Benchmark Graphs for Practical Graph Isomorphism*

Daniel Neuen[1] and Pascal Schweitzer[2]

1 **RWTH Aachen University, Aachen, Germany**
neuen@informatik.rwth-aachen.de
2 **RWTH Aachen University, Aachen, Germany**
schweitzer@informatik.rwth-aachen.de

Abstract

The state-of-the-art solvers for the graph isomorphism problem can readily solve generic instances with tens of thousands of vertices. Indeed, experiments show that on inputs without particular combinatorial structure the algorithms scale almost linearly. In fact, it is non-trivial to create challenging instances for such solvers and the number of difficult benchmark graphs available is quite limited.

We describe a construction to efficiently generate small instances for the graph isomorphism problem that are difficult or even infeasible for said solvers.

Up to this point the only other available instances posing challenges for isomorphism solvers were certain incidence structures of combinatorial objects (such as projective planes, Hadamard matrices, Latin squares, etc.). Experiments show that starting from 1500 vertices our new instances are several orders of magnitude more difficult on comparable input sizes. More importantly, our method is generic and efficient in the sense that one can quickly create many isomorphism instances on a desired number of vertices. In contrast to this, said combinatorial objects are rare and difficult to generate and with the new construction it is possible to generate an abundance of instances of arbitrary size.

Our construction hinges on the multipedes of Gurevich and Shelah and the Cai-Fürer-Immerman gadgets that realize a certain abelian automorphism group and have repeatedly played a role in the context of graph isomorphism. Exploring limits, we also explain that there are group theoretic obstructions to generalizing the construction with non-abelian gadgets.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph isomorphism, benchmark instances, practical solvers

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.60

1 Introduction

The graph isomorphism problem, which asks for structural equivalence of two given input graphs, has been extensively investigated since the beginning of theoretical computer science, both from a theoretical and a practical point of view. Designing isomorphism solvers in practice is a non-trivial task. Nevertheless various efficient algorithms, namely nauty and traces [15], bliss [9], conauto [11], and saucy [6] are available as software packages. These state-of-the-art solvers for the graph isomorphism problem (or more generally graph canonization) can readily solve generic instances with tens of thousands of vertices. Indeed, experiments show that on inputs without particular combinatorial structure the algorithms scale almost

* A full version of the paper is available at https://arxiv.org/abs/1705.03686.

linearly. In practice, this sets the isomorphism problem aside from typical NP-complete problems which usually have an abundance of difficult benchmark instances. The practical algorithms underlying these solvers differ from the ones employed to obtain theoretical results. Indeed, there is a big disconnect between theory and practice [2]. One could interpret Babai's recent breakthrough, the quasipolynomial time algorithm [1], as a first step of convergence. The result implies that if graph isomorphism is NP-complete then all problems in NP have quasi-polynomial time algorithms, which may lead one to also theoretically believe that graph isomorphism is not NP-complete.

In this paper we are interested in creating difficult benchmark instances for the graph isomorphism problem. It would be a misconception to think that for unstructured randomly generated instances graph isomorphism should be hard in practice. Quite the opposite is true. Instances that encompass sufficient randomness usually turn out to be among the easiest instances (see [14] for extensive tests on various graphs and [3, 10] for theoretical arguments). In fact, it is non-trivial to create challenging instances for efficient isomorphism solvers and so far the number of difficult benchmark graphs available is quite limited.

The prime source for difficult instances are graphs arising from combinatorial structures. In 1978, Mathon [13] provided a set of benchmark graphs arising from combinatorial objects such as strongly regular graphs, block designs and coherent configurations. The actual graphs that are provided, having less than 50 vertices, are small by todays standards. Nowadays, larger combinatorial objects yielding difficult graphs are known. The most challenging examples are for instance incidence graphs of projective planes, Hadamard matrices, or Latin squares (see [14]). It is important to understand that not all such incidence structures yield difficult graphs. Indeed, typically there is an algebraic version that can readily be generated. For example to obtain a projective plane it is possible to consider incidences between one- and two-dimensional subspaces of a three-dimensional vector space over a finite field. If however one uses such an algebraic construction then the resulting graph automatically has a large automorphism group. The practical isomorphism solvers are tuned to finding such automorphisms and to exploit them for search space contraction. On top of that, there are also theoretical reasons that lead one to believe that graphs with automorphisms make easier examples for the practical tools (see [22, Theorem 9]). The non-algebraic counterparts of combinatorial structures often do not have the shortcoming of having a large automorphism group, but in contrast to the algebraic constructions they are rare and difficult to generate.

The second source of difficult instances is based on modifications of the so called Cai-Fürer-Immerman construction [5]. This construction is connected to the family of Weisfeiler-Leman algorithms, which for every integer k contains a k-dimensional version that constitutes a polynomial time algorithm used to distinguish non-isomorphic graphs. The 1-dimensional variant (usually called color refinement) is a subroutine in all competitive practical isomorphism solvers. For larger k, however, the k-dimensional variant becomes impractical due to excessive space consumption. For each k, Cai, Fürer and Immerman construct pairs of non-isomorphic graphs that are not distinguished by the k-dimensional variant of algorithm. In 1996, the construction was adapted by Miyazaki to explicitly show that the then current version of nauty has exponential running time in the worst case [16]. The Cai-Fürer-Immerman graphs and the Miyazaki graphs constitute families of benchmark graphs that (despite being specifically designed to fool the Weisfeiler-Leman isomorphism algorithm and the canonical labeling tool nauty, respectively) are infeasible for ad hoc graph isomorphism algorithms. Nowadays, however, most state-of-the-art solvers scale reasonably well on these instances.

Contribution. We describe a construction to efficiently generate instances on any desired number of vertices for the graph isomorphism problem that are difficult or even infeasible for all state-of-the-art graph isomorphism solvers.

Experiments show the graphs pose by far the most challenging graphs to date. Already for 1500 vertices our new instances are by several orders of magnitude more difficult than all previously available benchmark graphs on comparable input sizes. More importantly, the algorithm that generates the instances is generic and efficient in that one can quickly create an abundance of isomorphism instances on a desired number of vertices. We can thereby generate instances in size ranges for which no other difficult benchmark graphs are available.

Our construction, the resulting graphs of which we call shrunken multipedes, hinges on a construction of Gurevich and Shelah [8]. For every k, they describe combinatorial structures that are rigid but cannot be distinguished by the k-dimensional Weisfeiler-Leman algorithm. Guided by the intuition that rigid graphs (i.e., graphs without non-trivial automorphisms) constitute harder instances, we alter the construction to a simple efficient algorithm. In this algorithm we start with a bipartite rigid base graph that is obtained by connecting two explicit graphs (cycles with added diagonals) randomly. We then apply a suitable adaptation of the Cai-Fürer-Immerman (CFI) construction. We prove that the resulting graph is rigid asymptotically almost surely. Our experiments show that even for small sizes the graphs are already rigid with high probability. The result is a simple randomized algorithm that efficiently creates challenging instances of the graph isomorphism problem. The algorithm can be used to create instances for any desired size range. Moreover it is possible to create many non-isomorphic graphs such that every set of two of them forms a difficult instance.

To create practical benchmark graphs, it is imperative to keep all gadget constructions as small as possible. We therefore employ two techniques to save vertices without changing the local automorphism structure of the gadgets. The first technique is based on linear algebra and reduces the vertices in a manner that maintains the rigidity of the graphs. The second technique bypasses certain vertices of the gadgets used in the CFI-construction.

The CFI-gadgets have repeatedly played a role in the context of graph isomorphism. Exploring limits of such constructions, we also explain that there are group theoretic obstructions to generalizing the construction with non-abelian gadgets.

Finally we show with experimental data that our benchmark instances constitute quite challenging problems for all state-of-the-art isomorphism solvers. We compare the running times on the new benchmarks with the combinatorial graphs, the CFI-graphs and the Miyazaki graphs mentioned above.

Canonical forms vs. Isomorphism testing. The task of computing a canonical form is to compute a graph isomorphic to a given input graph so that the output only depends on the isomorphism type of the input and not the actual input (see [15]). Various practical applications require the computation of canonical forms rather than isomorphism tests.

The isomorphism problem reduces, theoretically and practically, to the task of computing a canonical form. Indeed, to check two graphs for isomorphism one computes their canonical forms and then performs a trivial equality check of the results. While computing a canonical form of a graph could in principle be harder than testing isomorphism, several (but not all) of the currently fastest isomorphism solvers actually compute a canonical labeling and then perform the equality check. Our graphs constitute difficult instances both for the task to compute canonical forms and for graph isomorphism. While for the former single graphs need to be constructed, for the latter we need pairs of graphs. (See Subsection 3.1.)

Theoretical bounds. The benchmark graphs described in this paper are explicitly designed to pose a challenge for practical isomorphism solvers. Since they have bounded degree they can be solved (theoretically) in polynomial time [12]. We should remark that we expect a tailored algorithm can be designed that reduces our benchmarks to instances of bounded color class size and then efficiently performs an isomorphism test. Concerning individualization-refinement algorithms, which the tools mentioned above are, using a different but related construction, exponential lower bounds can be proven. We refer to [20].

Obtaining benchmark graphs. Our families of benchmark graphs can be downloaded at `https://www.lii.rwth-aachen.de/research/95-benchmarks.html`. While some of our instances may constitute the most difficult instances yet, in practice, worst-case running time is not the most important measure. In many applications, the solvers have to sift through an enormous number of instances, most of which are easy. It is therefore important to perform extremely well on easy instances, and adequately on difficult instances, not the other way around. We refer to [14] for a well-rounded library in that regard. Nevertheless, we hope our benchmark graphs help to improve current and future isomorphism solvers.

2 Preliminaries

Graphs and isomorphism. A *graph* is a pair $G = (V, E)$ with vertex set $V = V(G)$ and edge relation $E = E(G)$. In this work all graphs are simple, undirected graphs. The *neighborhood* of $v \in V(G)$ is denoted $N(v)$. A *path* is a sequence $(v_1, \dots, v_\ell)$ of distinct vertices such that $\{v_i, v_{i+1}\} \in E(G)$ for all $i \in \{1, \dots, \ell - 1\}$. If additionally $\{v_1, v_\ell\} \in E(G)$ then the sequence $(v_1, \dots, v_\ell)$ is a *cycle* of G. An *isomorphism* from a graph G to another graph H is a bijective mapping $\varphi\colon V(G) \to V(H)$ which preserves the edge relation, that is $\{v, w\} \in E(G)$ if and only if $\{\varphi(v), \varphi(w)\} \in E(H)$ for all $v, w \in V(G)$. The notation $G \cong H$ indicates that such an isomorphism exists. The graph isomorphism problem asks, given two graphs G and H, whether $G \cong H$. An *automorphism* of a graph G is an isomorphism from G to itself. By $\mathrm{Aut}(G)$ we denote the automorphisms of G. A graph G is *rigid* (or *asymmetric*) if its automorphism group $\mathrm{Aut}(G)$ is trivial, that is, the only automorphism of G is the identity.

The Cai-Fürer-Immerman Construction. Our constructions presented in this paper make use of a construction of Cai, Fürer and Immerman [5]. In terms of the graph-isomorphism problem, they used the construction to show that for every k there is a pair of graphs not distinguished by the k-dimensional Weisfeiler-Leman algorithm. The basic building block for these graphs is the CFI gadget X_3. This gadget has 4 *inner vertices* $m_1, \dots, m_4$ and 3 pairs of *outer vertices* a_i, b_i with $i \in \{1, 2, 3\}$. The edges are a_1m_3, a_1m_4, b_1m_1, b_1m_2, a_2m_2, a_2m_4, b_2m_1, b_2m_3, a_3m_2, a_3m_3, b_3m_1, and b_3m_4. The crucial property of X_3 is that a bijection of the outer vertices mapping the set $\{a_i, b_i\}$ to itself for all $i \in \{1, 2, 3\}$ extends to an automorphism of X_3 if and only if an even number of pairs of outer vertices is swapped.

Now let G be a connected 3-regular graph called the *base graph* of the construction and let $T \subseteq E(G)$ be a subset of its edges. Then the CFI construction applied to G results in the following graph $\mathrm{CFI}(G, T)$. For every vertex $v \in V(G)$ with incident edges $e_i = (v, w_i) \in E(G)$ we add a copy of X_3. The inner vertices are denoted by $m_1(v), \dots, m_4(v)$. Each pair $\{a_i, b_i\}$ is associated with the edge e_i and we denote the vertices by $a(v, w_i)$ and $b(v, w_i)$. For every edge $e = \{v, w\} \in E(G)$ with $e \in T$ we add edges $\{a(v, w), b(w, v)\}$ and $\{b(v, w), a(w, v)\}$. If $e \notin T$ we add edges $\{a(v, w), a(w, v)\}$ and $\{b(v, w), b(w, v)\}$. The edges $e \in T$ are called *twisted* edges whereas edges $e \notin T$ are *non-twisted*.

Using the properties of the gadget X_3 described above one can now show that the isomorphism type of $\mathrm{CFI}(G,T)$ depends only on the parity of $|T|$. That is $\mathrm{CFI}(G,T) \cong \mathrm{CFI}(G,T')$ if and only if $|T| \equiv |T'| \mod 2$. We obtain a pair of non-isomorphic graphs $\mathrm{CFI}(G) := \mathrm{CFI}(G,\emptyset)$ and $\widetilde{\mathrm{CFI}}(G) := \mathrm{CFI}(G,\{e\})$ for some edge $e \in E(G)$.

For appropriate base graphs G (i.e., large tree width [7]) these graphs are difficult to distinguish by the Weisfeiler-Leman algorithms. This makes them candidates for being difficult instances for isomorphism testing. However, practical isomorphism solvers cope with CFI graphs reasonably well. The main reason for this is that the underlying algorithms take advantage of automorphisms of the input graphs to restrict their search space and CFI graphs typically have many automorphisms. Indeed, let $C = (v_1, \ldots, v_\ell)$ be a cycle in the base graph G. Then there is an automorphism of $\mathrm{CFI}(G)$ that exactly swaps those $a(v,w)$ and $b(v,w)$ for which $\{v,w\}$ is an edge in C and fixes all other $a(v,w)$ and $b(v,w)$ vertices. Thus, if ℓ is the dimension of the cycle space of G then $\mathrm{CFI}(G)$ has at least 2^ℓ automorphisms. In the next section we address this issue and describe a similar but probabilistic construction that gives with high probability graphs without non-trivial automorphisms.

3 A Rigid Base Construction

Desirable properties. Before we describe our construction we would like to give some intuition about the desirable properties that we want the final graph to have and why we think they are responsible for making the graphs difficult.

- (rigidity) There are some arguments that may make us believe that rigid graphs are more difficult for isomorphism solvers than graphs with automorphisms. Indeed, the graphs for which we can prove theoretical lower bounds [20] are rigid. Furthermore, it is possible to obtain upper bounds on the running time in terms of $|\Gamma|/|\operatorname{Aut}(G)|$ where Γ is a group known to contain all automorphisms (see [22] for a discussion). However, we can also offer an intuitive explanation. Isomorphism solvers are used in practice in search algorithms (e.g., SAT-solvers) to exploit symmetry and thereby cut off parts of the search tree. However, the isomorphism solvers themselves also exploit this strategy in bootstrapping manner, using symmetries to cut off parts of their own search tree.
- (small constants) For our practical purposes it is imperative to keep the graphs small. We thus need to diverge from the theoretical construction in [20]. As a crucial part of the construction we devise two methods to reduce the number of vertices while maintaining the difficulty level of the graph. These reductions are described in the next section.
- (simplicity) We strive to have a simple construction that is not only easy to understand but can also be quickly generated by a simple algorithm. In contrast, for many other graphs coming from combinatorial constructions it takes far longer to construct the graphs than to perform isomorphism tests. The simplicity also allows us to construct an abundance of benchmark graphs.
- (difficult for the WL algorithm) The Weisfeiler-Leman algorithms are a family of theoretical graph isomorphism algorithms. For a description and intuition as to why one may believe they can be used as a measure of the difficulty of a graph, we refer to [20].

3.1 The multipede construction

In the following we describe an explicit construction for families of similar, rigid, non-isomorphic graphs. This construction is based on the multipedes of Gurevich and Shelah [8] yielding finite rigid structures and uses the construction by Cai, Fürer and Immerman [5].

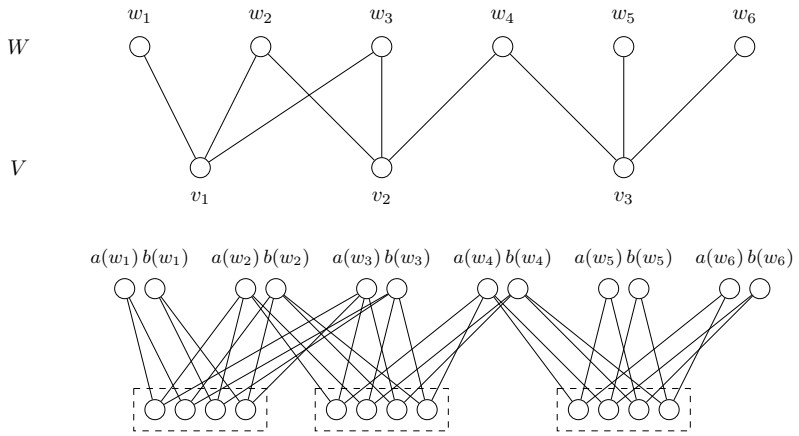

Figure 1 The figure depicts a base graph G on the top and the graph $R(G)$ obtained by applying the multipede construction on the bottom.

Let $G = (V, W, E)$ be a bipartite graph such that every vertex $v \in V$ has degree 3. Define the multipede construction as follows. We replace every $w \in W$ by two vertices $a(w)$ and $b(w)$. For $w \in W$ let $F(w) = \{a(w), b(w)\}$ and for $X \subseteq W$ let $F(X) = \bigcup_{w \in X} F(w)$. We then replace every $v \in V$ by a copy of X_3 and identify the vertices a_i and b_i with $a(w_i)$ and $b(w_i)$ where $w_1, \ldots, w_3$ are the neighbors of v. The resulting graph will be denoted by $R(G)$. An example of this construction is shown in Figure 1.

▶ **Definition 1.** Let $G = (V, W, E)$ be a bipartite graph. We say G is *odd* if for every $\emptyset \neq X \subseteq W$ there is a $v \in V$ such that $|X \cap N(v)|$ is odd.

For a graph G and a vertex $v \in V(G)$ we define the *second neighborhood* of v to be $N^2_G(v) = \{u \in V(G) \mid u \neq v \wedge \exists w \in V(G) \colon \{v, w\}, \{w, u\} \in E(G)\}$.

▶ **Lemma 2.** *Let $G = (V, W, E)$ be a bipartite graph, such that*
1. *$|N(v)| = 3$ for all $v \in V$,*
2. *G is odd,*
3. *G is rigid, and*
4. *there are no distinct $w_1, w_2 \in W$, such that $N^2_G(w_1) = N^2_G(w_2)$.*

Then $R(G)$ is rigid.

We now present a simple randomized algorithm to construct rigid bipartite graphs that are odd. Let G be a 3-regular graph and let $\sigma : E(G) \to E(G)$ be a random permutation of the edges of G. We define the bipartite graph $B(G, \sigma) = (V_B, W_B, E_B)$ by setting $V_B = V(G) \times \{0, 1\}$, $W_B = E(G)$, and $E_B = \{\{(v, 0), e\} \mid v \in e\} \cup \{\{(v, 1), e\} \mid v \in \sigma(e)\}$.

Thus, the edges of G correspond to the vertices in the partition class W_B of $B(G, \sigma)$. Each such edge $e = \{v, w\}$ has an associated edge $\sigma(e) = \{v', w'\}$ and e has exactly four neighbors, namely $(v, 0)$, $(w, 0)$, $(v', 1)$ and $(w', 1)$. Another way of visualizing this construction is to start with two copies of G, subdivide all edges in each copy and then identify the newly added vertices in the one copy with the newly added vertices in the other copy using a randomly chosen matching.

Let G_n be the *$2n$-cycle with diagonals added*. More precisely, $G_n := (V_n, E_n)$ where $V_n = \{1, \ldots, 2n\}$ and $E_n = \{\{i, i + 1 \bmod 2n\} \mid 1 \leq i \leq 2n\} \cup \{\{i, i + n\} \mid 1 \leq i \leq n\}$. We call the edges of the form $\{i, i + n\}$ *diagonals*.

In reference to [8] we call the graphs $R(B(G_n, \sigma))$ the multipede graphs. Observe that the multipede graphs can be constructed in linear time[1]. It can be computed that the graph $R(B(G_n, \sigma))$ has an average degree of $48/11 \approx 4.363$.

We can analyze this construction and show that for the base graphs G_n the resulting graphs $R(B(G_n, \sigma))$ are with high probability rigid. For this, building on Lemma 2, we show that with high probability the graph $B(G_n, \sigma)$ is odd, rigid, and has no distinct vertices in W_B with equal second neighborhoods.

▶ **Theorem 3.** *The probability that $R(B(G_n, \sigma))$ is not rigid is in $\mathcal{O}\left(\frac{\log^2 n}{n}\right)$.*

While it is difficult to extract hidden constants in the theorem, our experiments show that even for small n the probability that $R(B(G_n, \sigma))$ is not rigid is close to 0.

Creating pairs. The construction presented up to this point produces a single graph given n and σ, but instances to the graph isomorphism problem are pairs of graphs. Isomorphic pairs can of course always be obtained using two random permutations of one graph. For non-isomorphic pairs, following [5], to obtain the second graph, we twist one of the connections in one of the gadgets, that is, we switch the neighborhoods of the vertices $a(w)$ and $b(w)$ within one of the gadgets when creating $R(B(G_n, \sigma))$ from $B(G_n, \sigma)$. Arguments similar to those for Theorem 3 show that this creates with high probability pairs of non-isomorphic graphs. (This is not true anymore if the reduction from Subsection 4.1 is applied.)

4 The shrunken multipedes

As we described before we are interested in keeping the number of vertices of our construction small. In this section we describe two methods to reduce the number of vertices of the multipede graphs while, according to our experiments, essentially preserving the difficulty.

4.1 A linear algebra reduction

For a bipartite graph $G = (V, W, E)$ let $A_G \in \mathbb{F}_2^{V \times W}$ be the matrix with $A_{v,w} = 1$ if and only if $vw \in E(G)$ and let $\mathrm{rk}(A)$ be the $\mathbb{F}_2$-rank of A.

▶ **Lemma 4.** *A bipartite graph $G = (V, W, E)$ is odd if and only if $\mathrm{rk}(A_G) = |W|$.*

Proof. For the forward direction suppose $x \in \mathbb{F}_2^W$ such that $A_G x = 0$. Set $X = \{w \in W \mid x_w = 1\}$. Suppose towards a contradiction that $X \neq \emptyset$. Since G is odd there is some $v \in V$ such that $|N(v) \cap X|$ is odd. But then $(A_G)_v x = 1$ where $(A_G)_v$ is the v-th row of A_G. This is a contradiction and thus, $\{x \in \mathbb{F}_2^W \mid A_G x = 0\} = \{0\}$. So $\mathrm{rk}(A_G) = |W|$.

Suppose $\mathrm{rk}(A_G) = |W|$. Then $\{x \in \mathbb{F}_2^W \mid A_G x = 0\} = \{0\}$. Let $\emptyset \neq X \subseteq W$ and let $x \in \mathbb{F}_2^W$ be the vector with $x_w = 1$ if and only if $w \in X$. Since $x \notin \{x \in \mathbb{F}_2^W \mid A_G x = 0\}$ there is some $v \in V$ such that $(A_G)_v x = 1$. But this means that $|N(v) \cap X|$ is odd. ◀

▶ **Corollary 5.** *Let $G = (V, W, E)$ be an odd bipartite graph. Then there is some $V' \subseteq V$ with $|V'| \leq |W|$ such that the induced subgraph $G[V' \cup W]$ is odd.*

Using Gaussian elimination, we can compute such a set in polynomial time. Now suppose $B(G_n, \sigma)$ is odd. Then we can use the previous corollary to compute an induced subgraph $B^* := B^*(G_n, \sigma)$ of $B(G_n, \sigma)$ which is odd and has fewer vertices. Applying the rigid base

[1] Explicit pseudocode for constructing the multipede graphs is given in the full version [19].

construction to B^* the resulting graph has $|V(R(B^*))| = 4 \cdot 3 \cdot n + 2 \cdot 3 \cdot n = 18n$ vertices. The average degree has decreased to 4. In comparison, $|V(R(B(G_n, \sigma)))| = 4 \cdot 4 \cdot n + 2 \cdot 3 \cdot n = 22n$.

We want to stress at this point that the twisted and non-twisted version of $R(B^*)$ are indeed isomorphic. A simple trick to overcome this problem (but not investigated here) is to retain one more element of $V \setminus V'$ and to twist an edge of the respective additional gadget.

4.2 Bypassing the outer vertices

Consider the CFI-construction applied to a 3-regular base graph on n vertices. An easy trick is to identify the a and b vertices for each edge of the base graph instead of connecting them by an edge. More precisely, for each edge $e = \{v, w\}$ in the base graph, we can identify $a(v, w)$ with $a(w, v)$ and $b(v, w)$ with $b(w, v)$ in case of a non-twisted connection or identify $a(v, w)$ with $b(w, v)$ and $b(v, w)$ with $a(w, v)$ in case of a twisted connection. This way the number of vertices reduces from $10n$ to $7n$. This can further be improved by removing all a and b vertices altogether and connecting inner vertices $m_i(v)$ to $m_j(w)$ if both are connected to either $a(v, w)$ or $b(v, w)$. This way the construction only has $4n$ vertices.

A similar technique can also be applied to the rigid base construction. For this we bypass all vertices of degree 8 by directly joining their neighbors and then removing all such vertices. Given a bipartite base graph G we denote by $R^*(G)$ the graph obtained form $R(G)$ by bypassing the outer vertices. When reducing the graphs in that fashion one has to be aware that combinatorial structure of the graph might get lost. In particular it can be the case that vertices of degree 4 (in G) had the same neighborhood before they were removed, leading to inhomogeneity in the graph $R^*(G)$. In fact our experiments show that there is wider spread concerning the difficulty of the graphs $R^*(G)$ than the graphs $R(G)$. In other words, the rigidity and difficulty of those graphs depends more heavily on the choice of the matching σ than for the rigid base construction. However, the graphs $R^*(G)$ turn out to be more difficult than those obtained from the multipede construction.

4.3 Applying both reductions

We can combine the two vertex reduction techniques presented in this section by first applying Corollary 5 to the base graph and then bypassing the outer vertices. The number of vertices in the combined reduction decreases to $12n$. It is not difficult to see that the graphs have an average degree of at most 24, but the average degree varies among graphs on equally many vertices since there may be multiple bypass edges having the same endpoints.

Our experiments show that the combination of the two reductions (Section 6) yields the most difficult graphs. We call the resulting graphs $R^*(B^*(G_n, \sigma))$ the *shrunken multipedes*.

5 Cai-Fürer-Immerman gadgets for other groups

The constructions described in the previous sections revolve around the CFI-gadget X_3. On the outer vertices, the automorphism group of that gadget induces the set of all permutations that swap an even number of pairs. This corresponds to the subgroup of $(\mathbb{Z}_2)^3$ of elements (g_1, g_2, g_3) with $g_1 g_2 g_3 = 1$. A natural idea to push the CFI-construction to its limits is to encode other groups by the use of other gadgets.

Indeed, it is not difficult to see that for every permutation group Δ acting on a set Ω there is a graph gadget X_Δ with a vertex set containing Ω as $\mathrm{Aut}(X_\Delta)$-invariant subset such that $\mathrm{Aut}(X_\Delta) \cong \Delta$ and $\mathrm{Aut}(G)|_\Omega = \Delta$. (See [21, Lemma 16] for a construction.) In other words, every permutation group Δ can be *realized* by a graph gadget X_Δ. As explained

before, the original CFI-gadget realizes a group that is a subdirect product of $\mathbb{Z}_2^3$, which is in particular an abelian group. In analogy to our previous terminology we call the vertices in Ω the outer vertices and the vertices in $V(X_\Delta) \setminus \Omega$ the inner vertices. For our purpose of creating benchmark graphs, we are interested in keeping the number of inner vertices small.

In our constructions, Ω naturally decomposes into sets $C_1, C_2, C_3, \ldots, C_t$ of equal size n as the classes of the outer vertices. We think of the sets as being colored with different colors and only consider color preserving isomorphisms, i.e., permutations mapping each vertex to a vertex of the same color. If we insert a gadget X_Δ with $\Omega = C_1 \cup C_2 \cup \ldots C_t$ we obtain a subgroup of $\mathrm{Sym}(C_1) \times \mathrm{Sym}(C_2) \times \cdots \times \mathrm{Sym}(C_t)$, the direct product of symmetric groups.

For simplicity, we focus on the case $t = 3$ from now on. Thus we consider colored graph gadgets X_Δ such that $C_1 \cup C_2 \cup C_3 \subseteq V(X_\Delta)$ and for which the automorphism group $\mathrm{Aut}(X_\Delta)$ induces on $C_1 \cup C_2 \cup C_3$ a certain subgroup $\Delta \leq \mathrm{Sym}(C_1) \times \mathrm{Sym}(C_2) \times \mathrm{Sym}(C_3)$. Once we have a suitable gadget X_Δ we can use it to obtain a generalized CFI-construction (see [19]).

This brings us to the question, what types of gadgets are suitable for our purpose of creating hard benchmark graphs. Various results in the context of algorithmic group theory can lead one to believe that small groups and also abelian groups may be algorithmically easier than large or non-abelian groups ([4, 12, 24]). Thus, one may wonder whether more difficult benchmark graphs arise when employing graph gadgets inducing more complicated abelian or even non-abelian automorphism groups. We now explore these thoughts.

5.1 Examples for more general groups

General abelian groups. Let $\Gamma \leq S_n$ be an abelian permutation group on n elements. Then $\Delta := \{(a, b, c) \in \Gamma^3 \mid abc = 1\}$ is a subgroup of $S_n \times S_n \times S_n$. Here S_n denotes the symmetric group of the set $\{1, \ldots, n\}$. A description of a gadget realizing this group for the case in which Γ is transitive is described in the full version [19].

For $\Gamma = \mathbb{Z}_2$ we recover the CFI-gadget. Let us point out that for $\Gamma = \mathbb{Z}_k$ these gadgets were used in [23] to show hardness of graph isomorphism for certain complexity classes.

Applying the gadgets to suitable base graphs, this gives us a generalization of the rigid base construction to arbitrary abelian groups. However the difficulty of the graphs does not increase for the algorithms tested here as we will see in Section 6.

Since in various contexts abelian groups are algorithmically easier to handle than non-abelian groups, we would like to generalize the rigid base construction to non-abelian groups.

Semidirect products. Our next example is that of a semidirect product. Let $\Gamma = N \rtimes H \leq S_n$ be a semidirect product with an abelian normal subgroup N. For example Γ could be the dihedral group D_n. Then $\Delta := \{(ah, bh, ch) \in \Gamma^3 \mid a, b, c \in N,\ h \in H,\ abc = 1\}$ is a subgroup of $S_n \times S_n \times S_n$. Applying the gadget construction to suitable base graphs we obtain a family of new benchmark graphs that we call the *dihedral construction*.

Unentwined gadgets. As last example suppose $\Gamma_1 = H_2 \times H_3$, $\Gamma_2 = H_1 \times H_3$ and $\Gamma_3 = H_1 \times H_2$ with arbitrary finite groups H_i. Then the group $\Delta := \{((h_2, h_3), (h_1, h_3), (h_1, h_2))\}$ is a subgroup of $S_n \times S_n \times S_n$. We call these gadgets unentwined since they only create a pairwise interconnection between the classes. In particular, the gadgets are not beneficial for our cause since they do not force an interplay between all three classes C_1, C_2 and C_3.

5.2 Group theoretic restrictions on the gadgets

As discussed before, every group $\Delta \leq \mathrm{Sym}(C_1) \times \mathrm{Sym}(C_2) \times \mathrm{Sym}(C_3)$ can be realized by such a gadget X_Δ. Our intuition is that the isomorphism solvers can locally analyze a bounded size gadget and thus implicitly determine the automorphism group $\Delta = \mathrm{Aut}(X_\Delta)|_{C_1 \cup C_2 \cup C_3}$ of such a gadget. The goal must therefore be to encode the difficulty in the global dependency across the entire graph rather than in a local gadget.

Note that $\Delta \leq \pi_1(\Delta) \times \pi_2(\Delta) \times \pi_3(\Delta)$, where π_i is the projection on the i-th component. We say that Δ is a *subdirect product* of $\pi_1(\Delta) \times \pi_2(\Delta) \times \pi_3(\Delta)$. To understand what kind of gadgets can be constructed we should therefore investigate subdirect products. We call such a group $\Delta \leq \pi_1(\Delta) \times \pi_2(\Delta) \times \pi_3(\Delta)$ *2-factor injective* if each projection onto two of the factors is injective and *2-factor surjective* if each of these projections is surjective.

Intuitively, 2-factor injectivity says that two components determine the third. Thus when two different gadgets that are not 2-factor injective are attached (via parallel edges say) this may create local automorphism in the resulting graph, which we are trying to avoid. We are therefore interested in creating 2-factor injective gadgets. In fact, it is possible to restrict ourselves to 2-factor injective gadgets altogether, since we can quotient out the elements responsible for non-injectivity. It turns out however that the abelian example in the previous subsection are the only 2-factor injective and 2-factor surjective groups.

▶ **Lemma 6** ([18]). *Let $\Gamma = \Gamma_1 \times \Gamma_2 \times \Gamma_3$ be a finite group and Δ a subdirect product of Γ that is 2-factor surjective and 2-factor injective. Then Γ_1, Γ_2 and Γ_3 are isomorphic abelian groups and Δ is isomorphic to the subgroup of Γ_1^3 given by $\{(a, b, c) \in (\Gamma_1)^3 \mid abc = 1\}$, which in turn is isomorphic to $(\Gamma_1)^2$ as an abstract group.*

Thus, if we want to move beyond abelian groups, we cannot require 2-factor surjectivity. In general, however, it can be shown that every 2-factor injective group $\Delta \leq \pi_1(\Delta) \times \pi_2(\Delta) \times \pi_3(\Delta)$ is obtained by a combination of the three example constructions described in the previous subsection. Indeed, in [18] it is in particular shown that if one wants an entwined gadget then it is necessary to use an abelian group of the form described in Lemma 6. One can then take a finite extension similar to the semidirect product construction described above. We refer to [18] for more details. Overall we conclude that there are some group theoretic obstructions to using non-abelian gadgets, and are left with the intuition that the original construction leads to the most difficult examples.

6 Experimental Results

In the following we discuss experimental results for the constructions presented in this paper. The experiments were each performed on single node of a compute cluster with 2.00 GHz Intel Xeon X5675 processors. We always set a time limit of three hours (i.e., 10800 seconds) and the memory limit to 4 GB. Every single instance was processed once, but multiple instances were generated for each possible number of vertices with the same construction. We evaluated the following isomorphism solvers: `Bliss` version 0.72, `Nauty/Traces` version 25r9, `Saucy` version 3.0, and `Conauto` version 2.03.

All our instances consist of two graphs and the task for the algorithms was to check for isomorphism. For `Bliss` and `Nauty/Traces` this means that both graphs are canonized and the canonical forms are compared for equality. `Conauto` directly supports isomorphism testing and for `Saucy`, which only supports automorphism group computation, we compute the automorphism group of the disjoint union to check for isomorphism. In each series we performed the tests in increasing number of vertices with a time limit, which implied that once timeouts are reached repeatedly only a handful of further examples are computed.

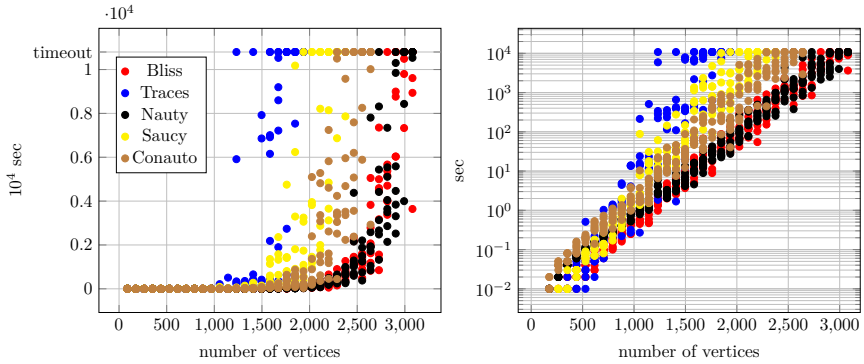

Figure 2 Performance of various algorithms on $R(B(G_n, \sigma))$ for random permutations σ in linear (left) and logarithmic scale (right).

We want to stress at this point that while the memory limit is irrelevant for `Bliss`, `Nauty`, `Saucy` and `Conauto`, it is significant for `Traces`. In fact, in many larger instances `Traces` reaches the memory limit before the time limit. For the sake of readability we do not distinguish between the two cases and in our figures runs that reached the memory limit are displayed as reaching the time limit. See [15] for details on the memory usage of `Traces`.

While all constructions pose difficult challenges for the solvers, those with more involved gadgets do not seem to yield more difficult examples. A reason for this effect could lie in the size of the gadgets. If the gadget is too large, asymptotic difficulty may emerge only for graphs with a number of vertices drastically larger than what could be tested. In any case, all of our constructions yield efficient methods to generate difficult isomorphism instances even with the size of the vertex set in regimes where no other difficult instances are available.

6.1 Shrunken Multipedes

Figure 2 shows the running times of the various isomorphism solvers on the multipede graphs described in Section 3 without any reductions. We observe that the multipede construction yields graphs which become infeasible for all solvers already for a few thousand vertices. Similarly, Figure 3 shows the running times on shrunken multipedes, i.e., the graphs to which the two node reductions of Section 4 have been applied. In comparison we observe that similar running times to the unreduced graphs are obtained already on graphs that have roughly half the number of vertices. While the shrunken version results in more difficult graphs one can also see that there is a significantly larger fluctuation (presumably depending on σ) among the graphs on a fixed number of vertices.

6.2 Comparison

We outline the main observations made in comparing our constructions with existing families of graphs. For a detailed comparison of the presented constructions among each other and to other families of difficult graphs we refer to the full version [19].

Reduction Techniques and other groups. We performed a series of experiments investigating the effect of the two reduction techniques from Section 4. Both reductions yield instances significantly more difficult and combining the two reductions yields the best results.

In experiments for the rigid base construction on other groups, we observe that the original construction based on $\mathbb{Z}_2$ yields the most difficult graphs.

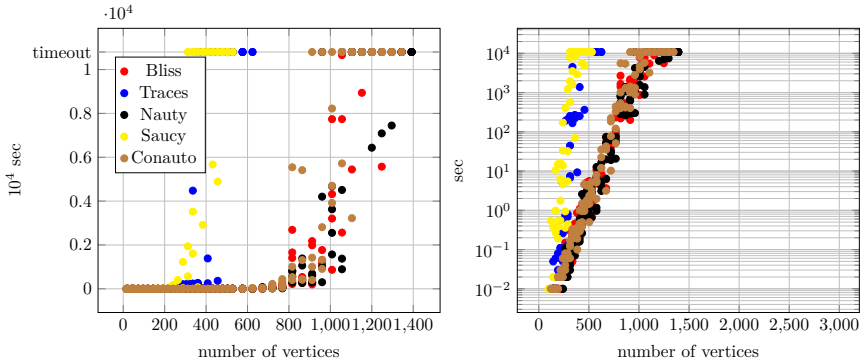

Figure 3 Performance of various algorithms on shrunken multipedes $R^*(B^*(G_n, \sigma))$ (both reductions applied) for random permutations σ in linear (left) and logarithmic scale (right).

Comparison with other benchmarks. Finally we compared the running times to benchmark graphs that previously existed. Since the existing benchmark graphs typically do not come in pairs of graphs we always take two copies, for each of which we randomly permute its vertices, and then perform an isomorphism test for the two copies.

We compared our graphs to the most difficult graphs from the library at [14]. We performed experiments for the families of Combinatorial graphs of Gordan Royle (combinatorial), Projective Planes of order 25 and 27 (pp), non-disjoint unions of tripartite graphs (tnn), Cai-Fürer-Immerman graphs (cfi), and Miyazaki graphs (mz-aug2). Starting from 1500 vertices our instances are several orders of magnitude more difficult on comparable input sizes.

7 Discussion

Variance. For each graph we tested each algorithm only once. Not all of the algorithms are deterministic and even for deterministic algorithms, there is also the question whether permuting the input graphs has any influence on the running times. We ran permutations of the same graph several times for each of the algorithms. Only saucy exhibits a non-negligible variance in some of the runs. However, even that variance is negligible in comparison to the exponential scaling of the running times on the benchmark graphs in the number of vertices.

Input size. The experiments we present all order the graphs according to the number of vertices. However, in practice one may be interested in sparse graphs, and the various algorithms are in particular tuned for this case. While all our graphs are sparse in that they only have a linear number of edges, average degrees vary. For shrunken multipedes we provided the coarse bound of 24 for the average degree. The multipedes have average degree 4.363, while the graphs obtained by only applying the linear algebra reduction ($R(B^*(G_n, \sigma))$) have average degree 4. It seems thus that when input size is measured in terms of the number of edges the graphs $R(B^*(G_n, \sigma))$ are the most difficult.

Conclusion. Overall we conclude that the new construction constitutes a simple algorithm that yields the most difficult benchmark graphs to date and experimentally we observe exponential behavior in terms of the running times of practical isomorphism solvers.

Acknowledgments. We thank Luis Núñez Chiroque, Tommi Junttila, Petteri Kaski, Brendan McKay, Adolfo Piperno, and José Luis López-Presa for numerous discussions related to graph isomorphism and for feedback on our benchmark instances.

References

1. László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *STOC*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
2. László Babai, Anuj Dawar, Pascal Schweitzer, and Jacobo Torán. The graph isomorphism problem (dagstuhl seminar 15511). *Dagstuhl Reports*, 5(12):1–17, 2015. doi:10.4230/DagRep.5.12.1.
3. László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. doi:10.1137/0209047.
4. László Babai and Youming Qiao. Polynomial-time isomorphism test for groups with abelian sylow towers. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 453–464. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.453.
5. Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.
6. Paolo Codenotti, Hadi Katebi, Karem A. Sakallah, and Igor L. Markov. Conflict analysis and branching heuristics in the search for graph automorphisms. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 907–914. IEEE Computer Society, 2013. doi:10.1109/ICTAI.2013.139.
7. Anuj Dawar and David Richerby. The power of counting logics on restricted classes of finite structures. In *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007. doi:10.1007/978-3-540-74915-8_10.
8. Yuri Gurevich and Saharon Shelah. On finite rigid structures. *J. Symb. Log.*, 61(2):549–562, 1996. doi:10.2307/2275675.
9. Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithms and Combinatorics*, pages 135–149. SIAM, 2007. doi:10.1137/1.9781611972870.13.
10. Ludek Kucera. Canonical labeling of regular graphs in linear average time. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 271–279. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.11.
11. José Luis López-Presa, Antonio Fernández Anta, and Luis Núñez Chiroque. Conauto-2.0: Fast isomorphism testing and automorphism group computation. *CoRR*, abs/1108.1060, 2011. URL: https://arxiv.org/abs/1108.1060.
12. Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.
13. Rudolf Mathon. Sample graphs for isomorphism testing. In *Proceedings of the ninth Southeastern Conference on Combinatorics, Graph Theory and Computing: Florida Atlanta University, Boca Raton, January 30-February 2, 1978*, Congressus Numerantium, 21. Winnipeg: Utilitas Mathematica Publish, 1978.
14. Brendan D. McKay and Adolfo Piperno. nautytraces software distribution web page. http://cs.anu.edu.au/~bdm/nauty/ and http://pallini.di.uniroma1.it.
15. Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

16 Takunari Miyazaki. The complexity of mckay's canonical labeling algorithm. In *Groups and Computation*, volume 28 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 239–256. DIMACS/AMS, 1995.

17 Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. `https://www.lii.rwth-aachen.de/research/95-benchmarks.html`.

18 Daniel Neuen and Pascal Schweitzer. Subgroups of 3-factor direct products, 2016. arXiv:1607.03444. URL: `https://arxiv.org/abs/1607.03444`.

19 Daniel Neuen and Pascal Schweitzer. Benchmark graphs for practical graph isomorphism. *CoRR*, abs/1705.03686, 2017. URL: `http://arxiv.org/abs/1705.03686`.

20 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. *CoRR*, abs/1705.03283, 2017. URL: `http://arxiv.org/abs/1705.03283`.

21 Yota Otachi and Pascal Schweitzer. Reduction techniques for graph isomorphism in the context of width parameters. In *SWAT*, volume 8503 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2014. `doi:10.1007/978-3-319-08404-6_32`.

22 Pascal Schweitzer. *Problems of unknown complexity: graph isomorphism and Ramsey theoretic numbers*. Phd. thesis, Universität des Saarlandes, Saarbrücken, Germany, 2009.

23 Jacobo Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33(5):1093–1108, 2004. `doi:10.1137/S009753970241096X`.

24 Faried Abu Zaid, Erich Grädel, Martin Grohe, and Wied Pakusa. Choiceless polynomial time on structures with small abelian colour classes. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 50–62. Springer, 2014. `doi:10.1007/978-3-662-44522-8_5`.