricted Composition Factors

In the following two subsections we discuss extensions of Babai's algorithm for the String Isomorphism Problem. We start by considering again the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups. For such an input group, Babai's algorithm never encounters an obstacle case (assuming $d = \mathcal{O}(\log n)$) and thus, it simply follows Luks's algorithm which solves the problem in time $n^{\mathcal{O}(d)}$ (see also Corollary 4.6). However, in light of the novel group-theoretic subroutines employed in Babai's algorithm, it seems plausible to also hope for improvements in case the input group is a $\widehat{\Gamma}_d$-group.

Looking at Babai's algorithm from a high level, there are two major hurdles that one needs to overcome. First, we need to analyse the obstacle cases for Luks's algorithm, i.e., one needs to classify primitive $\widehat{\Gamma}_d$-groups of size larger than $n^{\mathcal{O}(\log d)}$. Without going into any details, let us just state that by a deep analysis of primitive $\widehat{\Gamma}_d$-groups [41], one can prove that such groups are, once again, essentially Johnson groups in a similar manner as in Theorem 4.7.

The second crucial hurdle is the adaptation of the Local Certificates Routine. Recall that Babai's algorithm applies the Local Certificates Routine to test sets T of size $|T| = t = \mathcal{O}(\log n)$. The routine runs in time $t!n^{\mathcal{O}(1)}$ and performs $t!n$ many recursive calls to the String Isomorphism Problem over domain size at most n/t. The central issue is that, in order to prove correctness building on the Unaffected Stabilizers Theorem, the Local Certificates Routine requires that $t > \max\{8, \log n + 2\}$. However, to apply the Local Certificates Routine in the setting of $\widehat{\Gamma}_d$-groups, we wish to choose $t = \mathcal{O}(\log d)$ to attain the desired running time. Hence, a main obstacle for a faster algorithm for $\widehat{\Gamma}_d$-groups is to obtain a suitable variant of the Unaffected Stabilizers Theorem. Unfortunately, it is not difficult to see that the natural variant of the Unaffected Stabilizers Theorem (where we restrict the input group to be in $\widehat{\Gamma}_d$ and update the bound on t as desired) does not hold (see, e.g., [70, Example 5.5.10]).

The main idea to circumvent this problem [41] is to first normalize the input to ensure suitable restrictions on the input group Γ that enable us to prove the desired variant of the Unaffected Stabilizers Theorem. In the following, we only describe the main idea for the normalization. Given a normalized input, one can provide a suitable variant of the Local Certificates Routine and, building on a more precise analysis of the recursion, extend Babai's algorithm to the setting of $\widehat{\Gamma}_d$-groups. This results in an algorithm solving the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups in time $n^{\text{polylog}(d)}$. We remark that the algorithm does not exploit the combinatorial subroutines directly, but one can simply rely on using Babai's algorithm as a black box instead. In particular, this also simplifies the analysis of the entire algorithm.

To describe the normalization procedure, we follow the framework developed in the second author's PhD thesis [70]. A *rooted tree* is a pair (T, v_0) where T is a directed tree and $v_0 \in V(T)$ is the root of T (all edges are directed away from

the root). Let $L(T)$ denote the set of leaves of T, i.e., vertices $v \in V(T)$ without outgoing edges. For $v \in V(T)$ we denote by T^v the subtree of T rooted at vertex v.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group. A *structure tree* for Γ is a rooted tree (T, v_0) such that $L(T) = \Omega$ and $\Gamma \leq (\mathrm{Aut}(T))[\Omega]$, i.e., all $\gamma \in \Gamma$ preserve the structure of T.

Lemma 4.12 *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a transitive group and (T, v_0) a structure tree for Γ. For every $v \in V(T)$ the set $L(T^v)$ is a block of Γ. Moreover, $\{L(T^w) \mid w \in v^{\mathrm{Aut}(T)}\}$ forms a block system of the group Γ.*

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a transitive group. The last lemma implies that every structure tree (T, v_0) gives a sequence of Γ-invariant partitions $\{\Omega\} = \mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k = \{\{\alpha\} \mid \alpha \in \Omega\}$ (for each level $h \geq 0$, the blocks associated with the vertices at distance h from the root form one of the partitions). On the other hand, every such sequence of partitions gives a structure tree (T, v_0) with

$$V(T) = \Omega \cup \bigcup_{i=0,\ldots,k-1} \mathfrak{B}_i$$

and

$$E(T) = \{(B, B') \mid B \in \mathfrak{B}_{i-1}, B' \in \mathfrak{B}_i, B' \subseteq B\} \cup \{(B, \alpha) \mid B \in \mathfrak{B}_{k-1}, \alpha \in B\}.$$

The root is $v_0 = \Omega$.

The following definition describes the desired structure of a normalized group. For $\mathfrak{B}$ a partition of Ω and $S \subseteq \Omega$ we denote by $\mathfrak{B}[S] := \{B \cap S \mid B \in \mathfrak{B}, B \cap S \neq \emptyset\}$ the induced partition on S.

Definition 4.13 (Almost d-ary Sequences of Partitions) Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a group and let $\{\Omega\} = \mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k = \{\{\alpha\} \mid \alpha \in \Omega\}$ be a sequence of Γ-invariant partitions. The sequence $\mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k$ is *almost d-ary* if for every $i \in [k]$ and $B \in \mathfrak{B}_{i-1}$ it holds that

1. $|\mathfrak{B}_i[B]| \leq d$, or

2. $\Gamma_B[\mathfrak{B}_i[B]]$ is semi-regular.

If the first option is always satisfied the sequence $\mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k$ is called *d-ary*. Similarly, a structure tree (T, v_0) for Γ is *(almost) d-ary* if the corresponding sequence of partitions is.

Let Γ be a group for which there is an almost d-ary structure tree (T, v_0). Now, we can execute Luks's algorithm along the given structure tree (T, v_0) (by always picking the next partition when performing standard Luks reduction). Intuitively speaking, this allows us to restrict the primitive groups that are encountered during Luks's algorithm. For a d-ary structure tree all primitive groups are subgroups of S_d. For an almost d-ary structure tree we additionally need to be able to handle semi-regular groups. However, handling such groups is simple since they have size at most n where n denotes the size of the permutation domain.

The goal of the *Normalization Routine* is, given an instance $(\Gamma, \mathfrak{x}, \mathfrak{y})$ of the String Isomorphism Problem where $\Gamma \in \widehat{\Gamma}_d$, to compute a normalized equivalent instance

$(\Gamma^*, (T, v_0), \mathfrak{x}^*, \mathfrak{y}^*)$ such that (T, v_0) forms an almost d-ary structure tree for Γ^*. The main tool to achieve the normalization are tree unfoldings of certain *structure graphs*.

Let (G, v_0) be a rooted acyclic directed graph. For $v \in V(G)$ define $N^+(v) := \{w \in V(G) \mid (v, w) \in E(G)\}$ to be the set of outgoing neighbors of v. The *forward degree* of v is $\deg^+(v) := |N^+(v)|$. A vertex is a *leaf* of G if it has no outgoing neighbors, i.e., $\deg^+(v) = 0$. Let $L(G) = \{v \in V(G) \mid \deg^+(v) = 0\}$ denote the set of leaves of G.

Let $\Gamma \le \mathrm{Sym}(\Omega)$ be a permutation group. A *structure graph* for Γ is a triple (G, v_0, φ) where (G, v_0) is a rooted acyclic directed graph such that $L(G) = \Omega$ and $\Gamma \le (\mathrm{Aut}(G))[\Omega]$ and $\varphi \colon \Gamma \to \mathrm{Aut}(G)$ is a homomorphism such that $(\gamma^\varphi)[\Omega] = \gamma$ for all $\gamma \in \Gamma$.

Note that each structure tree can be viewed as a structure graph (for trees the homomorphism φ is uniquely defined and can be easily computed). As indicated above the strategy to normalize the action is to consider the tree unfolding of a suitable structure graph. The permutation domain of the normalized action then corresponds to the leaves of the tree unfolding for which there is a natural action of the group Γ.

Let (G, v_0) be a rooted acyclic directed graph. A *branch* of (G, v_0) is a sequence $(v_0, v_1, \ldots, v_k)$ such that $(v_{i-1}, v_i) \in E(G)$ for all $i \in [k]$. A branch $(v_0, v_1, \ldots, v_k)$ is *maximal* if it cannot be extended to a longer branch, i.e., if v_k is a leaf of (G, v_0). Let $\mathrm{Br}(G, v_0)$ denote the set of branches of (G, v_0) and $\mathrm{Br}^*(G, v_0)$ denote the set of maximal branches. Note that $\mathrm{Br}^*(G, v_0) \subseteq \mathrm{Br}(G, v_0)$. Also, for a maximal branch $\bar{v} = (v_0, v_1, \ldots, v_k)$ let $L(\bar{v}) := v_k$. Note that $L(\bar{v}) \in L(G)$.

For a rooted acyclic directed graph (G, v_0) the *tree unfolding* of (G, v_0) is defined to be the rooted tree $\mathrm{Unf}(G, v_0)$ with vertex set $\mathrm{Br}(G, v_0)$ and edge set

$$E(\mathrm{Unf}(G, v_0)) = \{((v_0, \ldots, v_k), (v_0, \ldots, v_k, v_{k+1})) \mid (v_0, \ldots, v_{k+1}) \in \mathrm{Br}(G, v_0)\}.$$

Note that $L(\mathrm{Unf}(G, v_0)) = \mathrm{Br}^*(G, v_0)$, i.e., the leaves of the tree unfolding of (G, v_0) are exactly the maximal branches of (G, v_0).

Example 4.14 Let $m \le d$, $t \le \frac{m}{2}$ and consider the Johnson group $\Gamma = A_m^{(t)}$. Then a structure graph (G, v_0, φ) for $\Gamma \le \mathrm{Sym}(\binom{[m]}{t})$ can be constructed as follows. The vertices of the graph are all subsets of $[m]$ of size at most t, i.e.,

$$V(G) := \binom{[m]}{\le t} = \{X \subseteq [m] \mid |X| \le t\}.$$

Two vertices X and Y are connected by an edge if Y is the extension of X by a single element, i.e.,

$$E(G) := \{(X, Y) \mid X \subseteq Y \wedge |Y \setminus X| = 1\}.$$

The root of the structure graph is $v_0 := \emptyset$. Note that $L(G) = \binom{[m]}{t}$ as desired. Intuitively speaking, (G, v_0) corresponds to the first $t + 1$ levels of the subset lattice of the set $[m]$. An example is given in Figure 5.

For $\gamma \in A_m$ let $\gamma^{(t)}$ be the element obtained from the natural action of γ on $\binom{[m]}{t}$. So $\Gamma = \{\gamma^{(t)} \mid \gamma \in A_m\}$. Let $\varphi \colon \Gamma \to \mathrm{Aut}(G)$ be defined by $X^{\varphi(\gamma^{(t)})} = X^\gamma$ where $X \in V(G)$. Then (G, v_0, φ) is a structure graph for Γ.
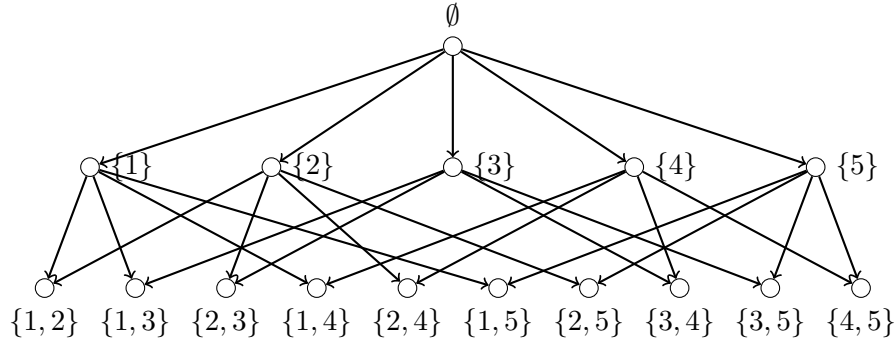
Figure 5: A structure graph for the Johnson group $A_5^{(2)}$

Now consider the tree unfolding $\mathrm{Unf}(G, v_0)$ of the graph (G, v_0). A maximal branch $\bar{v} \in \mathrm{Br}^*(G, v_0)$ is a sequence $\bar{v} = (X_0, X_1, \ldots, X_t)$ where $X_{i-1} \subseteq X_i$ and $|X_i| = i$ for all $i \in [t]$. This gives an ordering of the elements in X_t. Indeed, it is not difficult to see that there is a one-to-one correspondence between the maximal branches $\mathrm{Br}^*(G, v_0)$ and the set $[m]^{\langle t \rangle}$ of ordered t-tuples over the set $[m]$ with pairwise distinct elements.

The group A_m also acts naturally on the set $[m]^{\langle t \rangle}$. Let $A_m^{\langle t \rangle}$ be the permutation group obtained from this action. Now it can be observed that $\mathrm{Unf}(G, v_0)$ gives a structure tree for $A_m^{\langle t \rangle}$. Also, the degree of the permutation group $A_m^{\langle t \rangle}$ is only slightly larger than the degree of $A_m^{(t)}$. Indeed,

$$\left| \binom{[m]}{t} \right|^{\log m} = \binom{m}{t}^{\log m} \geq \left(\frac{m}{t} \right)^{t \log m} \geq 2^{t \log m} = m^t \geq |[m]^{\langle t \rangle}|. \tag{4.9}$$

In order to generalize this example to obtain a normalization for all groups one can first observe that a group Γ naturally acts on the set of maximal branches of a structure graph and additionally, the tree unfolding of the structure graph forms a structure tree for this action.

Lemma 4.15 *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group and let (G, v_0, φ) be a structure graph for Γ. Then there is an action $\psi \colon \Gamma \to \mathrm{Sym}(\mathrm{Br}^*(G))$ on the set of maximal branches of (G, v_0) such that*

   1. $L(\bar{v}^{\psi(\gamma)}) = (L(\bar{v}))^\gamma$ for all $\bar{v} \in \mathrm{Br}^*(G)$ and $\gamma \in \Gamma$, and

   2. $\mathrm{Unf}(G, v_0)$ forms a structure tree for Γ^ψ.

Moreover, given the group Γ and the structure graph (G, v_0, φ), the homomorphism ψ can be computed in time polynomial in $|\mathrm{Br}^*(G, v_0)|$.

Hence, for the normalization, the strategy is to first compute a suitable structure graph (G, v_0, φ) for Γ. Then we update the group by setting $\Gamma^* := \Gamma^\psi$ to be the *standard action on the set of maximal branches* described in Lemma 4.15. Here, the tree unfolding $\mathrm{Unf}(G, v_0)$ should form an almost d-ary structure tree for Γ^*. Towards this end, we say that (G, v_0, φ) is an *almost d-ary* structure graph for Γ if $\mathrm{Unf}(G, v_0)$ forms an almost d-ary structure tree for Γ^*. Moreover, we also update

the strings $\mathfrak{x}, \mathfrak{y} \colon \Omega \to \Sigma$ by setting $\mathfrak{x}^* \colon \mathrm{Br}^*(G) \to \Sigma \colon \bar{v} \mapsto \mathfrak{x}(L(\bar{v}))$ and similarly $\mathfrak{y}^* \colon \mathrm{Br}^*(G) \to \Sigma \colon \bar{v} \mapsto \mathfrak{y}(L(\bar{v}))$. One can easily show that

$$(\mathrm{Iso}_\Gamma(\mathfrak{x}, \mathfrak{y}))^\psi = \mathrm{Iso}_{\Gamma^*}(\mathfrak{x}^*, \mathfrak{y}^*).$$

Hence, to complete the normalization, it only remains to find an almost d-ary structure graph for which the number of maximal branches is not too large.

Theorem 4.16 ([41, 70]) *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a $\widehat{\Gamma}_d$-group. Then there is an almost d-ary structure graph (G, v_0, φ) for Γ such that $|\mathrm{Br}(G, v_0)| \leq n^{\mathcal{O}(\log d)}$. Moreover, there is an algorithm computing such a structure graph in time polynomial in the size of G.*

The proof is the theorem is far from trivial and builds on the characterization of large primitive $\widehat{\Gamma}_d$-groups (see [41]).

Corollary 4.17 *There is a Turing-reduction from the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups to the String Isomorphism Problem for groups equipped with an almost d-ary structure tree running in time $n^{\mathcal{O}(\log d)}$.*

As already indicated above, having normalized the input to the String Isomorphism Problem, one can prove an adaptation of the Unaffected Stabilizers Theorem and extend Babai's algorithm to the setting of $\widehat{\Gamma}_d$-groups.

Theorem 4.18 (Grohe, Neuen and Schweitzer [41]) *The String Isomorphism Problem for $\widehat{\Gamma}_d$-groups can be solved in time $n^{\mathrm{polylog}(d)}$.*

Recall that there is a polynomial-time Turing reduction from the Graph Isomorphism Problem for graphs of maximum degree d to the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups [61, 11]. Hence, the Graph Isomorphism Problem for graphs of maximum degree d can be solved in time $n^{\mathrm{polylog}(d)}$. Actually, we reprove this result in Section 5 using slightly different tools.

4.5   From Strings to Hypergraphs

Next, we present another extension of the presented methods taking more general input structures into account. Specifically, up to this point, we only considered strings for the input structures which are quite restrictive. In this subsection, we briefly describe how to extend the methods to isomorphism testing for hypergraphs.

A hypergraph is a pair $\mathcal{H} = (V, \mathcal{E})$ where V is a finite vertex set and $\mathcal{E} \subseteq 2^V$ (where 2^V denotes the power set of V). As for strings, we are interested in the isomorphism problem between hypergraphs where we are additionally given a permutation group that restricts possible isomorphisms between the two given hypergraphs. More precisely, in this work, the *Hypergraph Isomorphism Problem* takes as input two hypergraphs $\mathcal{H}_1 = (V, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V, \mathcal{E}_2)$ over the same vertex set and a permutation group $\Gamma \leq \mathrm{Sym}(V)$ (given by a set of generators), and asks whether is some $\gamma \in \Gamma$ such that $\gamma \colon \mathcal{H}_1 \cong \mathcal{H}_2$ (i.e., γ is an isomorphism from $\mathcal{H}_1$ to $\mathcal{H}_2$).

For the purpose of designing an algorithm it is actually more convenient to consider the following equivalent problem. The *Set-of-Strings Isomorphism Problem* takes as input two sets $\mathfrak{X} = \{\mathfrak{x}_1, \ldots, \mathfrak{x}_m\}$ and $\mathfrak{Y} = \{\mathfrak{y}_1, \ldots, \mathfrak{y}_m\}$ where $\mathfrak{x}_i, \mathfrak{y}_i \colon \Omega \to \Sigma$

are strings, and a group $\Gamma \leq \mathrm{Sym}(\Omega)$, and asks whether there is some $\gamma \in \Gamma$ such that $\mathfrak{X}^\gamma := \{\mathfrak{r}_1^\gamma, \ldots, \mathfrak{r}_m^\gamma\} = \mathfrak{Y}$.

Theorem 4.19 *The Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups is polynomial-time equivalent to the Set-of-Strings Isomorphism Problem for $\widehat{\Gamma}_d$-groups under many-one reductions.*

Proof Sketch A hypergraph $\mathcal{H} = (V, \mathcal{E})$ can be translated into a set $\mathfrak{X} := \{\mathfrak{r}_E \mid E \in \mathcal{E}\}$ over domain $\Omega := V$ where $\mathfrak{r}_E$ is the characteristic function of E, i.e., $\mathfrak{r}(v) = 1$ if $v \in E$ and $\mathfrak{r}(v) = 0$ if $v \notin E$.

In the other direction, a set of strings $\mathfrak{X}$ over domain Ω and alphabet Σ can be translated into a hypergraph $\mathcal{H} = (V, \mathcal{E})$ where $V := \Omega \times \Sigma$ and $\mathcal{E} := \{E_\mathfrak{r} \mid \mathfrak{r} \in \mathfrak{X}\}$ where $E_\mathfrak{r} := \{(\alpha, \mathfrak{r}(\alpha)) \mid \alpha \in \Omega)\}$. The group $\Gamma \leq \mathrm{Sym}(\Omega)$ is translated into the natural action of Γ on V defined via $(\alpha, a)^\gamma := (\alpha^\gamma, a)$.

It is easy to verify that both translations preserve isomorphisms. $\square$

For the purpose of building a recursive algorithm, we consider a slightly different problem that crucially allows us to modify instances in a certain way.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a group and let $\mathfrak{P}$ be a Γ-invariant partition of the set Ω. A $\mathfrak{P}$-*string* is a pair $(P, \mathfrak{r})$ where $P \in \mathfrak{P}$ and $\mathfrak{r} \colon P \to \Sigma$ is a string over a finite alphabet Σ. For $\sigma \in \mathrm{Sym}(\Omega)$ the string $\mathfrak{r}^\sigma$ is defined by $\mathfrak{r}^\sigma \colon P^\sigma \to \Sigma \colon \alpha \mapsto \mathfrak{r}(\alpha^{\sigma^{-1}})$. A permutation $\sigma \in \mathrm{Sym}(\Omega)$ is a Γ-*isomorphism* from $(P, \mathfrak{r})$ to a second $\mathfrak{P}$-string $(Q, \mathfrak{h})$ if $\sigma \in \Gamma$ and $(P^\sigma, \mathfrak{r}^\sigma) = (Q, \mathfrak{h})$.

The *Generalized String Isomorphism Problem* takes as input a permutation group $\Gamma \leq \mathrm{Sym}(\Omega)$, a Γ-invariant partition $\mathfrak{P}$ of Ω, and $\mathfrak{P}$-strings $(P_1, \mathfrak{r}_1), \ldots, (P_m, \mathfrak{r}_m)$ and $(Q_1, \mathfrak{h}_1), \ldots, (Q_m, \mathfrak{h}_m)$, and asks whether there is some $\gamma \in \Gamma$ such that

$$\{(P_1^\gamma, \mathfrak{r}_1^\gamma), \ldots, (P_m^\gamma, \mathfrak{r}_m^\gamma)\} = \{(Q_1, \mathfrak{h}_1), \ldots, (Q_m, \mathfrak{h}_m)\}.$$

We denote $\mathfrak{X} = \{(P_1, \mathfrak{r}_1), \ldots, (P_m, \mathfrak{r}_m)\}$ and $\mathfrak{Y} = \{(Q_1, \mathfrak{h}_1), \ldots, (Q_m, \mathfrak{h}_m)\}$. Additionally, $\mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{Y})$ denotes the set of Γ-isomorphisms from $\mathfrak{X}$ to $\mathfrak{Y}$ and $\mathrm{Aut}_\Gamma(\mathfrak{X}) := \mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{X})$.

It is easy to see that the Set-of-Strings Isomorphism Problem forms a special case of the Generalized String Isomorphism Problem where $\mathfrak{P}$ is the trivial partition consisting of one block.

For the rest of this subsection we denote by $n := |\Omega|$ the size of the domain, and m denotes the size of $\mathfrak{X}$ and $\mathfrak{Y}$ (we always assume $|\mathfrak{X}| = |\mathfrak{Y}|$, otherwise the problem is trivial). The goal is to sketch an algorithm that solves the Generalized String Isomorphism Problem for $\widehat{\Gamma}_d$-groups in time $(n + m)^{\mathrm{polylog}(d)}$.

As a starting point it was already observed by Miller [67] that Luks's algorithm can be easily extended to hypergraphs resulting in an isomorphism test running in time $(n + m)^{\mathcal{O}(d)}$ for $\widehat{\Gamma}_d$-groups. Similar to the previous subsection, the main obstacle we are facing to obtain a more efficient algorithm is the adaptation of the Local Certificates Routine.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a $\widehat{\Gamma}_d$-group, $\mathfrak{P}$ a Γ-invariant partition of Ω, and $\mathfrak{X}, \mathfrak{Y}$ two sets of $\mathfrak{P}$-strings. Recall that, in a nutshell, the Local Certificates Routine considers a Γ-invariant window $W \subseteq \Omega$ such that $\Gamma[W] \leq \mathrm{Aut}(\mathfrak{X}[W])$ (i.e., the group Γ respects $\mathfrak{X}$ restricted to the window W) and aims at creating automorphisms of the

entire structure $\mathfrak{X}$ (from the local information that Γ respects $\mathfrak{X}$ on the window W). In order to create these automorphisms the Local Certificates Routine considers the group $\Gamma_{(\Omega \setminus W)}$ fixing every point outside of W. Remember that the Unaffected Stabilizers Theorem (resp. the variant suitable for $\widehat{\Gamma}_d$-groups) guarantees that the group $\Gamma_{(\Omega \setminus W)}$ is large.

For the String Isomorphism Problem it is easy to see that $\Gamma_{(\Omega \setminus W)}$ consists only of automorphisms of the input string (assuming Γ respects the input string $\mathfrak{x}$ on the window W) since there are no dependencies between the positions within the window W and outside of W. However, for the Generalized String Isomorphism Problem, this is not true anymore. As a simple example, suppose the input is a graph on vertex set Ω (which, in particular, can be interpreted as a hypergraph and translated into a set of strings) and the edges between W and $\Omega \setminus W$ form a perfect matching. Then a permutation $\gamma \in \Gamma_{(\Omega \setminus W)}$ is not necessarily an automorphism of G even if it respects $G[W]$, since it may not preserve the edges between W and $\Omega \setminus W$. Actually, since the edges between W and $\Omega \setminus W$ form a perfect matching, the only automorphism of G in the group $\Gamma_{(\Omega \setminus W)}$ is the identity mapping.

In other words, in order to compute $\mathrm{Aut}(\mathfrak{X})$, it is not possible to consider $\mathfrak{X}[W]$ and $\mathfrak{X}[\Omega \setminus W]$ independently as is done by the Local Certificates Routine.

The solution to this problem is guided by the following simple observation. Suppose that $\mathfrak{X}[W]$ is *simple*, i.e.,

$$m_{\mathfrak{X}[W]}(P) := |\{\mathfrak{x}[W \cap P] \mid (P, \mathfrak{x}) \in \mathfrak{X}, W \cap P \neq \emptyset\}| = 1$$

for all $P \in \mathfrak{P}[W]$ (in other words, when restricted to W, each block P only contains one string). In this case it is possible to consider $\mathfrak{X}[W]$ and $\mathfrak{X}[\Omega \setminus W]$ independently since there can be no more additional dependencies between W and $\Omega \setminus W$.

Let $W \subseteq \Omega$ be a Γ-invariant set such that $\Gamma[W] \leq \mathrm{Aut}(\mathfrak{X}[W])$ (this is the case during the Local Certificates Routine). In order to solve the problem described above in general, the basic idea is to introduce another normalization procedure, referred to as *Simplification Routine*, modifying the instance in such a way that $\mathfrak{X}[W]$ becomes simple. Eventually, this allows us to extend the Local Certificates Routine to the setting of the Generalized String Isomorphism Problem.

In the following we briefly describe the *Simplification Routine* which exploits the specific definition of the Generalized String Isomorphism Problem. Consider a set $P \in \mathfrak{P}$. In order to "simplify" the instance we define an equivalence relation on the set $\mathfrak{X}[[P]] := \{\mathfrak{x} \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$ of all strings on P. Two $\mathfrak{P}$-strings $(P, \mathfrak{x}_1)$ and $(P, \mathfrak{x}_2)$ are W-equivalent if they are identical on the window W, i.e., $\mathfrak{x}_1[W \cap P] = \mathfrak{x}_2[W \cap P]$. For each equivalence class we create a new block P' containing exactly the strings from the equivalence class. Since the group Γ respects the induced sub-instance $\mathfrak{X}[W]$ it naturally acts on the equivalence classes. This process is visualized in Figure 6 and formalized below.

Let $\Omega' := \bigcup_{P \in \mathfrak{P}} P \times \{\mathfrak{x}[W \cap P] \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$ and $\mathfrak{P}' := \{P \times \{\mathfrak{x}[W \cap P]\} \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$. Also define

$$\mathfrak{X}' := \left\{ \left(P \times \{\mathfrak{x}[W \cap P]\}, \mathfrak{x}' \right) \ \middle| \ (P, \mathfrak{x}) \in \mathfrak{X} \right\}$$

where $\mathfrak{x}' \colon P \times \{\mathfrak{x}[W \cap P]\} \to \Sigma \colon (\alpha, \mathfrak{x}[W \cap P]) \mapsto \mathfrak{x}(\alpha)$. The set $\mathfrak{Y}'$ is defined similarly for the instance $\mathfrak{Y}$. Note that $\mathfrak{X}'$ and $\mathfrak{Y}'$ are sets of $\mathfrak{P}'$-strings. Finally, the group Γ faithfully acts on the set Ω' via $(\alpha, \mathfrak{z})^\gamma = (\alpha^\gamma, \mathfrak{z}^\gamma)$ yielding an injective
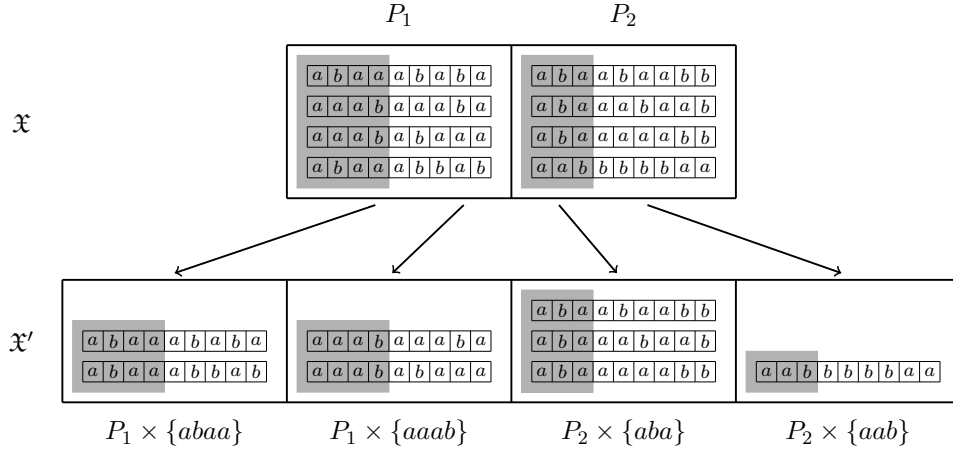
Figure 6: A set $\mathfrak{X}$ of $\mathfrak{P}$-strings is given in the top and the "simplified" instance $\mathfrak{X}'$ is given below. The window W is marked in gray. Note that $\mathfrak{X}'[W']$ is simple where W' denotes the window marked in gray in the bottom part of the figure.

homomorphism $\psi \colon \Gamma \to \mathrm{Sym}(\Omega')$. Define $\Gamma' \coloneqq \Gamma^{\psi}$. It can be easily checked that the updated instance is equivalent to the original instance. Also, $\mathfrak{X}'[W']$ is simple where $W' \coloneqq \{(\alpha, \mathfrak{z}) \in \Omega' \mid \alpha \in W\}$.

While this simplification allows us to treat $\mathfrak{X}'[W']$ and $\mathfrak{X}'[W' \setminus \Omega']$ independently and thus solves the above problem, it creates several additional issues that need to be addressed. First, this modification may destroy the normalization property (i.e., the existence of an almost d-ary sequence of partitions). As a result, the Local Certificates Routine constantly needs to re-normalize the input instances which requires a precise analysis of the increase in size occurring from the re-normalization. In turn, the re-normalization of instances creates another problem. In the *Aggregation of Local Certificates* (where the local certificates are combined into a relational structure), the outputs of the Local Certificates Routine are compared with each other requiring the outputs to be isomorphism-invariant. However, the re-normalization procedure is not isomorphism-invariant. The solution to this problem is to run the Local Certificates Routine in parallel on all pairs of test sets compared later on. This way, one can ensure that all instances are normalized in the same way.

Overall, these ideas allow us to obtain the following theorem.

Theorem 4.20 (Neuen [71]) *The Generalized String Isomorphism Problem for $\widehat{\Gamma}_d$-groups can be solved in time $(n + m)^{\mathrm{polylog}(d)}$ where n denotes the size of the domain and m the number of strings in the input sets.*

Note that this gives an algorithm for the Set-of-Strings Isomorphism Problem as well as the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups with the same running time. In particular, the Hypergraph Isomorphism Problem without any input group can be solved in time $(n + m)^{\mathrm{polylog}(n)}$. This is the fasted known algorithm for the problem. A different algorithm with the same running time was obtained by Wiebking [87].
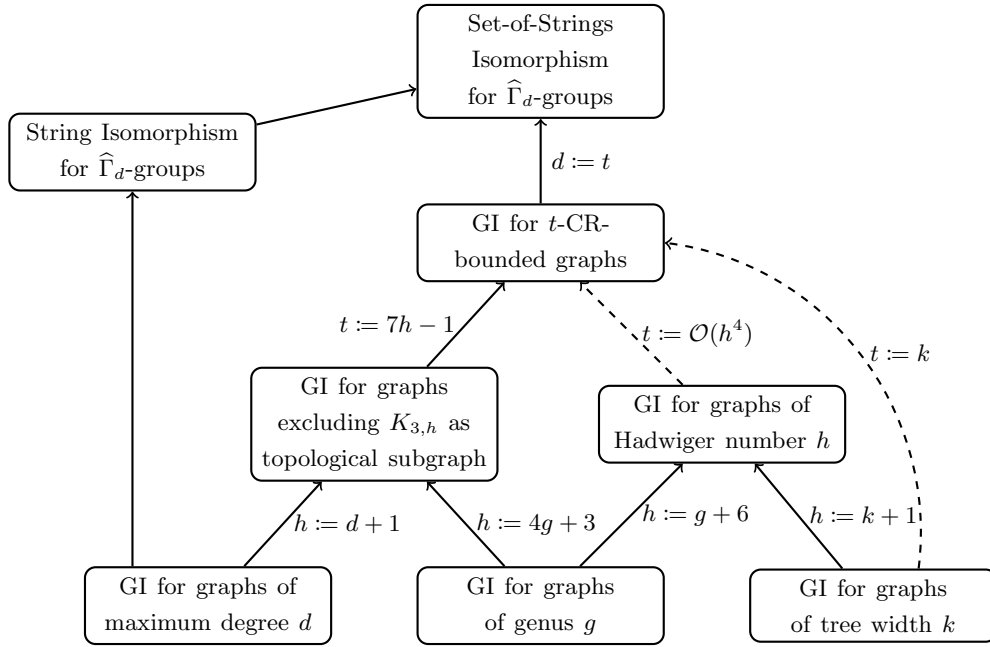
Figure 7: Dependencies between isomorphism problems considered in this paper. Each solid edge represents a polynomial-time Turing reduction where the parameter is set according to the edge label. The dashed edges represent fpt reductions.

5   Quasi-Polynomial Parameterized Algorithms for Isomorphism Testing

In this section we present several applications of the results presented above for isomorphism testing on restricted classes of graphs. Towards this end, we first introduce the notion of *t-CR-bounded* graphs recently introduced in [71] and build an isomorphism test for such graphs based on Theorem 4.20. It turns out that the notion of t-CR-bounded graphs forms a powerful tool when it comes to the task of designing isomorphism tests for restricted classes of graphs. In this direction we build a series of reductions for the isomorphism problem for well-known parameterized classes of graphs to the isomorphism problem for t-CR-bounded graphs leading to the most efficient algorithms for isomorphism testing for mentioned classes. An overview on the reductions can be found in Figure 7.

5.1   Allowing Color Refinement to Split Small Classes

As indicated above, the central tool for designing more efficient isomorphism tests based on the results of the previous section is the notion of t-CR-bounded graphs. Intuitively speaking, a vertex-colored graph (G, χ) is *t-CR-bounded* if the vertex-coloring can be transformed into a discrete coloring (i.e., a coloring where each vertex has its own color) by repeatedly applying the following operations:

1. applying the Color Refinement algorithm, and

2. picking a color class $[v]_\chi := \{w \in V(G) \mid \chi(w) = \chi(v)\}$ for some vertex v such that $|[v]_\chi| \leq t$ and assigning each vertex in the class its own color.

The next definition formalizes this intuition. For formal reason, it turns out to be more convenient to consider vertex- and arc-colored graphs.

Definition 5.1 A vertex- and arc-colored graph (G, χ_V, χ_E) is t-*CR-bounded* if the sequence $(\chi_i)_{i \geq 0}$ reaches a discrete coloring where $\chi_0 := \chi_V$,

$$\chi_{2i+1} := \chi_{(\infty)}(G, \chi_{2i}, \chi_E)$$

and

$$\chi_{2i+2}(v) := \begin{cases} (v, 1) & \text{if } |[v]_{\chi_{2i+1}}| \leq t \\ (\chi_{2i+1}(v), 0) & \text{otherwise} \end{cases}$$

for all $i \geq 0$.

Also, for the minimal $i_\infty \geq 0$ such that $\chi_{i_\infty} \equiv \chi_{i_\infty + 1}$, we refer to χ_{i_∞} as the t-*CR-stable* coloring of G and denote it by $\chi_{t\text{-}\mathsf{CR}}(G)$.

In order to give an efficient isomorphism test for t-CR-bounded graphs we present a polynomial-time reduction to the Set-of-Strings Isomorphism Problem discussed in Section 4.5.

Theorem 5.2 ([71]) *There is a polynomial-time Turing reduction from the Graph Isomorphism Problem for t-CR-bounded graphs to the Set-of-Strings Isomorphism Problem for $\widehat{\Gamma}_t$-groups.*

Proof Sketch Let G be a t-CR-bounded graph and let $(\chi_i)_{i \geq 0}$ be the corresponding sequence of colorings. For simplicity we assume the arc-coloring of G is trivial and restrict ourselves to computing a generating set for the automorphism group $\mathrm{Aut}(G)$. Let $\mathfrak{P}_i := \pi(\chi_i)$ be the partition of the vertex set into the color classes of the coloring χ_i. Observe that $\mathfrak{P}_i$ is defined in an isomorphism-invariant manner with respect to G and $\mathfrak{P}_{i+1} \preceq \mathfrak{P}_i$ for all $i \geq 0$.

The idea for the algorithm is to iteratively compute $\widehat{\Gamma}_t$-groups $\Gamma_i \leq \mathrm{Sym}(\mathfrak{P}_i)$ such that $(\mathrm{Aut}(G))[\mathfrak{P}_i] \leq \Gamma_i$ for all $i \geq 0$. First note that this gives an algorithm for computing the automorphism group of G. Since G is t-CR-bounded there is some $i_\infty \geq 0$ such that $\mathfrak{P}_{i_\infty}$ is the discrete partition. Hence, $\mathrm{Aut}(G) \leq \Gamma_{i_\infty}$[3] where Γ_{i_∞} is a $\widehat{\Gamma}_t$-group and the automorphism group of G can be computed using Theorems 4.19 and 4.20.

For the sequence of groups the algorithm sets $\Gamma_0 := \{\mathrm{id}\}$ to be the trivial group (this is correct since χ_0 is the vertex-coloring of the input graph G). Hence, let $i > 0$ and suppose the algorithm already computed a generating set for Γ_{i-1}.

If i is even $\mathfrak{P}_i$ is obtained from $\mathfrak{P}_{i-1}$ by splitting all sets $P \in \mathfrak{P}_{i-1}$ of size at most t. Towards this end, the algorithm first updates $\Gamma'_{i-1} := \mathrm{Aut}_{\Gamma_{i-1}}(\{\mathfrak{x}_{i-1}\})$ where $\mathfrak{x}_{i-1}(P) = |P|$ for all $P \in \mathfrak{P}_{i-1}$. Then, for each orbit $\mathfrak{B}$ of Γ'_{i-1} there is some number $p \in \mathbb{N}$ such that $|P| = p$ for all $P \in \mathfrak{B}$. Now Γ_i is obtained from Γ'_{i-1} by taking the wreath product with the symmetric group S_p for all orbits where $p \leq t$. Clearly, Γ_i is still a $\widehat{\Gamma}_t$-group.

---

[3]Formally, this is not correct since $\mathrm{Aut}(G) \leq \mathrm{Sym}(V(G))$ and $\Gamma_{i_\infty} \leq \mathrm{Sym}(\{\{v\} \mid v \in V(G)\})$. However, the algorithm can simply identify v with the singleton set $\{v\}$ to obtain the desired supergroup.

If i is odd $\mathfrak{P}_i$ is obtained from $\mathfrak{P}_{i-1}$ by performing the Color Refinement algorithm. Consider a single iteration of the Color Refinement algorithm. More precisely, let $\mathfrak{P}$ be an invariant partition (with respect to $\mathrm{Aut}(G)$) and $\Gamma \leq \mathrm{Sym}(\mathfrak{P})$ be a $\widehat{\Gamma}_t$-group such that $(\mathrm{Aut}(G))[\mathfrak{P}] \leq \Gamma$. Let $\mathfrak{P}'$ be the partition obtained from $\mathfrak{P}$ by performing a single iteration of the Color Refinement algorithm. We argue how to compute a $\widehat{\Gamma}_t$-group $\Gamma' \leq \mathrm{Sym}(\mathfrak{P}')$ such that $(\mathrm{Aut}(G))[\mathfrak{P}'] \leq \Gamma'$. Repeating this procedure for all iterations of the Color Refinement algorithm then gives the desired group Γ_i.

In order to compute the group Γ' consider the following collection of strings. For every $v \in V(G)$ define $\mathfrak{x}_v \colon \mathfrak{P} \to \mathbb{N}^2$ via $\mathfrak{x}_v(P) = (1, |N(v) \cap P|)$ if $v \in P$ and $\mathfrak{x}_v(P) = (0, |N(v) \cap P|)$ otherwise. Observe that, by the Definition of the Color Refinement algorithm, $\mathfrak{x}_v = \mathfrak{x}_w$ if and only if there is some $P' \in \mathfrak{P}'$ such that $v, w \in P'$. Hence, there is a natural one-to-one correspondence between $\mathfrak{X} \coloneqq \{\mathfrak{x}_v \mid v \in V(G)\}$ and $\mathfrak{P}'$. Now define Γ' be the induced action of $\mathrm{Aut}_\Gamma(\mathfrak{X})$ on the set $\mathfrak{P}'$ (obtained from this correspondence). It is easy to verify that $(\mathrm{Aut}(G))[\mathfrak{P}'] \leq \Gamma'$.

Clearly, all steps of the algorithm can be performed in polynomial time using an oracle to the Set-of-Strings Isomorphism Problem for $\widehat{\Gamma}_t$-groups. $\qquad\square$

We remark that the proof of the last theorem also implies that $\mathrm{Aut}(G) \in \widehat{\Gamma}_t$ for every t-CR-bounded graph G. In combination with Theorem 4.20 we obtain an efficient isomorphism test for t-CR-bounded graphs.

Corollary 5.3 *The Graph Isomorphism Problem for t-CR-bounded graphs can be solved in time $n^{\mathrm{polylog}(t)}$.*

For the remainder of this section we shall exploit the algorithm from the corollary to design efficient isomorphism tests for a number of graph classes. Towards this end, we typically build on another standard tool for isomorphism testing which is individualization of single vertices. Intuitively, this allows us to break potential regularities in the input graph (for example, on a d-regular graph, the t-CR-stable coloring achieves no refinement unless $t \geq n$) and identify a "starting point" for analyzing the t-CR-stable coloring.

Let G be a graph and let $X \subseteq V(G)$ be a set of vertices. Let $\chi_V^* \colon V(G) \to C$ be the vertex-coloring obtained from individualizing all vertices in the set X, i.e., $\chi_V^*(v) \coloneqq (v, 1)$ for $v \in X$ and $\chi_V^*(v) \coloneqq (0, 0)$ for $v \in V(G) \setminus X$. Let $\chi \coloneqq \chi_{t\text{-}\mathsf{CR}}(G, \chi_V^*)$ denote the t-CR-stable coloring with respect to the input graph (G, χ_V^*). We define the *t-closure* of the set X (with respect to G) to be the set

$$\mathrm{cl}_t^G(X) \coloneqq \{v \in V(G) \mid |[v]_\chi| = 1\}.$$

Observe that $X \subseteq \mathrm{cl}_t^G(X)$. For $v_1, \ldots, v_\ell$ we use $\mathrm{cl}_t^G(v_1, \ldots, v_\ell)$ as a shorthand for $\mathrm{cl}_t^G(\{v_1, \ldots, v_\ell\})$

If $\mathrm{cl}_t^G(X) = V(G)$ then there is an isomorphism test for G running in time $n^{|X| + \mathrm{polylog}(t)}$. An algorithm first individualizes all vertices from X creating $n^{|X|}$ many instances of GI for t-CR-bounded graphs each of which can be solved using Corollary 5.3. This provides us a generic and powerful method for obtaining polynomial-time isomorphism tests for various classes of graphs. As a first, simple example we argue that isomorphism for graphs of maximum degree d can be tackled this way.

Theorem 5.4 *Let G be a connected graph of maximum degree d and let $v \in V(G)$. Then $\mathrm{cl}_d^G(v) = V(G)$.*

In particular, there is a polynomial-time Turing reduction from the Graph Isomorphism Problem for graphs of maximum degree d to the Graph Isomorphism Problem for d-CR-bounded graphs.

Proof   Let $\chi := \chi_{d\text{-CR}}(G, \chi_V^*)$ denote the d-CR-stable coloring where χ_V^* is the coloring obtained from individualizing v. For $i \geq 0$ let $V_i := \{w \in V(G) \mid \mathrm{dist}_G(v, w) \leq i\}$. We prove by induction on $i \geq 0$ that $V_i \subseteq \mathrm{cl}_d^G(v)$. Since $V_n = V(G)$ this implies that $\mathrm{cl}_d^G(v) = V(G)$.

The base case $i = 0$ is trivial since $V_0 = \{v\} \subseteq \mathrm{cl}_d^G(v)$ as already observed above. So suppose $i \geq 0$ and let $w \in V_{i+1}$. Then there is some $u \in V_i$ such that $uw \in E(G)$. Moreover, $u \in \mathrm{cl}_d^G(v)$ by the induction hypothesis. This means that $|[u]_\chi| = 1$. Since χ is stable with respect to the Color Refinement algorithm $[w]_\chi \subseteq N(u)$. So $|[w]_\chi| \leq \deg(u) \leq d$. Hence, $|[w]_\chi| = 1$ because χ is d-CR-stable. $\square$

This gives an isomorphism test for graphs of maximum degree d running in time $n^{\mathrm{polylog}(d)}$. Observe that the algorithm is obtained via a reduction to the Set-of-Strings Isomorphism Problem for $\widehat{\Gamma}_d$-groups. We remark that, using similar, but slightly more involved ideas, there is also a polynomial-time Turing reduction from the isomorphism problem for graphs of maximum degree d to the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups [61, 11] (see also Section 4.2). Recall that the String Isomorphism Problem is a simpler problem which can be seen as a special case of the Set-of-Strings Isomorphism Problem (where the set of strings contains only one element). Indeed, the original $n^{\mathrm{polylog}(d)}$ isomorphism test for graphs of maximum degree d from [41] is obtained via this route. However, considering the remaining applications of the isomorphism test for t-CR-bounded graphs in this section, such a behaviour seems to be an exception and reductions to the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups are usually not known. This highlights the significance of the Set-of-Strings Isomorphism Problem in comparison to the String Isomorphism Problem.

5.2   Graphs of Small Genus

Next, we turn to the isomorphism problem for graphs of bounded Euler genus. Recall that a graph has Euler genus at most g if it can be embedded on a surface of Euler genus g. We omit a formal definition of the genus of a graph and instead only rely on the following basic property. A graph H is a *minor* of graph G if H can be obtained from G by deleting vertices and edges as well as contracting edges. A graph G *excludes H as a minor* if it has no minor isomorphic to H. It is well known that graphs of Euler genus g exclude $K_{3,4g+3}$ (the complete bipartite graph with 3 vertices on the left and $4g + 3$ vertices on the right) as a minor [78]. The next lemma connects graphs that exclude $K_{3,h}$ as a minor to t-CR-bounded graphs.

Recall that a graph G is *3-connected* if $G - X$ is connected for every set $X \subseteq V(G)$ of size $|X| \leq 2$.

Lemma 5.5 *Let (G, χ) be a 3-connected, vertex-colored graph that excludes $K_{3,h}$ as a minor and suppose $V_1 \uplus V_2 = V(G)$ such that*
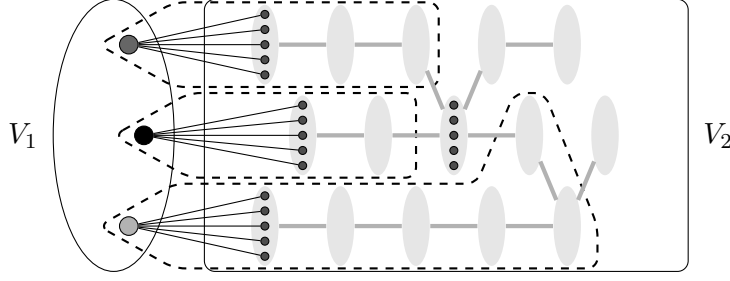
Figure 8: Visualization of the spanning tree T described in the proof of Lemma 5.5. Contracting the dashed regions into single vertices gives a subgraph isomorphic to $K_{3,h}$.

1. $|[v]_\chi| = 1$ for all $v \in V_1$,

2. χ is stable with respect to the Color Refinement algorithm, and

3. $|V_1| \geq 3$.

Then there exists $u \in V_2$ such that $|[u]_\chi| \leq h - 1$.

Proof Let $C := \chi(V(G))$, $C_1 := \chi(V_1)$ and $C_2 := \chi(V_2)$. Also define H to be the graph with vertex set $V(H) := C$ and edge set

$$E(H) = \{c_1 c_2 \mid \exists v_1 \in \chi^{-1}(c_1), v_2 \in \chi^{-1}(c_2) \colon v_1 v_2 \in E(G)\}.$$

Let $C' \subseteq C_2$ be the vertex set of a connected component of $H[C_2]$. Then $|N_H(C')| \geq 3$ since each $v \in V_1$ forms a singleton color class with respect to χ and G is 3-connected.

Now let $c_1, c_2, c_3 \in N_H(C')$ be distinct and also let $v_i \in \chi^{-1}(c_i)$ for $i \in [3]$. Also let T be a spanning tree of $H[C' \cup \{c_1, c_2, c_3\}]$ such that $c_1, c_2, c_3 \in L(T)$ where $L(T)$ denotes the set of leaves of T (see Figure 8). Moreover, let T' be the subtree of T obtained from repeatedly removing all leaves $c \in C'$. Hence, $L(T') = \{c_1, c_2, c_3\}$. Then there is a unique color c such that $\deg_{T'}(c) = 3$. Also, for $i \in [3]$, define C'_i to be the set of internal vertices on the unique path from c_i to c in the tree T'.

Since $|\chi^{-1}(c_i)| = 1$ and χ is stable with respect to the Color Refinement algorithm it holds that

$$G\left[\chi^{-1}(C'_i \cup \{c_i\})\right]$$

is connected. Let $U_i := \chi^{-1}(C'_i \cup \{c_i\})$, $i \in [3]$. Also let $U = \chi^{-1}(c)$ and suppose that $|U| \geq h$. Then $N(U_i) \cap U \neq \emptyset$ by the definition of the tree T. Moreover, this implies $U \subseteq N(U_i)$ since χ is stable with respect to the Color Refinement algorithm. Hence, G contains a minor isomorphic to $K_{3,h}$. $\square$

Corollary 5.6 ([71]) Let (G, χ_V, χ_E) be a 3-connected, vertex- and arc-colored graph that excludes $K_{3,h}$ as a minor and let $v_1, v_2, v_3 \in V(G)$ be distinct vertices. Then $\mathrm{cl}^G_{h-1}(v_1, v_2, v_3) = V(G)$.

Proof Let χ_V^* be the coloring obtained from χ_V after individualizing v_1, v_2 an v_3. Also let $\chi := \chi_{(h-1)\text{-}\mathsf{CR}}(G, \chi_V^*, \chi_E)$ be the $(h-1)$-CR-stable coloring with respect to the graph (G, χ_V^*, χ_E).

Suppose towards a contradiction that χ is not discrete (i.e., not every color class is a singleton). Let $V_2 := \{v \in V(G) \mid |[v]_\chi| > 1\}$ and let $V_1 := N_G(V_2)$. Then $|V_1| \geq 3$ since $|V(G) \setminus V_2| \geq 3$ and G is 3-connected. Also note that $\chi|_{V_1 \cup V_2}$ is a stable coloring for the graph $G[V_1 \cup V_2]$. Hence, by Lemma 5.5, there is some vertex $u \in V_2$ such that $|[u]_\chi| \leq h - 1$. Also $|[u]_\chi| > 1$ by the definition of V_2. But this contradicts the definition of an $(h-1)$-CR-stable coloring. □

Note that Corollary 5.6 only provides an isomorphism test for 3-connected graphs excluding $K_{3,h}$ as a minor. We can easily remedy this problem by using the well-known fact that, for a minor-closed class of graph, it suffices to solve the isomorphism problem on vertex- and arc-colored 3-connected graphs exploiting the standard decomposition into 3-connected components [48]. This gives us an isomorphism test for graphs of Euler genus at most g running in time $n^{\text{polylog}(g)}$.

5.3 Graphs of Small Tree Width

After discussing two fairly simple applications of the isomorphism test for t-CR-bounded graphs we now turn to the first more involved application considering graphs of small tree width. In the following subsection we will extend some of the basic ideas discussed for graphs of small tree width further to deal with the more general case of excluding the complete graph K_h as a minor.

We assume that the reader is familiar with the notion of tree width (see, e.g., [25, Chapter 12], [23, Chapter 7]). For the sake of completeness, let us nevertheless give the basic definitions. Let G be a graph. A *tree decomposition* of G is a pair (T, β) where T is a rooted tree and $\beta \colon V(T) \to 2^{V(G)}$, where $2^{V(G)}$ denotes the powerset of $V(G)$, such that

(T.1) for every edge $vw \in E(G)$ there is some $t \in V(T)$ such that $v, w \in \beta(t)$, and

(T.2) for every $v \in V(G)$ the set $\beta^{-1}(v) := \{t \in V(T) \mid v \in \beta(t)\}$ is non-empty and connected in T, that is, the induced subgraph $T[\beta^{-1}(v)]$ is a subtree.

The sets $\beta(t)$, $t \in V(T)$, are the *bags* of the decomposition. Also, the sets $\beta(t) \cap \beta(s)$, $st \in E(T)$, are the *adhesion sets* of the decomposition. The *width* of a tree decomposition is defined as the maximum bag size minus one, i.e., $\text{wd}(T, \beta) := \max_{t \in V(T)} |\beta(t)| - 1$. The *tree width* of a graph G, denoted $\text{tw}(G)$, is the minimal width of any tree decomposition of G.

Our goal for this section is to design efficient isomorphism tests for graphs of small tree width. Once again, we want to build on the isomorphism test for t-CR-bounded graphs. However, compared to graphs of small degree or genus, identifying parts of graphs of bounded tree width that are t-CR-bounded (after fixing a constant number of vertices) is more challenging. This comes from the fact that the automorphism group of a graph of tree width k may contain arbitrarily large symmetric groups. For example, let $T_{d,h}$ denote a complete d-ary tree of height h, where we should think of d as being unbounded and much larger than k. Also, let $K_k \otimes T_{d,h}$ denote the graph obtained from $T_{d,h}$ by replacing each vertex with a clique of size k and

two cliques being completely connected whenever there is an edge in $T_{d,h}$. Then $\mathrm{tw}(K_k \otimes T_{d,h}) \leq 2k$, the graph $K_k \otimes T_{d,h}$ does not contain any $(k-1)$-separator, and the automorphism group contains various symmetric groups S_d.

To circumvent this problem we exploit *clique-separator decompositions* of *improved graphs* which already form an essential part of the first fixed-parameter tractable isomorphism test for graphs of bounded tree width [58], running in time $2^{\mathcal{O}(k^5 \log k)} n^5$. We first require some additional notation. Let G be a graph. A pair (A, B) where $A \cup B = V(G)$ is called a *separation* if $E(A \setminus B, B \setminus A) = \emptyset$ (here, $E(X, Y) := \{vw \in E(G) \mid v \in X, w \in Y\}$ denotes the set of edges with one endpoint in X and one endpoint in Y). In this case we call $A \cap B$ a *separator*. A separation (A, B) is a called a *clique separation* if $A \cap B$ is a clique and $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. In this case we call $A \cap B$ a *clique separator*.

Definition 5.7 ([15, 58]) The *k-improvement* of a graph G is the graph G^k obtained from G by adding an edge between every pair of non-adjacent vertices v, w for which there are more than k pairwise internally vertex disjoint paths connecting v and w. We say that a graph G *is k-improved* when $G^k = G$.

A graph is *k-basic* if it is k-improved and does not have any separating cliques.

Note that a k-basic graph is 2-connected. We summarize several structural properties of G^k.

Lemma 5.8 ([58]) *Let G be a graph and $k \in \mathbb{N}$.*

1. *The k-improvement G^k is k-improved, i.e., $(G^k)^k = G^k$.*

2. *Every tree decomposition of G of width at most k is also a tree decomposition of G^k.*

3. *There exists an algorithm that, given G and k, runs in $\mathcal{O}(k^2 n^3)$ time and either correctly concludes that $\mathrm{tw}(G) > k$, or computes G^k.*

Since the construction of G^k from G is isomorphism-invariant, the concept of the improved graph can be exploited for isomorphism testing. Given a graph, we can compute the k-improvement and assign a special color to all added edges to distinguish them from the original edges. Hence, it suffices to solve the isomorphism problem for k-improved graphs. In order to further reduce isomorphism testing to the case of k-basic graphs, we build on a decomposition result of Leimer [55] which provides an isomorphism-invariant tree decomposition of a graph into clique-separator free parts.

Theorem 5.9 ([55, 29]) *For every connected graph G there is an isomorphism-invariant tree decomposition (T, β) of G, called clique separator decomposition, such that*

1. *for every $t \in V(T)$ the graph $G[\beta(t)]$ is clique-separator free, and*

2. *each adhesion set of (T, β) is a clique.*

Moreover, given a graph G, the clique separator decomposition of G can be computed in polynomial time.

Observe that applying the theorem to a k-improved graph results in a tree decomposition (T, β) such that $G[\beta(t)]$ is k-basic for every $t \in V(T)$. Using standard dynamic programming arguments, this decomposition allows us to essentially restrict to the case of k-basic graphs at the cost of an additional factor $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ in the running time. Now, a key insight is that k-basic graphs are once again k-CR-bounded after individualizing a single vertex. The next lemma provides us the main tool for proving this. Recall that, for a graph G and sets $A, B \subseteq V(G)$, we denote $E(A, B) := \{vw \in E(G) \mid v \in A, w \in B\}$.

Lemma 5.10 *Let G be a graph and let χ be a coloring that is stable with respect to the Color Refinement algorithm. Let $X_1, \ldots, X_m$ be distinct color classes of χ and suppose that $E(X_i, X_{i+1}) \neq \emptyset$ for all $i \in [m-1]$. Then there are $\min_{i \in [m]} |X_i|$ many vertex-disjoint paths from X_1 to X_m.*

To gain some intuition on the lemma, consider the simple case where $\ell := |X_i| = |X_j|$ for all $i, j \in [m]$. Since χ is stable we conclude that the bipartite graph $H_i := (X_i \cup X_{i+1}, E(X_i, X_{i+1}))$ is non-empty and biregular. Hence, by Hall's Marriage Theorem, there is a perfect matching M_i in H_i of size $|M_i| = \ell$. Taking the disjoint union of all sets M_i, $i \in [m-1]$, gives ℓ vertex-disjoint paths from X_1 to X_m. The general case can be proved using similar arguments. The details can be found in [43].

Lemma 5.11 *Let G be a k-basic graph. Also let $v \in V(G)$ such that $\deg(v) \leq k$. Then $\mathrm{cl}_k^G(v) = V(G)$.*

Proof Let χ denote the k-CR-stable coloring of G after individualizing v. First note that $N(v) \subseteq \mathrm{cl}_k^G(v)$ since $\deg(v) \leq k$. Now suppose towards a contradiction that $\mathrm{cl}_k^G(v) \neq V(G)$ and let W be the vertex set of a connected component of $G - \mathrm{cl}_k^G(v)$. Then $v \notin N(W)$. Since G does not have any separating cliques there exist $w_1, w_2 \in N(W)$ such that $w_1 w_2 \notin E(G)$. Let $u_0, u_1, \ldots, u_m, u_{m+1}$ be a path from $w_1 = u_0$ to $w_2 = u_{m+1}$ such that $u_i \in W$ for all $i \in [m]$. Let $X_i := [u_i]_\chi$ and note that $|X_i| \geq k + 1$. Also note that $X_1 \subseteq N(w_1)$ and $X_m \subseteq N(w_2)$ since $w_1, w_2 \in \mathrm{cl}_k^G(v)$ and χ is stable. Hence, by Lemma 5.10, there are $k + 1$ internally vertex-disjoint paths from w_1 to w_2. But this contradicts the fact that G is k-improved. $\qquad\square$

Overall, this gives an isomorphism test for graphs of tree width k running in time $2^{\mathcal{O}(k \log k)} n^{\mathrm{polylog}(k)}$ as follows. First, the algorithm computes the clique-separator decomposition of the k-improved graph using Lemma 5.8 and Theorem 5.9. The isomorphism problem for the k-basic parts can be solved in time $n^{\mathrm{polylog}(k)}$ by Lemma 5.11 and Corollary 5.3. Finally, the results for the bags can be joined using dynamic programming along the clique separator decomposition. (Formally, this requires the algorithm to solve a more general problem for k-basic graphs where information obtained by dynamic programming on the adhesion sets to the children of a bag are incorporated. But this can be easily achieved within the group-theoretic framework; see e.g. [43].)

Using further advanced techniques (which are beyond the scope of this article) this result can be strengthened into two directions. First, building isomorphism-

invariant decompositions of k-basic graphs, it is possible to build an improved fixed-parameter tractable isomorphism test for graphs of tree width k running in time $2^{k \operatorname{polylog}(k)} n^3$ [42] (improving on the $2^{\mathcal{O}(k^5 \log k)} n^5$ algorithm by Lokshtanov et al. [58]).

And second, isomorphism of graphs of tree width k can also be tested in time $n^{\operatorname{polylog}(k)}$. Here, the crucial step is to speed up to dynamic programming procedure along the clique separator decomposition. Let D be the vertex set of the root bag of the clique separator decomposition and let $Z_1, \ldots, Z_\ell$ be connected components of $G - D$ such that $S := N_G(Z_i) = N_G(Z_j)$ for all $i, j \in [\ell]$. Observe that $|S| \leq k + 1$ since it is a clique by the properties of the clique separator decomposition. After recursively computing representations for the sets of isomorphisms between $(G[Z_i \cup S], S)$ and $(G[Z_j \cup S], S)$ for all $i, j \in [\ell]$ the partial solutions have to be joined into the automorphism group of $(G[Z_1 \cup \cdots \cup Z_\ell \cup S], S)$. A simple strategy is to iterate through all permutations $\gamma \in \operatorname{Sym}(S)$ and check in polynomial time which permutations can be extended to an automorphism. This gives a running time $2^{\mathcal{O}(k \log k)} \ell^{\mathcal{O}(1)}$ as already exploited above. Giving another extension of Babai's quasi-polynomial time algorithm, Wiebking [87] presented an algorithm solving this problem in time $\ell^{\operatorname{polylog}(k)}$. This enables us to achieve the running time $n^{\operatorname{polylog}(k)}$ for isomorphism testing of graphs of tree width k.

5.4 Graphs Excluding Small (Topological) Minors

In the previous two subsections we have seen isomorphism tests running in time $n^{\operatorname{polylog}(g)}$ for graphs of genus g and $n^{\operatorname{polylog}(k)}$ for graphs of tree width k. Both algorithms follow a similar pattern of computing an isomorphism-invariant tree-decomposition of small adhesion-width such that (the torso of) each bag is t-CR-bounded after individualizing a constant number of vertices. Recall that the same strategy also works for graphs of bounded degree (taking the trivial tree decomposition with a single bag). This suggests that a similar approach might work in a much more general setting considering graphs that exclude an arbitrary h-vertex graph as a minor or even as a topological subgraph.

Let G, H be graphs. Recall that H is a minor of G if H can be obtained from G by deleting vertices and edges as well as contracting edges. Moreover, H is a *topological subgraph* of G if H can be obtained from G by deleting vertices and edges as well as dissolving degree-two vertices (i.e., deleting the vertex and connecting its two neighbors). Note that, if H is a topological subgraph of G, then H is also a minor of G (in general, this is not true in the other direction). A graph G *excludes H as a (topological) minor* if it has no (topological) minor isomorphic to H.

A common argument for the correctness of the algorithms considered so far is the construction of forbidden subgraphs from t-CR-stable colorings that are not discrete. In order to generalize the existing algorithms we first introduce a tool that allows us to construct large (topological) minors from t-CR-stable colorings which are not discrete, but still contain some individualized vertices. Let G be a graph, let $\chi \colon V(G) \to C$ be a vertex-coloring and let T be a tree with vertex set $V(T) = C$. A subgraph $H \subseteq G$ *agrees* with T if $\chi|_{V(H)} \colon H \cong T$, i.e., the coloring χ induces an isomorphism between H and T. Observe that each $H \subseteq G$ that agrees with a tree T is also a tree.

For a tree T we define $V_{\leq i}(T) := \{t \in V(T) \mid \deg(t) \leq i\}$ and $V_{\geq i}(T) := \{t \in V(T) \mid \deg(t) \geq i\}$. It is well known that for trees T it holds that $|V_{\geq 3}(T)| < |V_{\leq 1}(T)|$.

Lemma 5.12 ([43]) *Let G be a graph and let $\chi \colon V(G) \to C$ be a vertex-coloring that is stable with respect to the Color Refinement algorithm. Also let T be a tree with vertex set $V(T) = C$ and assume that for every $c_1 c_2 \in E(T)$ there are $v_1 \in \chi^{-1}(c_1)$ and $v_2 \in \chi^{-1}(c_2)$ such that $v_1 v_2 \in E(G)$. Let $m := \min_{c \in C} |\chi^{-1}(c)|$ and let $\ell := 2|V_{\leq 1}(T)| + |V_{\geq 3}(T)|$.*
Then there are (at least) $\lfloor \frac{m}{\ell} \rfloor$ pairwise vertex-disjoint trees in G that agree with T.

Intuitively speaking, assuming there is a stable coloring in which all color classes are large, the lemma states that one can find a large number of vertex-disjoint trees with predefined color patterns. Observe that Lemma 5.12 generalizes Lemma 5.10 which provides such a statement for paths instead of arbitrary trees (ignoring the additional factor of two for the number of vertex-disjoint paths in comparison to the minimum color class size).

The proof of this lemma is quite technical and we omit any details here. As a first, simple application we can provide a strengthened version of Lemma 5.5.

Lemma 5.13 *Let (G, χ) be a 3-connected, vertex-colored graph that excludes $K_{3,h}$ as a topological subgraph and suppose $V_1 \uplus V_2 = V(G)$ such that*

1. *$|[v]_\chi| = 1$ for all $v \in V_1$,*

2. *χ is stable with respect to the Color Refinement algorithm, and*

3. *$|V_1| \geq 3$.*

Then there exists $u \in V_2$ such that $|[u]_\chi| \leq 7h - 1$.

In comparison with Lemma 5.5 the graph G is only required to exclude $K_{3,h}$ as a topological subgraph instead of a minor. Hence, Lemma 5.13 covers a much larger class of graphs. Observe that the bound on the size of the color class $[u]_\chi$ is slightly worse which however is not relevant for most algorithmic applications since the additional factor disappears in the $\mathcal{O}$-notation.

The proof of Lemma 5.13 is similar to the proof of Lemma 5.5. Recall that in the proof of Lemma 5.5 we constructed a tree T' with three leaves representing three vertices $v_1, v_2, v_3 \in V_1$. Instead of using T' to construct a minor $K_{3,h}$ this tree can also be used to construct a topological subgraph $K_{3,h}$ building on Lemma 5.12. Note that $2|V_{\leq 1}(T')| + |V_{\geq 3}(T')| \leq 7$ which gives the additional factor on the bound of $|[u]_\chi|$.

Following the same arguments as in Section 5.2 we obtain a polynomial-time Turing reduction from GI for graphs excluding $K_{3,h}$ as a topological subgraph to GI for $(7h - 1)$-CR-bounded graphs.

Theorem 5.14 *The Graph Isomorphism Problem for graphs excluding $K_{3,h}$ as a topological subgraph can be solved in time $n^{\mathrm{polylog}(h)}$.*

This generalizes the previous algorithms for graphs of bounded degree as well as graphs of bounded genus (see Figure 7).

Next, we wish to generalize the algorithms for graphs of bounded genus and graphs of bounded tree width. More precisely, we aim for an algorithm that decides isomorphism of graphs that exclude an arbitrary h-vertex graph as a minor in time $n^{\mathrm{polylog}(h)}$. Towards this end, first observe that a graph G excludes an arbitrary h-vertex graph as a minor if and only if G excludes K_h as a minor. The maximum h such that K_h is a minor of G is also known as the *Hadwiger number* of G. Note that a graph of Hadwiger number h excludes K_{h+1} as a minor. So in other words, the goal is to give an isomorphism test for graphs of Hadwiger number h running in time $n^{\mathrm{polylog}(h)}$.

As a first basic tool we argue that, given a set $X \subseteq V(G)$, the t-closure of X only stops to grow at a separator of small size. Note that similar properties are implicitly proved for graphs of small genus and graphs of small tree width in the previous two subsections.

Lemma 5.15 ([43]) *Let $t \geq 3h^3$. Let G be a graph that excludes K_h as a topological subgraph. Also let $X \subseteq V(G)$ and let Z be the vertex set of a connected component of $G - \mathrm{cl}_t^G(X)$. Then $|N(Z)| < h$.*

The strategy for the proof is very similar to the previous arguments building on Lemma 5.12. We assume towards a contradiction that $|N(Z)| \geq h$ for some connected component Z of $G - \mathrm{cl}_t^G(X)$ and construct a topological subgraph K_h. Towards this end, it suffices to find vertex-disjoint paths between all pairs of vertices of a subset $Y \subseteq N(Z)$ of size at least h. Using Lemma 5.12 we can actually construct a large number of vertex-disjoint trees such that each $v \in Y$ has a neighbor in all of those trees. In particular, this gives the desired set of vertex-disjoint paths.

Now, similar to the previous examples, the central step is to obtain a suitable isomorphism-invariant decomposition of the input graph so that each part of the decomposition is t-CR-bounded after individualizing a constant number of vertices. For graphs of bounded genus as well as graphs of bounded tree width we could rely on existing decompositions from the literature. However, for graphs excluding K_h as a minor, such results do not seem to be known. In particular, neither of the decompositions applied for graphs of bounded genus and graphs of bounded tree width are suitable for graph classes that only exclude K_h as a minor.

The central idea to resolve this problem is to use the t-closure of a set itself as a way of defining a graph decomposition. We have already established in the last lemma that the t-closure can only stop to grow at a separator of small size. In particular, this enables us to bound the adhesion-width of the resulting tree decomposition (i.e., the maximum size of an adhesion set). The second, more involved challenge is to obtain a decomposition which is isomorphism-invariant. Recall that, in all previous examples, we individualized a constant number of vertices to "initiate the growth of the t-closure". In order to still obtain an isomorphism-invariant decomposition we have to ensure that $\mathrm{cl}_t^G(v)$ is independent of the specific choice of v. Towards this end, we want to argue that there is an isomorphism-invariant set $X \subseteq V(G)$ such that $\mathrm{cl}_t^G(v) = \mathrm{cl}_t^G(w)$ for all $v, w \in X$.

This basic idea can indeed be realized, but the concrete formulation turns out to be slightly more complicated. More precisely, in order to incorporate information

not "visible" to the Color Refinement algorithm, the t-closure is computed on an extended graph which allows us to explicitly encode certain structural information of the input graph.

Lemma 5.16 ([43]) *Let $t = \Omega(h^4)$. There is a polynomial-time algorithm that, given a connected vertex-colored graph G, either correctly concludes that G has a minor isomorphic to K_h or computes a vertex-colored [4] graph (G', χ') and a set $\emptyset \neq X \subseteq V(G')$ such that*

1. *$X = \{v \in V(G') \mid \chi'(v) = c\}$ for some color c,*

2. *$X \subseteq \mathrm{cl}_t^{(G', \chi')}(v)$ for every $v \in X$, and*

3. *$X \subseteq V(G)$.*

Moreover, the output of the algorithm is isomorphism-invariant with respect to G.

The proof of the lemma is rather technical and builds on the 2-dimensional Weisfeiler-Leman algorithm. For details we refer the reader to [43].

Now, an algorithm for testing isomorphism of graphs excluding K_h as a minor can proceed as follows. Given a graph G, we first compute an isomorphism-invariant set $X \subseteq V(G)$ using Lemma 5.16 as well as its closure $D := \mathrm{cl}_t^G(X)$. Then $|N_G(Z)| < h$ for every connected component Z of the graph $G - D$ by Lemma 5.15. Actually, by proceeding recursively on all connected components of $G - D$ we obtain an isomorphism-invariant tree decomposition (T, β) of adhesion-width at most $h - 1$. Moreover, for each bag, we can consider the corresponding extended graph from Lemma 5.16 to obtain a structure associated with the bag that is t-CR-bounded for $t = \Omega(h^4)$ (after individualizing a single vertex). Hence, using similar techniques as for graphs of bounded tree width in the previous subsection, we obtain the following result.

Theorem 5.17 (Grohe, Neuen and Wiebking [43]) *The Graph Isomorphism Problem for graphs excluding K_h as a minor can be solved in time $n^{\mathrm{polylog}(h)}$.*

From the algorithm described above we can also infer some information on the structure of automorphism groups of graphs excluding K_h as a minor.

Theorem 5.18 *Let G be a graph that excludes K_h as a minor. Then there is an isomorphism-invariant tree decomposition (T, β) of G such that*

1. *the adhesion-width of (T, β) is at most $h - 1$, and*

2. *for all $t \in V(T)$ there exists $v \in \beta(t)$ such that $(\mathrm{Aut}(G))_v[\beta(t)] \in \widehat{\Gamma}_d$ for $d = \mathcal{O}(h^4)$.*

---

[4]In [43, Theorem IV.2], this graph is pair-colored, i.e., a color is assigned to each pair of vertices. By adding gadgets for the pair colors such a graph can easily be transformed into a vertex-colored graph while preserving all required properties.

6 Concluding Remarks

We survey recent progress on the Graph Isomorphism Problem. Our focus is on Babai's quasi-polynomial time algorithm and a series of subsequent quasi-polynomial parameterized algorithms. In particular, we highlight the power of a new technique centered around the notion of t-CR-boundedness. Actually, upon completion of this survey article, the second author showed that these techniques can be pushed even further obtaining an isomorphism test running in time $n^{\mathrm{polylog}(h)}$ for graphs only excluding a graph of order h as a topological subgraph [72]. More precisely, building on a very lengthy and technical analysis, it is possible to generalize Lemma 5.16 to graphs only excluding K_h as a topological subgraph. Following the same strategy outlined above, this implies a faster isomorphism test graphs excluding K_h as a topological subgraph.

Research on the isomorphism problem is still very active, and interesting questions remain open. Of course, the most important question remains whether isomorphism of graphs can be tested in polynomial time. Let us also mention a few more specific questions that seem to be within reach.

1. Can isomorphism of hypergraphs be tested in time $n^{\mathrm{polylog}(n)}m^{\mathcal{O}(1)}$ where n denotes the number of vertices and m the number of hyperedges of the input?

2. Can the techniques based on t-CR-boundedness be exploited to design faster isomorphism tests for dense graphs? For example, is there an isomorphism algorithm running in time $n^{\mathrm{polylog}(k)}$ for graphs of rank width at most k?

3. Is GI parameterized by the Hadwiger number fixed-parameter tractable, that is, is there an isomorphism algorithm running in time $f(k)n^{O(1)}$ for graphs excluding a graph of order k as a minor, for some arbitrary function f? The same question can be asked for topological subgraphs.

4. Is GI parameterized by maximum degree number fixed-parameter tractable?

The fourth question and the third for topological subgraphs seem to be hardest. The question whether GI parameterized by maximum degree is fixed parameter tractable has already been raised more than twenty years ago in Downey and Fellow's monograph on parameterized complexity [27].

Other interesting research directions include the Group Isomorphism Problem (see, e.g., [8, 17, 18, 79]) and the Tournament Isomorphism Problem (see, e.g., [11, 81]).

References

[1] Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan, *Exploiting symmetries for scaling loopy belief propagation and relational training*, Mach. Learn. **92** (2013), no. 1, 91–132.

[2] Albert Atserias and Elitza N. Maneva, *Sherali-Adams relaxations and indistinguishability in counting logics*, SIAM J. Comput. **42** (2013), no. 1, 112–137.

[3] Albert Atserias and Joanna Ochremiak, *Definable ellipsoid method, sums-of-squares proofs, and the isomorphism problem*, Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018 (Anuj Dawar and Erich Grädel, eds.), ACM, 2018, pp. 66–75.

[4] László Babai, *Monte Carlo algorithms in graph isomorphism testing*, Tech. Report 79-10, Université de Montréal, 1979.

[5] ———, *Graph isomorphism in quasipolynomial time*, CoRR **abs/1512.03547** (2015).

[6] ———, *Graph isomorphism in quasipolynomial time [extended abstract]*, Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016 (Daniel Wichs and Yishay Mansour, eds.), ACM, 2016, pp. 684–697.

[7] László Babai, Peter J. Cameron, and Péter P. Pálfy, *On the orders of primitive groups with restricted nonabelian composition factors*, J. Algebra **79** (1982), no. 1, 161–168.

[8] László Babai, Paolo Codenotti, and Youming Qiao, *Polynomial-time isomorphism test for groups with no abelian normal subgroups - (extended abstract)*, Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I (Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, eds.), Lecture Notes in Computer Science, vol. 7391, Springer, 2012, pp. 51–62.

[9] László Babai, Paul Erdös, and Stanley M. Selkow, *Random graph isomorphism*, SIAM J. Comput. **9** (1980), no. 3, 628–635.

[10] László Babai, William M. Kantor, and Eugene M. Luks, *Computational complexity and the classification of finite simple groups*, 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983, IEEE Computer Society, 1983, pp. 162–171.

[11] László Babai and Eugene M. Luks, *Canonical labeling of graphs*, Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA (David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, eds.), ACM, 1983, pp. 171–183.

[12] László Babai, Eugene M. Luks, and Ákos Seress, *Permutation groups in NC*, Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA (Alfred V. Aho, ed.), ACM, 1987, pp. 409–420.

[13] Christoph Berkholz, Paul S. Bonsma, and Martin Grohe, *Tight lower and upper bounds for the complexity of canonical colour refinement*, Theory Comput. Syst. **60** (2017), no. 4, 581–614.

[14] Christoph Berkholz and Martin Grohe, *Limitations of algebraic approaches to graph isomorphism testing*, Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I (Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, eds.), Lecture Notes in Computer Science, vol. 9134, Springer, 2015, pp. 155–166.

[15] Hans L. Bodlaender, *Necessary edges in k-chordalisations of graphs*, J. Comb. Optim. **7** (2003), no. 3, 283–290.

[16] Béla Bollobás, *Distinguishing vertices of random graphs*, Ann. Discrete Math. **13** (1982), 33–49.

[17] Jendrik Brachter and Pascal Schweitzer, *On the Weisfeiler-Leman dimension of finite groups*, LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020 (Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, eds.), ACM, 2020, pp. 287–300.

[18] Peter A. Brooksbank, Joshua A. Grochow, Yinan Li, Youming Qiao, and James B. Wilson, *Incorporating Weisfeiler-Leman into algorithms for group isomorphism*, CoRR **abs/1905.02518** (2019).

[19] Jin-yi Cai, Martin Fürer, and Neil Immerman, *An optimal lower bound on the number of variables for graph identification*, Comb. **12** (1992), no. 4, 389–410.

[20] Peter J. Cameron, *Finite permutation groups and finite simple groups*, Bull. London Math. Soc. **13** (1981), no. 1, 1–22.

[21] Alain Cardon and Maxime Crochemore, *Partitioning a graph in $O(|A| \log_2 |V|)$*, Theor. Comput. Sci. **19** (1982), no. 1, 85–98.

[22] Paolo Codenotti, Hadi Katebi, Karem A. Sakallah, and Igor L. Markov, *Conflict analysis and branching heuristics in the search for graph automorphisms*, 25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013, Herndon, VA, USA, November 4-6, 2013, IEEE Computer Society, 2013, pp. 907–914.

[23] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh, *Parameterized algorithms*, Springer, 2015.

[24] Anuj Dawar, David Richerby, and Benjamin Rossman, *Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs*, Ann. Pure Appl. Log. **152** (2008), no. 1-3, 31–50.

[25] Reinhard Diestel, *Graph theory*, 5th ed., Springer Verlag, 2016.

[26] John D. Dixon and Brian Mortimer, *Permutation groups*, Graduate Texts in Mathematics, vol. 163, Springer-Verlag, New York, 1996.

[27] Rodney G. Downey and Michael R. Fellows, *Parameterized complexity*, Monographs in Computer Science, Springer, 1999.

[28] Zdenek Dvořák, *On recognizing graphs by numbers of homomorphisms*, J. Graph Theory **64** (2010), no. 4, 330–342.

[29] Michael Elberfeld and Pascal Schweitzer, *Canonizing graphs of bounded tree width in logspace*, ACM Trans. Comput. Theory **9** (2017), no. 3, 12:1–12:29.

[30] Sergei Evdokimov, Ilia N. Ponomarenko, and Gottfried Tinhofer, *Forestal algebras and algebraic forests (on a new class of weakly compact graphs)*, Discret. Math. **225** (2000), no. 1-3, 149–172.

[31] I. S. Filotti and Jack N. Mayer, *A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper)*, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA (Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, eds.), ACM, 1980, pp. 236–243.

[32] Erich Grädel, Martin Grohe, Benedikt Pago, and Wied Pakusa, *A finite-model-theoretic view on propositional proof complexity*, Log. Methods Comput. Sci. **15** (2019), no. 1, 4:1–4:53.

[33] Martin Grohe, *Fixed-point logics on planar graphs*, Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998, IEEE Computer Society, 1998, pp. 6–15.

[34] ———, *Isomorphism testing for embeddable graphs through definability*, Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA (F. Frances Yao and Eugene M. Luks, eds.), ACM, 2000, pp. 63–72.

[35] Martin Grohe, *Descriptive complexity, canonisation, and definable graph structure theory*, Lecture Notes in Logic, vol. 47, Cambridge University Press, 2017.

[36] Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman, *Dimension reduction via colour refinement*, Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings (Andreas S. Schulz and Dorothea Wagner, eds.), Lecture Notes in Computer Science, vol. 8737, Springer, 2014, pp. 505–516.

[37] Martin Grohe and Sandra Kiefer, *A linear upper bound on the Weisfeiler-Leman dimension of graphs of bounded genus*, 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, eds.), LIPIcs, vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 117:1–117:15.

[38] Martin Grohe and Julian Mariño, *Definability and descriptive complexity on databases of bounded tree-width*, Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings (Catriel Beeri and Peter Buneman, eds.), Lecture Notes in Computer Science, vol. 1540, Springer, 1999, pp. 70–82.

[39] Martin Grohe and Dániel Marx, *Structure theorem and isomorphism test for graphs with excluded topological subgraphs*, SIAM J. Comput. **44** (2015), no. 1, 114–159.

[40] Martin Grohe and Daniel Neuen, *Canonisation and definability for graphs of bounded rank width*, 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019, IEEE, 2019, pp. 1–13.

[41] Martin Grohe, Daniel Neuen, and Pascal Schweitzer, *A faster isomorphism test for graphs of small degree*, 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018 (Mikkel Thorup, ed.), IEEE Computer Society, 2018, pp. 89–100.

[42] Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking, *An improved isomorphism test for bounded-tree-width graphs*, ACM Trans. Algorithms **16** (2020), no. 3, 34:1–34:31.

[43] Martin Grohe, Daniel Neuen, and Daniel Wiebking, *Isomorphism testing for graphs excluding small minors*, 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, November 16-19, 2020 (Virtual Conference) (Sandy Irani, ed.), IEEE Computer Society, 2020, pp. 625–637.

[44] Martin Grohe and Martin Otto, *Pebble games and linear equations*, J. Symb. Log. **80** (2015), no. 3, 797–844.

[45] Martin Grohe and Pascal Schweitzer, *The graph isomorphism problem*, Commun. ACM **63** (2020), no. 11, 128–134.

[46] Martin Grohe, Pascal Schweitzer, and Daniel Wiebking, *Deep Weisfeiler Leman*, Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms, 2021.

[47] Harald Andrés Helfgott, Jitendra Bajpai, and Daniele Dona, *Graph isomorphisms in quasi-polynomial time*, CoRR **abs/1710.04574** (2017).

[48] John E. Hopcroft and Robert Endre Tarjan, *Isomorphism of planar graphs*, Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (Raymond E. Miller and James W. Thatcher, eds.), The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 131–152.

[49] Neil Immerman and Eric Lander, *Describing graphs: A first-order approach to graph canonization*, Complexity Theory Retrospective (Alan L. Selman, ed.), Springer New York, New York, NY, 1990, pp. 59–81.

[50] Tommi A. Junttila and Petteri Kaski, *Conflict propagation and component recursion for canonical labeling*, Theory and Practice of Algorithms in (Computer) Systems - First International ICST Conference, TAPAS 2011, Rome, Italy, April 18-20, 2011. Proceedings (Alberto Marchetti-Spaccamela and

Michael Segal, eds.), Lecture Notes in Computer Science, vol. 6595, Springer, 2011, pp. 151–162.

[51] Richard M. Karp, *Reducibility among combinatorial problems*, Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (Raymond E. Miller and James W. Thatcher, eds.), The IBM Research Symposia Series, Plenum Press, New York, 1972, pp. 85–103.

[52] Sandra Kiefer, *The Weisfeiler-Leman algorithm: An exploration of its power*, ACM SIGLOG News **7** (2020), no. 3, 5–27.

[53] Sandra Kiefer and Daniel Neuen, *The power of the Weisfeiler-Leman algorithm to decompose graphs*, 44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany (Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, eds.), LIPIcs, vol. 138, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 45:1–45:15.

[54] Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer, *The Weisfeiler-Leman dimension of planar graphs is at most 3*, J. ACM **66** (2019), no. 6, 44:1–44:31.

[55] Hanns-Georg Leimer, *Optimal decomposition by clique separators*, Discret. Math. **113** (1993), no. 1-3, 99–123.

[56] Martin W. Liebeck and Aner Shalev, *Simple groups, permutation groups, and probability*, J. Amer. Math. Soc. **12** (1999), no. 2, 497–520.

[57] Steven Lindell, *A logspace algorithm for tree canonization (extended abstract)*, Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada (S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, eds.), ACM, 1992, pp. 400–404.

[58] Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh, *Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth*, SIAM J. Comput. **46** (2017), no. 1, 161–189.

[59] José Luis López-Presa, Luis F. Chiroque, and Antonio Fernández Anta, *Novel techniques to speed up the computation of the automorphism group of a graph*, J. Appl. Math. **2014** (2014), 934637:1–934637:15.

[60] László Lovász, *Operations with structures*, Acta Math. Acad. Sci. Hungar. **18** (1967), 321–328.

[61] Eugene M. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, J. Comput. Syst. Sci. **25** (1982), no. 1, 42–65.

[62] Peter N. Malkin, *Sherali-Adams relaxations of graph isomorphism polytopes*, Discret. Optim. **12** (2014), 73–97.

[63] Attila Maróti, *On the orders of primitive groups*, J. Algebra **258** (2002), no. 2, 631–640.

[64] Brendan D. McKay, *Practical graph isomorphism*, Congr. Numer. **30** (1981), 45–87.

[65] Brendan D. McKay and Adolfo Piperno, *Practical graph isomorphism, II*, J. Symb. Comput. **60** (2014), 94–112.

[66] Gary L. Miller, *Isomorphism testing for graphs of bounded genus*, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA (Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, eds.), ACM, 1980, pp. 225–235.

[67] ———, *Isomorphism of graphs which are pairwise k-separable*, Inf. Control. **56** (1983), no. 1/2, 21–33.

[68] Howard L. Morgan, *The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service.*, J. Chem. Doc. **5** (1965), no. 2, 107–113.

[69] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe, *Weisfeiler and Leman go neural: Higher-order graph neural networks*, The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press, 2019, pp. 4602–4609.

[70] Daniel Neuen, *The power of algorithmic approaches to the graph isomorphism problem*, Ph.D. thesis, RWTH Aachen University, Aachen, Germany, 2019.

[71] ———, *Hypergraph isomorphism for groups with restricted composition factors*, 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference) (Artur Czumaj, Anuj Dawar, and Emanuela Merelli, eds.), LIPIcs, vol. 168, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 88:1–88:19.

[72] ———, *Isomorphism testing for graphs excluding small topological subgraphs*, CoRR **abs/2011.14730** (2020).

[73] Ryan O'Donnell, John Wright, Chenggang Wu, and Yuan Zhou, *Hardness of robust graph isomorphism, Lasserre gaps, and asymmetry of random graphs*, Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014 (Chandra Chekuri, ed.), SIAM, 2014, pp. 1659–1677.

[74] Robert Paige and Robert E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput. **16** (1987), no. 6, 973–989.

[75] Ilia N. Ponomarenko, *Polynomial isomorphism algorithm for graphs which do not pinch to $K_{3,g}$*, J. Math. Sci. **34** (1986), 1819–1831.

[76] ———, *The isomorphism problem for classes of graphs that are invariant with respect to contraction*, Zap. Nauchn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI) **174** (1988), no. Teor. Slozhn. Vychisl. 3, 147–177, 182.

[77] Louis C. Ray and Russell A. Kirsch, *Finding chemical records by digital computers*, Science **126** (1957), no. 3278, 814–819.

[78] Gerhard Ringel, *Das geschlecht des vollständigen paaren graphen*, Abh. Math. Semin. Univ. Hambg. **28** (1965), no. 3, 139–150.

[79] David J. Rosenbaum, *Beating the generator-enumeration bound for solvable-group isomorphism*, ACM Trans. Comput. Theory **12** (2020), no. 2, 12:1–12:18.

[80] Joseph J. Rotman, *An introduction to the theory of groups*, fourth ed., Graduate Texts in Mathematics, vol. 148, Springer-Verlag, New York, 1995.

[81] Pascal Schweitzer, *A polynomial-time randomized reduction from tournament isomorphism to tournament asymmetry*, 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland (Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, eds.), LIPIcs, vol. 80, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 66:1–66:14.

[82] Ákos Seress, *Permutation group algorithms*, Cambridge Tracts in Mathematics, vol. 152, Cambridge University Press, Cambridge, 2003.

[83] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt, *Weisfeiler-Lehman graph kernels*, J. Mach. Learn. Res. **12** (2011), 2539–2561.

[84] Gottfried Tinhofer, *A note on compact graphs*, Discret. Appl. Math. **30** (1991), no. 2-3, 253–264.

[85] Stephen H. Unger, *GIT - a heuristic program for testing pairs of directed line graphs for isomorphism*, Commun. ACM **7** (1964), no. 1, 26–34.

[86] Boris Y. Weisfeiler and Andrei A. Leman, *The reduction of a graph to canonical form and the algebra which appears therein*, NTI, Series 2 (1968), English transalation by G. Ryabov available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

[87] Daniel Wiebking, *Graph isomorphism in quasipolynomial time parameterized by treewidth*, 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference) (Artur Czumaj, Anuj Dawar, and Emanuela Merelli, eds.), LIPIcs, vol. 168, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 103:1–103:16.

[88] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, *How powerful are graph neural networks?*, 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.

RWTH Aachen University
Ahornstr. 55
52074 Aachen
Germany
grohe@informatik.rwth-aachen.de

CISPA Helmholtz Center for Information Security
Stuhlsatzenhaus 5
66123 Saarbruecken
Germany
daniel.neuen@cispa.saarland