An Improved Isomorphism Test for Bounded-tree-width Graphs

MARTIN GROHE and DANIEL NEUEN, RWTH Aachen University, Germany
PASCAL SCHWEITZER, Technische Universität Kaiserslautern, Germany
DANIEL WIEBKING, RWTH Aachen University, Germany

We give a new FPT algorithm testing isomorphism of \( n \)-vertex graphs of tree-width \( k \) in time \( 2^k \text{polylog}(k) n^3 \), improving the FPT algorithm due to Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh (FOCS 2014), which runs in time \( 2^{O(k^5 \log k)} n^5 \). Based on an improved version of the isomorphism-invariant graph decomposition technique introduced by Lokshtanov et al., we prove restrictions on the structure of the automorphism groups of graphs of tree-width \( k \). Our algorithm then makes heavy use of the group theoretic techniques introduced by Luks (JCSS 1982) in his isomorphism test for bounded degree graphs and Babai (STOC 2016) in his quasipolynomial isomorphism test. In fact, we even use Babai’s algorithm as a black box in one place.

We also give a second algorithm that, at the price of a slightly worse running time \( 2^{O(k^2 \log k)} n^3 \), avoids the use of Babai’s algorithm and, more importantly, has the additional benefit that it can also be used as a canonization algorithm.

CCS Concepts: • Theory of computation → Fixed parameter tractability; • Mathematics of computing → Graph algorithms;

Additional Key Words and Phrases: Graph isomorphism, graph canonization, tree-width, decompositions

ACM Reference format:
Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. 2020. An Improved Isomorphism Test for Bounded-tree-width Graphs. ACM Trans. Algorithms 16, 3, Article 34 (May 2020), 31 pages.
https://doi.org/10.1145/3382082

1 INTRODUCTION

Already early on in the beginning of research on the graph isomorphism problem (which asks for structural equivalence of two given input graphs) a close connection to the structure and study of the automorphism group of a graph was observed. For example, Mathon [20] argued that the isomorphism problem is polynomially equivalent to the task of computing a generating set for the automorphism group and also to computing the size of the automorphism group.
With Luks’s polynomial time isomorphism test for graphs of bounded degree [19], the striking usefulness of group theoretic techniques for isomorphism problems became apparent and they have been exploited ever since (e.g., [5, 21, 22, 24]). In his algorithm, Luks shows and uses that the automorphism group of a connected graph of bounded degree, after a vertex has been fixed, has a very restricted structure. More precisely, the group is in the class $\Gamma_k$ of all groups whose composition factors are isomorphic to subgroups of the symmetric group $\text{Sym}(k)$.

Most recently, Babai’s quasipolynomial time algorithm for general graph isomorphism [2] adds several novel techniques to tame and manage the groups that may appear as the automorphism group of the input graphs.

A second approach towards isomorphism testing is via decomposition techniques (e.g., References [6, 11, 12, 14]). These decompose the graph into smaller pieces while maintaining control of the complexity of the interplay between the pieces. When taking this route it is imperative to decompose the graph in an isomorphism-invariant fashion so as not to compare two graphs that have been decomposed in structurally different ways.

A prime example of this strategy is Bodlaender’s isomorphism test [6] for graphs of bounded tree-width. Bodlaender’s algorithm is a dynamic programming algorithm that takes into account all $k$-tuples of vertices that separate the graph, leading to a running time of $O(n^{k+c})$ to test isomorphism of graphs of tree-width at most $k$.

Only recently, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [18] designed a fixed-parameter tractable isomorphism test for graphs of bounded tree-width running in time $2^{O(k^3 \log k)}n^3$. This algorithm first “improves” a given input graph $G$ to a graph $G^k$ by adding an edge between every pair of vertices between which more than $k$ pairwise internally vertex disjoint paths exist. The improved graph $G^k$ decomposes in an isomorphism-invariant manner along clique separators into clique-separator free parts, which we will call basic throughout the article. The decomposition can in fact be extended to an isomorphism-invariant tree decomposition into basic parts, as was shown in [10] to design a logspace isomorphism test for graphs of bounded tree-width. For the basic parts, Lokshtanov et al. [18] show that, after fixing a vertex of sufficiently low degree, it is possible to compute an isomorphism-invariant tree decomposition whose bags have a size at most exponential in $k$ and whose adhesion is at most $O(k^3)$. They use this invariant decomposition to compute a canonical form essentially by a brute-force dynamic programming algorithm.

The problem of computing a canonical form is the task to compute, to a given input graph $G$, a graph $G'$ isomorphic to $G$ such that the output $G'$ depends only on the isomorphism class of $G$ and not on $G$ itself.

The isomorphism problem reduces to the task of computing a canonical form: For two given input graphs, we compute their canonical forms and check whether the canonical forms are equal (rather than isomorphic).

As far as we know, computing a canonical form could be algorithmically more difficult than testing isomorphism. It is usually not very difficult to turn combinatorial isomorphism tests into canonization algorithms; sometimes the algorithms are canonization algorithms in the first place. However, canonization based on group-theoretic isomorphism tests often is more challenging.

**Our Results**

Our main result is a new FPT algorithm testing isomorphism of graphs of bounded tree-width.

**Theorem 1.** There is a graph isomorphism test running in time

$$2^k \text{polylog}(k)n^3,$$

where $n$ is the size and $k$ the minimum tree-width of the input graphs.

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
In the first part of the article, we analyze the structure of the automorphism group of a graph $G$ of tree-width $k$. Following References [18] and [10], we pursue a two-stage decomposition strategy for graphs of bounded tree-width, where in the first step, we decompose the improved graph along clique separators into basic parts. We observe that these basic parts are essential for understanding the automorphism groups. We show (Theorem 9) that with respect to a fixed vertex $v$ of degree at most $k$, we can construct for each basic graph $H$ an isomorphism-invariant tree decomposition of width at most $2^{O(k \log k)}$ and adhesion at most $O(k^2)$ where, in addition, each bag is equipped with a graph of small degree that is defined in an isomorphism-invariant way and gives us insight about the structure of the bag. In particular, using Luks’s results [19], this also restricts the structure of the automorphism group.

Our construction is based on a similar construction of an isomorphism-invariant tree decomposition in [18]. While Lokshatov et al. actually only work with an isomorphism-invariant set of bags, our algorithm also provides a corresponding tree decomposition. This allows us to improve the dependence on the input size to cubic running time. Also, we improve the adhesion-width (that is, the maximum size of intersections between adjacent bags of the decomposition) from $O(k^3)$ to $O(k^2)$. More importantly, we expand the decomposition by assigning a group and a graph to each bag.

Using these groups, we can prove that $\text{Aut}(H)_v$ (the group of all automorphisms of $H$ that keep the vertex $v$ fixed) is a $\Gamma_{k+1}$ group. This significantly restricts possible automorphism groups. Moreover, using the graph structure assigned to each bag, we can also compute the automorphism group of a graph of tree-width $k$ within the desired time bounds. The first already nontrivial step towards computing the automorphism group is a reduction from arbitrary graphs of tree-width $k$ to basic graphs. The second step reduces the problem of computing the automorphism group of a basic graph to the problem of computing the automorphism group of a structure that we call an expanded $d$-ary tree. In the reduction, the parameter $d$ will be polynomially bounded in $k$. Then, as the third step, we can apply a recent result [13], due to the first three authors, that allows us to compute the automorphism groups of such expanded $d$-ary trees. This result is heavily based on techniques introduced by Babai [2] in his quasipolynomial isomorphism test. In fact, it even uses Babai’s algorithm as a black box in one place.

We prove a second result that avoids the results of References [2, 13] and even allows us to compute canonical forms, albeit at the price of an increased running time.

**Theorem 2.** There is a graph canonization algorithm running in time

$$2^{O(k^2 \log k)} n^3,$$

where $n$ is the size and $k$ the tree-width of the input graph.

Even though it does not employ Babai’s new techniques, this algorithm still heavily depends on the group-theoretic machinery. As argued above, the design of group-theoretic canonization algorithms often requires extra work, and can be slightly technical, compared to the design of an isomorphism algorithm. Here, we need to combine the group-theoretic canonization techniques going back to Babai and Luks [5] with graph decomposition techniques, which poses additional technical challenges and requires new canonization subroutines.

**Organization of the article**

In the next section, we introduce the necessary preliminaries. The next two Sections 3–4 of the article deal with the decomposition of bounded tree-width graphs. They describe the isomorphism-invariant decomposition into basic parts and the isomorphism-invariant decomposition of the basic parts with respect to a fixed vertex of low degree.
The two following Sections 5–6 are concerned with isomorphism. We define a particular sub-
problem, coset-hypergraph-isomorphism, identified to be of importance. Then, we assemble the
recursive isomorphism algorithm.

In Section 7, we devise several subroutines for canonization in general and assemble the recursive
canonization algorithm.

2 PRELIMINARIES

Graphs. We use standard graph notation. All graphs \( G = (V, E) \) considered are undirected finite
simple graphs. We denote an edge \( \{u, v\} \in E \) by \( uv \). Let \( U, W \subseteq V \). We write \( E(U, W) \) for the edges
with one vertex in \( U \) and the other vertex from \( W \), whereas \( E(U) \) are the edges with both vertices
in \( U \). By \( N(U) \), we denote the neighborhood of \( U \), i.e., all vertices outside \( U \) that are adjacent to \( U \).
For the induced subgraph on \( U \), we write \( G[U] \), whereas \( G - U \) is the induced subgraph on \( V \setminus U \). A
rooted graph is a triple \( G = (V, E, r) \) where \( r \in V \) is the root of the graph. For two vertices \( v, w \in V \),
we denote by \( \operatorname{dist}_G(v, w) \) the distance between \( v \) and \( w \), i.e., the length of the shortest path from
\( v \) to \( w \). The depth of a rooted graph is the maximum distance from a vertex to the root; that is,
\[ \operatorname{depth}(G) = \max_{v \in V} \operatorname{dist}_G(r, v) \] (if \( G \) is not connected, then \( \operatorname{depth}(G) = \infty \)). The forward-degree of a
vertex \( v \in V \) is \( \phi(v) = |\{w \in N(v) \mid \operatorname{dist}(w, r) = \operatorname{dist}(v, r) + 1\}| \). Note that \( |V| \leq (d + 1)^\operatorname{depth}(G) \)
where \( d = \max_{v \in V} \phi(v) \) is the maximum forward-degree.

Separators. A pair \((A, B)\) where \( A \cup B = V(G) \) is called a separation if \( E(A \setminus B, B \setminus A) = \emptyset \). In this
case, we call \( A \cap B \) a separator. A separation \((A, B)\) is an \((L, R)\)-separation if \( L \subseteq A \) and \( R \subseteq B \),
and in this case \( A \cap B \) is called an \((L, R)\)-separator. A separation \((A, B)\) is 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.

Tree Decompositions.

Definition 3. A tree decomposition of a graph \( G \) is a pair \((T, \beta)\), where \( T \) is a rooted tree and
\( \beta : V(T) \to \Pow(V(G)) \) is a mapping into the power set of \( V(G) \) such that:

1. For each vertex \( v \in V(G) \), the set \( \{t \in V(T) \mid v \in \beta(t)\} \) induces a nonempty and connected
subtree of \( T \), and
2. For each edge \( e \in E(G) \), there exists \( t \in V(T) \) such that \( e \subseteq \beta(t) \).

The sets \( \beta(t) \) for \( t \in V(T) \) are called the bags of the decomposition, while the sets \( \beta(s) \cap \beta(t) \) for
\( st \in E(T) \) are called the adhesions sets. The width of a tree decomposition \( T \) is equal to its
maximum bag size decremented by one, i.e., \( \max_{t \in V(T)} |\beta(t)| - 1 \). The adhesion-width of \( T \) is equal
to its maximum adhesion size, i.e., \( \max_{st \in E(T)} |\beta(s) \cap \beta(t)| \). The tree-width of a graph, denoted by
tw(\( G \)), is equal to the minimum width of its tree decompositions.

A graph \( G \) is \( k \)-degenerate if every subgraph of \( G \) has a vertex with degree at most \( k \). It is well
known that every graph of tree-width \( k \) is \( k \)-degenerate. In particular, for a graph \( G \) of tree-width \( k \)
it holds that \( |E(G)| \leq k \cdot |V(G)| \).

Groups. For a function \( \varphi : V \to V' \) and \( v \in V \), we write \( v^\varphi \) for the image of \( v \) under \( \varphi \); that is,
\( v^\varphi = \varphi(v) \). We write composition of functions from left to right, e.g., \( v^{(\sigma \rho)} = (v^\sigma)^\rho = \rho(\sigma(v)) \). By
\([t]\), we denote the set of natural numbers from 1 to \( t \). By \( \Sym(V) \), we denote the symmetric group
on a set \( V \) and we also write \( \Sym(t) \) for \( \Sym([t]) \). We use upper-case Greek letters \( \Delta, \Phi, \Gamma, \Theta \) and
\( \Psi \) for permutation groups.

Labeling cosets. A labeling coset of a set \( V \) is a set of bijective mappings \( \tau \Delta \) where \( \tau \) is a bijection
from \( V \) to \( [|V|] \) and \( \Delta \) is a subgroup of \( \Sym(|V|) \). By \( \Label(V) \), we denote the labeling coset
\( \tau \Sym(|V|) \). We say that \( \tau \Delta \) is a labeling subcoset of a labeling coset \( \rho \Theta \), written \( \tau \Delta \leq \rho \Theta \), if \( \tau \Delta \) is
a subset of \( \rho \Theta \) and \( \tau \Delta \) forms a labeling coset again. Sometimes, we will choose a single symbol to
denote a labeling coset \( \tau \Delta \). For this, we will usually use the Greek letter \( \Delta \). Recall that \( \Gamma_k \) denotes the class of all finite groups whose composition factors are isomorphic to subgroups of \( \text{Sym}(k) \). Let \( \tilde{\Gamma}_k \) be the class of all labeling cosets \( \Lambda = \tau \Delta \) such that \( \Delta \in \Gamma_k \).

Orderings on sets of natural numbers. We extend the natural ordering of the natural numbers to finite sets of natural numbers. For two such sets \( M_1, M_2 \), we define \( M_1 < M_2 \) if \( |M_1| < |M_2| \) or if \( |M_1| = |M_2| \) and the smallest element of \( M_1 \setminus M_2 \) is smaller than the smallest element of \( M_2 \setminus M_1 \).

Isomorphisms. In this article, we will always define what the isomorphisms between our considered objects are. But the notion of an isomorphism can also be defined in a more general context. Let \( \varphi : V \to V' \). For a vector \((v_1, \ldots, v_k)\), we define \((v_1, \ldots, v_k)^\varphi = (v_1^\varphi, \ldots, v_k^\varphi)\) inductively. Analogously, for a set, we define \([v_1, \ldots, v_n]^\varphi = [v_1^\varphi, \ldots, v_n^\varphi] \). For a labeling coset \( \Lambda \leq \text{Label}(V) \), we write \( \Lambda^\varphi \) for \( \varphi^{-1} \Lambda \). In the article, we will introduce isomorphisms \( \text{Iso}(X, X') \) for various objects \( X \) and \( X' \). Unless otherwise stated, these are all \( \varphi : V \to V' \) such that \( X^\varphi = X' \) where we apply \( \varphi \) as previously defined. For example, the isomorphisms between two graphs \( G \) and \( G' \) are all \( \varphi : V \to V' \) such that \( G^\varphi = G' \), which means that \( G \) has an edge \( uv \) if and only if \( G' \) has the edge \( u^\varphi v^\varphi \).

3 CLIQUE SEPARATORS AND IMPROVED GRAPHS

To perform isomorphism tests of graphs of bounded tree-width, a crucial step in [18] is to deal with clique separators. For this step the concept of a \( k \)-improved graph is the key.

Definition 4 ([7, 8, 18]). 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 ([18]). 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 \((T, \beta)\) 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 \( O(k^2 n^3) \) time and either correctly concludes that \( \text{tw}(G) > k \), 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 and canonization. A \( k \)-basic graph has severe limitations concerning its structure, as we explore in the following sections. In the canonization algorithm from [18], a result of Leimer [16] is exploited that says that every graph has a tree decomposition into clique-separator free parts, and the family of bags is isomorphism-invariant. While it is usually sufficient to work with an isomorphism-invariant set of bags (see [23]), we actually require an isomorphism-invariant decomposition, which can indeed be obtained.

Theorem 6 ([10, 16]). For every connected graph \( G \) there is an isomorphism-invariant tree decomposition \((T, \beta)\) of \( G \), called clique separator decomposition, with the following properties:

1. For every \( t \in V(T) \) the graph \( G[\beta(t)] \) is clique-separator free (and in particular 2-connected).
2. Each adhesion set of \((T, \beta)\) is a clique.
3. \( |V(T)| \in O(|V(G)|) \) and \( \sum_{t \in V(T)} |\beta(t)| \in O(k_{\text{max}} \cdot |V(G)|) \) where \( k_{\text{max}} \) is the size of the largest clique in \( G \).
(4) For each bag $\beta(t)$ the adhesion sets to the children are all equal to $\beta(t)$ or the adhesion sets to the children are all distinct.

Moreover, given a graph $G$, the clique separator decomposition of $G$ can be computed in time $O(k_{\text{max}} \cdot |V(G)| \cdot |E(G)|)$ where $k_{\text{max}}$ is the size of the largest clique in $G$.

Proof Sketch. It is argued in [16] (see also [18]) that the set maximal clique separator free subgraphs $\mathcal{P}$ can be computed in time $O(|V(G)| \cdot |E(G)|)$. The maximal clique separator free subgraphs $\mathcal{P}$ will take the role of the bags $\beta(t)$ of the decomposition and they satisfy the size constraints required by the theorems. The adhesion sets will be minimal clique separators. To obtain an isomorphism-invariant decomposition, we follow the strategy of [10] to first compute tree decompositions of the maximal subgraphs without clique separators of size at most $k$ and assemble those to obtain tree decompositions of the maximal subgraphs without clique separators of size at most $k - 1$ and so on. In that paper, the analysis of the algorithm shows that it can be implemented in logarithmic space, implying polynomial run time. However, we need to analyze the running time more precisely, which we do next.

Overall, we need to define an isomorphism-invariant tree on $\mathcal{P}$. We proceed inductively. Suppose we have computed the collection $\mathcal{P}_i$ subgraphs that are maximal among all subgraphs without clique separators of size at most $i$. Also suppose we have inductively computed a tree decomposition $T_P$ of each such subgraph $P \in \mathcal{P}_i$ into basic parts. Using the techniques of [16] again, we can compute in time $O(|V(G)| \cdot |E(G)|)$ the collection $\mathcal{P}_{i-1}$ of maximal subgraphs without clique separators of size at most $i - 1$. Each $P \in \mathcal{P}_i$ is a subgraph in exactly one $P' \in \mathcal{P}_{i-1}$. To obtain a tree decomposition $T_{P'}$ for $P' \in \mathcal{P}_{i-1}$, we collect in a set $A_{P'}$ all $P \in \mathcal{P}_i$ with $P \subseteq P'$. We form the disjoint union $V(T_{P'}) := \bigcup_{P \subseteq A_{P'}} V(T_P)$ and set $\beta_P(t) = \beta_{P'}(t)$ for $t \in V(T_P)$. We need to explain under what conditions two nodes $t_1, t_2 \in V(T_{P'})$ are adjacent. Two nodes $t_1, t_2 \in V(T_P)$ that lie in the same $P \in A_{P'}$ are adjacent if they are adjacent in $T_P$. There will be an edge between some bag $t_1$ of $T_{P_1}$ and some bag $t_2$ of $T_{P_2}$ if $V(P_1)$ and $V(P_2)$ intersect in a clique of size $i$. Specifically, for each $j \in \{1, 2\}$ the bag $t_j \in V(T_{P_j})$ is chosen so that it satisfies $V(T_{P_1}) \cap V(T_{P_2}) \subseteq \beta(t_j)$ and so that it is the bag closest to the root having this property. As shown for example in [10], this defines a tree-decomposition for $P'$. Concerning the running time of the $i$th step, by iterating over all elements and checking in which $P \in \mathcal{P}_i$ they are contained, we can find all pairs $V(T_{P_1})$ and $V(T_{P_2})$ that intersect non-trivially in a set of size $i$ in time $O(|V(G)| \cdot k_{\text{max}} \cdot |V(G)|)$. The number of such pairs is in $O(|V(G)|)$, and for each pair, we can compute in time $O(k_{\text{max}} \cdot |V(G)|)$ the bags closest to the center containing the intersection. For both these running times, we use that $\sum_{P \in \mathcal{P}_i} |V(P)| \in O(k_{\text{max}} \cdot |V(G)|)$. We perform at most $k_{\text{max}}$ steps of the procedure so the overall running time is in $O(k_{\text{max}}^2 \cdot |V(G)| \cdot |E(G)|)$.

Finally, to achieve Property 4, we insert for each bag violating the property new children that group children with equal adhesion sets. \hfill \Box

4 DECOMPOSING BASIC GRAPHS

In this section, we shall construct bounded-width tree decompositions of $k$-basic graphs of tree-width at most $k$. Crucially, these decompositions will be isomorphism-invariant after fixing one vertex of the graph. Our construction refines a similar construction of [18].

Let us define three parameters $c_S$, $c_M$, and $c_L$ (small, medium, and large) that depend on $k$ as follows:

\[ c_S := 6(k + 1), \quad c_M := c_S + c_S(k + 1), \quad c_L := c_M + 2(k + 1) \left( \frac{c_M}{k + 2} \right)^2. \]
Note that $c_S \in O(k)$, $c_M \in O(k^2)$ and $c_L \in 2^{O(k \log k)}$. The interpretation of these parameters is that $c_M$ will bound the size of the adhesion sets and $c_L$ will bound the bag size. The parameter $c_S$ is used by the algorithm that in certain situations behaves differently depending on sets being larger than $c_S$ or not.

The bound $c_M \in O(k^2)$ improves the corresponding bound $c_M \in O(k^3)$ in [18]. However, the more significant extension of the construction in [18] is that in addition to the tree decomposition, we also construct both an isomorphism-invariant graph of bounded forward-degree and depth and an isomorphism-invariant $\Gamma_{k+1}$-group associated with each bag.

The weight of a set $S \subseteq V(G)$ with respect to a (weight) function $w : V(G) \rightarrow \mathbb{N}$ is $\sum_{v \in S} w(v)$. The weight of a separation $(A, B)$ is the weight of its separator $A \cap B$. For sets $L, R \subseteq V(G)$, among all $(L, R)$-separations $(A, B)$ of minimal weight there exists a unique separation with an inclusion-minimal $A$ (this follows from the submodularity of the vertex cut function, see, e.g., [9, Chapter 8]). For this separation, we call $A \cap B$ the leftmost minimal separator and denote it by $S_{L,R}(w)$. Moreover, we define $S_{L,R} = S_{L,R}(1)$ where 1 denotes the function that maps every vertex to 1.

For $U \subseteq V(G)$, we define a weight function $w_{U,k}$ such that $w_{U,k}(u) = k$ for all $u \in U$ and $w_{U,k}(v) = 1$ for all $v \in V(G) \setminus U$. Given a weight function $w : V(G) \rightarrow \mathbb{N}$, using Menger’s theorem and the Ford-Fulkerson algorithm, it is possible to compute $S_{L,R}(w)$. For such a weight function, the Ford-Fulkerson algorithm runs in time $O(|E(G)| \cdot w(S_{L,R}(w)) + |V(G)|)$. The following lemma generalizes Lemma 3.2 of [18]. Through this generalization, we obtain the adhesion bound $O(k^2)$ for our decomposition.

**Lemma 7.** Let $G$ be a graph, let $S \subseteq V(G)$, and let $\{T_i \subseteq V(G)\}_{i \in \{1\}}$ and $\{w_i : V(G) \rightarrow \mathbb{N}\}_{i \in \{1\}}$ be families where each $T_i$ is a minimum weight $(L_i, R_i)$-separator with respect to $w_i$ for some $L_i, R_i \subseteq S$. Let $w : V(G) \rightarrow \mathbb{N}$ be another weight function such that for all $i \in \{1\}$:

1. $w(v) = w_i(v)$ for all $v \in V(G) \setminus S$, and
2. $w(v) \geq w_i(v)$ for all $v \in V(G)$.

Let $D := S \cup \bigcup_{i \in \{1\}} T_i$. Suppose that $Z$ is the vertex set of any connected component of $G - D$. Then $w(N(Z)) \leq w(S)$.

**Proof.** We proceed by induction on $t$. For $t = 0$, we have $N(Z) \subseteq D = S$, so the claim is trivial. Assume $t \geq 1$ and define $D' := S \cup \bigcup_{i \in \{t-1\}} T_i$. Let $Z$ be the vertex set of a connected component of $G - D$, and let $Z' \supseteq Z$ be the vertex set of the connected component of $G - D'$ containing $Z$. Let $L_i, R_i \subseteq S$ be sets such that $T_i$ is a minimum weight $(L_i, R_i)$-separator with respect to $w_i$. Let $(A, B)$ be an $(L_i, R_i)$-separation with separator $A \cap B = T_i$. Without loss of generality, we may assume that $Z \subseteq A \setminus B$. The three sets $A \setminus T_i$, $T_i$, and $B \setminus T_i$ partition $V(G)$. Similarly, the three sets $V \setminus (Z \cup N(Z'))$, $N(Z')$, and $Z'$ partition $V(G)$. We define $Q_{i,j}$ to be the intersection of the $i$th set of the first triple with the $j$th set of the second triple. This way the sets $Q_{i,j}$ with $i, j \in \{1, 2, 3\}$ partition $V(G)$ into nine parts, as shown in Figure 1.

We have $w(N(Z')) \leq w(S)$ by the induction hypothesis. Since $N(Z') = Q_{1,2} \cup Q_{2,2} \cup Q_{3,2}$ and $N(Z) \subseteq Q_{1,2} \cup Q_{2,2} \cup Q_{3,2}$, it suffices to show $w(Q_{2,3}) \leq w(Q_{3,2})$. Observe that $Q_{2,1} \cup Q_{2,2} \cup Q_{3,2}$ is also an $(L_i, R_i)$-separator, because $L_i \subseteq A$ and $R_i \subseteq B$ by the choice of $(A, B)$, and $R_i \subseteq S \subseteq V(G) \setminus Z'$.

By the minimality of $T_i$, we have $w_{T_i}(T_i) \leq w_{T_i}(Q_{2,1} \cup Q_{2,2} \cup Q_{3,2})$, and as $T_i = Q_{2,1} \cup Q_{2,2} \cup Q_{3,2}$, this implies $w_{T_i}(Q_{2,3}) \leq w_{T_i}(Q_{3,2})$. By Assumptions (1) and (2) of the lemma, it follows that $w(Q_{2,3}) \leq w(Q_{3,2}) \leq w(Q_{3,2})$. \qed
Fig. 1. Sets appearing in the proof of Lemma 7.

The lemma can be used to extend a set of vertices $S$ that is not a clique separator to a set $D$ in an isomorphism-invariant fashion while controlling the size of the adhesion sets of the components of $G - D$. It will be important for us that we can also extend a labeling coset of $S$ to a labeling coset of $D$ and furthermore construct a graph of bounded forward-degree and depth associated with $D$ and $S$.

**Lemma 8.** Let $k \in \mathbb{N}$ and let $G$ be a graph that is $k$-improved. Let $S \subseteq V(G)$ and let $\Lambda \leq \text{Label}(S)$ be a labeling coset such that

1. $\emptyset \subsetneq S \subsetneq V(G)$,
2. $|S| \leq c_M$,
3. $S$ is not a clique,
4. $G - S$ is connected,
5. $S = N_G(V(G) \setminus S)$, and
6. $\Lambda \in \overline{\Gamma}_{k+1}^+$.

There is an algorithm that either correctly concludes that $\text{tw}(G) > k$ or finds a proper superset $D$ of $S$ and a labeling coset $\widehat{\Lambda} \leq \text{Label}(D)$ and a connected rooted graph $H$ with the following properties:

A. $D \supseteq S$,
B. $|D| \leq c_L$,
C. if $Z$ is the vertex set of any connected component of $G - D$, then $|N(Z)| \leq c_M$,
D. $\widehat{\Lambda} \in \overline{\Gamma}_{k+1}^+$,
E. $D \subseteq V(H)$, depth$(H) \leq k + 3$ and $\text{fdeg}(v) \in k^{O(1)}$ for all $v \in V(H)$.

The algorithm runs in time $2^{O(k \log k)} |V(G)|$ and the output $(D, \widehat{\Lambda}, H)$ is isomorphism-invariant (w.r.t. the input data $G, S, \Lambda$ and $k$).
Here, the output of an algorithm $\mathcal{A}$ is isomorphism-invariant if all isomorphisms between two input data $(G, S, \Lambda, k)$ and $(G', S', \Lambda', k')$ extend to an isomorphism between the output $(D, \Lambda, H)$ and $(D', \Lambda', H')$ (an isomorphism between $(G, S, \Lambda, k)$ and $(G', S', \Lambda', k')$ is a mapping $\varphi : V(G) \to V(G')$ such that $(G, S, \Lambda, k) \varphi = (G', S', \Lambda', k')$ where we apply $\varphi$ as defined in the preliminaries).

**Proof.** We consider two cases depending on the size of $S$.

Case $|S| \leq c_S$: Let $I := \{(\{x\}, \{y\}) : x, y \in S, x \neq y, xy \notin E(G)\}$ and let

$$D = S \cup \bigcup_{(L, R) \in I} S_{L,R}(w_{L \cup R, k+1}).$$

We set $w := w_{S,k+1}$ and then we have the following for every vertex set $Z$ of a connected component of $G - D$ by Lemma 7:

$$|N(Z)| \leq w(N(Z)) \leq w(S) \leq c_S(k+1) \leq c_M.$$

For every $xy \notin E(G)$ there is a $(\{x\}, \{y\})$-separator of size at most $k$ disjoint from $\{x, y\}$, because $G$ is $k$-improved. Thus, $|D| \leq |S| + k|S|^2 \leq c_S + k c_S^2 \leq c_1$. Moreover, since $G - S$ is connected and $S = N_G(V(G) \setminus S)$, for all distinct $x, y \in S$ every minimum weight $(\{x\}, \{y\})$-separator contains a vertex that is not in $S$. It follows that $D \neq S$.

The computation of a single $S_{L,R}(w_{L \cup R, k+1})$ can be done in time $O(|E(G)| \cdot k^2 + |V(G)|)$. Moreover, we can assume that $|E(G)| \leq k \cdot |V(G)|$; otherwise, we would conclude $tw(G) > k$. Hence, the set $D$ can be computed in time $O(k^3 \cdot |V(G)|)$.

Case $c_S < |S| \leq c_M$: Let $I := \{(L, R) : L, R \subseteq S, |L| = |R| \leq k + 2, |S_{L,R}| \leq k + 1\}$ and let

$$D = S \cup \bigcup_{(L, R) \in I} S_{L,R}.$$

The properties of $D$ follow from similar arguments as in the first case. The fact that $I$ is nonempty follows from the existence of a *balanced* separation (for details, see [18]). Also, the size of $I$ is bounded by $2^{O(k \log k)}$.

The computation of a single $S_{L,R}$ takes time $O(|E(G)| \cdot k + |V(G)|)$. As in the previous case, we can assume that $|E(G)| \leq k \cdot |V(G)|$. So, overall, the set $D$ can be computed in time $2^{O(k \log k)}|V(G)|$.

Next, we show how to find $\hat{\Lambda}$ in both cases. To each $x \in D \setminus S$, we associate the set $A_x := \{(L, R) \in I \mid x \in S_{L,R}\}$. Two vertices $x$ and $y$ occur in exactly the same separators if $A_x = A_y$. In this case, we call them equivalent and write $x \equiv y$. Let $A_1, \ldots, A_t \subseteq D \setminus S$ be the equivalence classes of "$\equiv$". Since each $x$ is contained in some separator of size at most $k + 1$, we conclude that the size of each $A_i$ is at most $k + 1$.

For each labeling $\lambda \in \text{Label}(S)$, we choose an extension $\hat{\lambda} : D \to \{1, \ldots, |D|\}$ such that $\hat{\lambda}|_S = \lambda$ and for $x, y \in D \setminus S$, we have $x^\lambda < y^\lambda$ if $A_x^\lambda < A_y^\lambda$ (recall that $<$ is the linear order of subsets of $\mathbb{N}$ as defined in the preliminaries). Inside each equivalence class $A_i$, the ordering is chosen arbitrarily. Define $\hat{\Lambda} := (|S|) \times \text{Sym}(A_1) \times \cdots \times \text{Sym}(A_t)) \cdot (\hat{\lambda} \mid \lambda \in \Lambda) \leq \text{Label}(D)$. By construction the coset $\hat{\Lambda}$ does not depend on the choices of the extensions $\hat{\lambda}$. Since $|A_i| \leq k + 1$ for all $1 \leq i \leq t$, we conclude that $\hat{\Lambda} \in \Gamma_{k+1}$, as desired.

It remains to explain how to efficiently compute $\hat{\Lambda}$. For this, we simply remark that it suffices to use a set of extensions $M \subseteq \Lambda$ such that $\Lambda$ is the smallest coset containing all elements of $M$ (i.e., we can use a coset analogue of a generating set). We conclude that $\hat{\Lambda}$ can be computed in polynomial time in the size of $I$. 

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
Last but not least, we show how to construct the graph $H$. The Case $|S| \leq c_S$ is easy to handle. In this case, we define $H$ as the complete graph on the set $D \cup \{r\}$ where $r$ is some new vertex, which becomes the root of $H$. The forward-degree of $r$ is bounded by $|D|$, which in turn is bounded by $k^{O(1)}$. We consider the Case $c_S < |S| \leq c_M$. We define $V(H) := \{(L, R) \mid L, R \subseteq S, |L| = |R| \leq k + 2\} \cup D$. Clearly, we have $I \subseteq V(H)$. For the root, we choose $(\emptyset, \emptyset) \in V(H)$. We define the edges $E(H) := \{(L, R)(L', R') \mid L \subseteq L', R \subseteq R', |L| + 1 = |R| + 1 = |L'| = |R'|\} \cup \{x(L, R) \mid x \in D, x \in S_{L,R}, (L, R) \in I\}$. Since for each pair $(L, R)$ there are at most $|S|^2$ different extensions $(L', R')$ with $|L| + 1 = |R| + 1 = |L'| = |R'|$ and, since each separator $S_{L,R}$ contains at most $k + 1$ vertices, we conclude that the forward-degree of each vertex in $H$ is bounded by $|S|^2 + k + 1 \in k^{O(1)}$. Moreover, the depth of $H$ is bounded by $k + 3$. The graph $H$ can be computed in time polynomial in the size of $V(H)$, which in turn is bounded by $2^{O(k \log k)}$.

A labeled tree decomposition $(T, \beta, \alpha, \eta)$ is a 4-tuple where $(T, \beta)$ is a tree decomposition and $\alpha$ is a function that maps each $t \in V(T)$ to a labeling coset $\alpha(t) \leq \text{Label}(\beta(t))$ and $\eta$ is a function that maps each $t \in V(T)$ to a graph $\eta(t)$.

The lemma can be used as a recursive tool to compute our desired isomorphism-invariant labeled tree decomposition.

**Theorem 9.** Let $k \in \mathbb{N}$ and let $G$ be a $k$-basic graph and let $v$ be a vertex of degree at most $k$. There is an algorithm that either correctly concludes that $tw(G) > k$ or computes a labeled tree decomposition $(T, \beta, \alpha, \eta)$ with the following properties:

1. the width of $(T, \beta)$ is bounded by $c_L$, 
2. the adhesion width of $(T, \beta)$ is bounded by $c_M$, 
3. the degree of $T$ is bounded by $kc_L^2$ and there are at most $k$ distinct children $t_1, \ldots, t_t$ of $t$ with common adhesion set (i.e., $\beta(t) \cap \beta(t_i) = \beta(t) \cap \beta(t_j)$ for all $i, j \in [t]$) for each $t \in V(T)$, 
4. $|V(T)| \in O(|V(G)|)$ and $\sum_{t \in V(T)} |\beta(t)| \in O(k \cdot |V(G)|)$, 
5. for each bag $\beta(t)$ the adhesion sets to the children are all equal to $\beta(t)$ or the adhesion sets to the children are all distinct. In the former case the bag size is bounded by $c_M$, 
6. for each $t \in V(T)$ the graph $\eta(t) = H_t$ is a connected rooted graph such that $\beta(t) \cup \beta(t)^2 \subseteq V(H_t)$ and for each adhesion set $S$ there is a corresponding vertex $S \in V(H_t)$, depth($H_t$) $\in O(k)$ and $\text{fdeg}(v) \in k^{O(1)}$ for all $v \in V(H_t)$, and 
7. $\alpha(t) \in \Gamma_{k+1}$.

The algorithm runs in time $2^{O(k^2 \log k)}|V(G)|^2$ and the output $(T, \beta, \alpha, \eta)$ of the algorithm is isomorphism invariant (w.r.t. $G$, $v$, and $k$). Furthermore, if we drop Property (VII) as a requirement, the triple $(T, \beta, \alpha, \eta)$ can be computed in time $2^{O(k \log k)}|V(G)|^2$.

Here, the output of an algorithm $\mathcal{A}$ is isomorphism-invariant if all isomorphisms between two input data extend to an isomorphism between the output. More precisely, an isomorphism $\varphi \in \text{Iso}((G, v, k), (G', v', k'))$ extends to an isomorphism between $(T, \beta, \alpha, \eta)$ and $(T', \beta', \alpha', \eta')$ if there is a bijection between the tree decompositions $\varphi_T : V(T) \rightarrow V(T')$ and for each node $t \in V(T)$ a bijection between the vertices of graphs $\varphi_t : V(\eta(t)) \rightarrow V(\eta'(\varphi_T(t)))$, which extends $\varphi$, i.e., $\varphi_t(x) = x^{\varphi}$ for all $x \in \beta(t) \cup \beta(t)^2 \cup 2^{\beta(t)}$ where we naturally apply $\varphi$ as defined in the preliminaries. Furthermore, these extensions define an isomorphism between the output data, i.e., for all nodes $t \in V(T)$, we have that $\beta(t)^{\varphi} = \beta'(\varphi_T(t))$, $\alpha(t)^{\varphi} = \alpha'(\varphi_T(t))$ and $\eta(\varphi_T(t)) = \eta'(\varphi_T(t))$.

**Proof.** We describe a recursive algorithm $\mathcal{A}$ that has $(G, S \subseteq V(G), \Lambda \leq \text{Label}(S))$ as the input such that the data satisfy the assumptions of Lemma 8. That is:
The output is a labeled tree decomposition \((T, \beta, \alpha, \eta)\) with \(S\) being a subset of the root bag. Since \(G\) has no separating cliques and therefore \(\nu\) is not a cut vertex, the triple \((G, \{\nu\}, \{\nu \mapsto 1\})\) fulfills the input conditions that will complete the proof. The algorithm \(\mathcal{A}\) works as follows:

**Step 1:** We describe how to construct \(\beta(t) = D\) and \(\alpha(t) = \tilde{\Lambda}\) for the root node \(t \in V(T)\). If \(S\) is a clique, we define \(D := S \cup \mathcal{N}(S)\). Notice that \(\mathcal{N}(S) \leq k\) due to Property 3. Let \(\tau : \mathcal{N}(S) \rightarrow \{1, \ldots, |D|\}\) be an arbitrary bijection. We define \(\tilde{\Lambda} := \Lambda \times \tau \text{Sym}(\{|1| + 1, \ldots, |D|\}) \leq \text{Label}(D)\). Otherwise (i.e., \(S\) is not a clique), we compute and define \(D\) and \(\tilde{\Lambda}\) as in Lemma 8.

Let \(Z_1, \ldots, Z_r\) be the connected components of the graph \(G - D\). Let \(S_i := N_G(Z_i)\) and \(G_i := G[Z_i \cup S_i]\).

**Step 2:** We describe how to construct the graph \(\eta(t) = H_t\). First, we compute the graph \(H\) as in Lemma 8. To achieve that pairs of vertices from \(\beta(t)\) are contained in this graph, we define a Cartesian product \(H^2\), i.e., \(V(H^2) := V(H)^2\) and

\[
E(H^2) := \{(u_1, u_2)(v_1, v_2) \mid u_1v_1 \in E(H) \text{ and } u_2v_2 \in E(H) \text{ and } u_1 = v_1\}. 
\]

Now, we define \(H_t\) as follows: The vertex set is \(V(H_t) := V(H) \cup V(H^2) \cup \{S_i \mid i \in [r]\}\). The edge set is \(E(H_t) := E(H^2) \cup \{(v, v) \mid v \in H\} \cup \{(u, v)S_i \mid u, v \in S_i, uv \notin E(D)\}\), and the root of \(H_t\) is \((r, r)\) where \(r\) is the root of \(H\).

**Step 3:** We call the algorithm recursively as follows: Let

\[
\Lambda_i := \{\lambda \in \tilde{\Lambda} \mid S_i^\lambda = \min_{\lambda \in \Lambda}[S_i^\lambda \mid \lambda \in \Lambda]\}. 
\]

(This set is essentially only an isomorphism-invariant restriction of \(\Lambda\) to \(S_i\). The minimum is taken with respect to \(<\), the ordering of subsets of \(N\) as defined in the preliminaries.) The image \(S_i^\lambda\) might be a set of natural numbers different from \([1, \ldots, |S_i|]\). To rectify this, we let \(\pi_i\) be the bijection from \(S_i^\lambda\) to \([1, \ldots, |S_i|]\) that preserves the normal ordering "\(<\" of natural numbers and set \(\Lambda'_i = \tilde{\Lambda} \times \pi_i\). We compute \(\mathcal{A}(G_i, S_i, \Lambda'_i) = (T_i, \beta_i, \alpha_i, \eta_i)\) recursively. By possibly renaming the vertices of the trees \(T_i\), we can assume w.l.o.g. \(V(T_i) \cap V(T_j) = \emptyset\).

**Step 4:** We define a labeled tree decomposition \((T, \beta, \alpha, \eta)\) as follows: Define a new root vertex \(t\) and attach the root vertices \(t_1, \ldots, t_r\) computed by the recursive calls as children. We define \(\beta(t) := D\) and \(\alpha(t) := \tilde{\Lambda}\) and \(\eta(t) := H_t\). Combine all decompositions \((T_i, \beta_i, \alpha_i, \eta_i)\) together with the new root \(t\) and return the resulting decomposition \((T, \beta, \alpha, \eta)\).

(Correctness) First, we show that \((G_j, S_j, \Lambda_j)\) fulfills the Conditions 1–6 and the recursive call in Step 3 is justified. Condition 1 is clear. For 2, notice that \(D\) and \(\tilde{\Lambda}\) satisfy (A)–(D) from Lemma 8 regardless of which option is applied in Step 1. Condition 2 then follows from (C). For 3, notice that \(G\) does not have any separating cliques, but \(S_j\) is a separator. The Conditions 4 and 5 follow from the definition of \(G_i\) and \(S_i\), whereas 6 follows from (D).

Next, we show (I)–(V). The bound for the width in (I) immediately follows from (B), while the bound on the adhesion width in (II) follows from (C). For (III), observe that \(G\) is \(k\)-improved and therefore each non-edge in \(D\) is contained in at most \(k\) different \(S_i\), i.e., \(|\{i \mid u, v \in S_i\}| \leq k\) for each \(u, v \in D, uv \notin E(D)\). Therefore, \(r \leq k|E(D)| \leq k\epsilon^2\).

Moreover, since each \(S_j\) contains a non-edge \(uv\), we conclude \(|\{i \mid S_i = S_j\}| \leq |\{i \mid u, v \in S_i\}| \leq k\). For (IV), we associate each bag \(\beta(t) = D\) with the

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
set \( D \setminus S \) in an injective way. The number of sets \( D \setminus S \) occurring in each recursive call is bounded by \( |V(G)| \), since these sets are not empty and pairwise-distinct. We conclude \( |V(T)| \leq |V(G)| \).

Property (VII) follows from (D). For any bag not satisfying Property (V), we introduce a new bag containing all equal adhesions. For (VI), we observe that the forward-degree of the vertices \((u, v) \in V(H^2)\) is the sum of the forward-degrees of \( u \) and \( v \) in \( H \). The depth of \( H^2 \) is twice the depth of \( H \). Therefore, we have \( \text{fdeg}(x) \in k^{O(1)} \) for each \( x \in V(H_t) \) and \( \text{depth}(H_t) \leq 2k + 7 \). The construction is obviously isomorphism-invariant.

(Running time.) We have already seen that the number of recursive calls is bounded by \( |V(G)| \). The computation of the sets \( D \) and \( \Lambda \) as in Lemma 8 can be done in time \( 2^{O(k \log k)} |V(G)| \). We need to explain how to compute \( \Lambda_t \) from \( \Lambda \). With standard group-theoretic techniques, including the Schreier-Sims algorithm, one can find a minimal image of \( S_t \). Using set transporters, one can find all labelings in \( \Lambda \) mapping to this minimal image. This algorithm runs in time \( |D|^{O(k)} \subseteq 2^{O(k^2 \log k)} \).

Since we are only interested in an isomorphism-invariant restriction of \( \Lambda \) to \( S_t \), we could also use techniques from Section 7 as follows: We define a hypergraph \( X \) on the vertex set \( D \) consisting of one single hyperedge \( S_t \). Let \( \Lambda_t := \text{CL}(X; \Lambda)|_{S_t} \) be the restriction of a canonical labeling of that hypergraph. This algorithm also runs in time \( |D|^{O(k)} \). This leads to a total run time of \( 2^{O(k^2 \log k)} |V(G)|^2 \). We remark that the computation of \( \Lambda_t \) from \( \Lambda \) is only needed if we require Property (VII). For the computation of a labeled tree decomposition \((T, \beta, \eta)\) the bottleneck arises in Lemma 8, which gives a run time of \( 2^{O(k \log k)} |V(G)|^2 \).

Remark 10. We later use the isomorphism-invariance of the labeled tree decomposition \((T, \beta, \alpha, \eta)\) from the previous theorem in more detail. Let \( t \in V(T) \) be a non-root node and \( S \subseteq V(G) \) the adhesion set to the parent node of \( t \) and let \( I_t = (T_t, \beta_t, \alpha_t, \eta_t) \) be the decomposition of the subtree rooted at \( t \) and \( G_t \) the graph corresponding to \( I_t \). Then \( \eta_t \) is isomorphism-invariant w.r.t. \( T_t, \beta_t, G_t, \) and \( S \).

5 COSET-HYPERGRAPH-ISOMORPHISM

After having computed isomorphism-invariant tree decompositions in the previous sections, we now want to compute the set of isomorphisms from one graph to another in a bottom-up fashion. Let \( G_1, G_2 \) be the two input graphs and suppose we are given isomorphism-invariant tree decompositions \((T_1, \beta_1)\) and \((T_2, \beta_2)\). For a node \( t \in V(T_i) \), we let \((G_i)_t\) be the graph induced by the union of all bags contained in the subtree rooted at \( t \). The basic idea is to compute for all pairs \( t \in V(T_i), t' \in V(T_j) \) the set of isomorphisms from \((G_i)_t\) to \((G_j)_{t'}\) (restricting to those isomorphisms that also respect the underlying tree decomposition) in a bottom-up fashion.

The purpose of this section is to give an algorithm that solves this problem at a given bag (assuming we have already solved the problem for all pairs of children of \( t \) and \( t' \)). Let us first give some intuition for this task. Suppose we are looking for all bijections from \( \beta_1(t) \) to \( \beta_2(t') \) that can be extended to an isomorphism from \((G_1)_t\) to \((G_2)_{t'}\). Let \( t_1, \ldots, t_{\ell} \) be the children of \( t \) and \( t'_1, \ldots, t'_{\ell'} \) the children of \( t' \). Then, we essentially have to solve the following two problems: First, we have to respect the edges appearing in the bags \( \beta_1(t) \) and \( \beta_2(t') \). But also, every adhesion set \( \beta(t) \cap \beta(t_i) \) has to be mapped to another adhesion set \( \beta(t') \cap \beta(t'_i) \) in such a way that the corresponding bijection (between the adhesion sets) extends to an isomorphism from \((G_1)_{t_i}\) to \((G_2)_{t'_i}\). To solve this problem, we first consider the case in which the adhesion sets are all distinct and define the following abstraction:

An instance of coset-hypergraph-isomorphism is a tuple \( I = (V_1, V_2, S_1, S_2, \chi_1, \chi_2, F, \xi) \) such that

1. \( S_i \subseteq \text{Pow}(V_i) \),
2. \( \chi_i : S_i \to \mathbb{N} \) is a coloring.
(3) \( \mathcal{F} = \{ \Theta_S \mid S \in \mathcal{S}_1 \} \) containing a group \( \Theta_S \leq \text{Sym}(S) \) for each \( S \in \mathcal{S}_1 \), and
(4) \( I = \{ \tau_{S_1, S_2} \mid S_1, S_2 \in \mathcal{S}_2 \text{ such that } \chi_1(S_1) = \chi_2(S_2) \} \) where \( \tau_{S_1, S_2} : S_1 \to S_2 \) such that
   (a) every \( \tau_{S_1, S_2} \in I \) is bijective, and
   (b) for every color \( i \in \mathbb{N} \) and every \( S_1, S'_1 \in \chi_1^{-1}(i) \) and \( S_2, S'_2 \in \chi_2^{-1}(i) \) and \( \theta \in \Theta_{S_1}, \theta' \in \Theta_{S_2} \) it holds that
   \[
   \theta\tau_{S_1, S_2}(\tau_{S_1, S_2})^{-1} \theta' \tau_{S_1, S_2} \in \Theta_{S_1}\tau_{S_1, S_2}\theta'.
   \]
   (N)

The instance is **solvable** if there is a bijective mapping \( \varphi : V_1 \to V_2 \) such that

1. \( S \in \mathcal{S}_1 \) if and only if \( S^\varphi \in \mathcal{S}_2 \) for all \( S \in \text{Pow}(V_1) \),
2. \( \chi_1(S) = \chi_2(S^\varphi) \) for all \( S \in \mathcal{S}_1 \), and
3. for every \( S \in \mathcal{S}_1 \) it holds that \( \varphi|_S \in \Theta_{S, S^\varphi} \).

In this case, we call \( \varphi \) an **isomorphism** of the instance \( I \). Moreover, let \( \text{Iso}(I) \) be the set of all isomorphisms of \( I \). Observe that property (N) describes a consistency condition: If we can use \( \sigma_1 \) to map \( S_1 \) to \( S_2 \), \( \sigma_2 \) to map \( S'_1 \) to \( S'_2 \), and \( \sigma_3 \) to map \( S'_1 \) to \( S'_2 \), then mapping \( \sigma_1 \sigma_2^{-1} \sigma_2 \) can be used to map \( S_1 \) to \( S'_2 \). As a result, the set of all isomorphisms of the instance \( I \) forms a coset; that is, \( \text{Iso}(I) = \Theta \varphi \) for some \( \Theta \leq \text{Sym}(V_1) \) and \( \varphi \in \text{Iso}(I) \).

In the application in the main recursive algorithm, the sets \( V_i = \beta(t_i) \), the hyperedges \( S_i \) are the adhesion sets of \( t_i \) (and we will also encode the edges appearing in the bag in this way), and the cosets \( \Theta_S, \tau_{S_1, S_2} \) tell us which mappings between the adhesion sets \( S_1 \) and \( S_2 \) extend to an isomorphism between the corresponding subgraphs. The colorings \( \chi_1 \) and \( \chi_2 \) are used to indicate which subgraphs can not be mapped to each other (and also to distinguish between the adhesion sets and edges of the bags that will both appear in the set of hyperedges).

The next lemma gives us one of the central subroutines for our recursive algorithm.

**Lemma 11.** Let \( I = (V_1, V_2, S_1, S_2, \chi_1, \chi_2, \mathcal{F}, I) \) be an instance of coset-hypergraph-isomorphism. Moreover, suppose there are isomorphism-invariant rooted graphs \( H_1 = (W_1, E_1, r_1) \) and \( H_2 = (W_2, E_2, r_2) \) such that

1. \( V_i \cup S_i \subseteq W_i \),
2. \( \text{deg}(w) \leq d \) for all \( w \in W_i \),
3. \( \text{depth}(H_i) \leq h \), and
4. \( |S| \leq d \) for all \( S \in S_i \)

for all \( i \in \{1, 2\} \). Then a representation for the set \( \text{Iso}(I) \) can be computed in time \( 2^{O(h \cdot \log d)} \) for some constant \( c \).

Here, isomorphism-invariant means that for every isomorphism \( \varphi \in \text{Iso}(I) \) there is an isomorphism \( \varphi_H \) from \( H_1 \) to \( H_2 \) such that \( \varphi^\varphi = \varphi^{\varphi_H} \) for all \( \varphi \in V_1 \) and \( |\varphi^\varphi| \mid \varphi \in S \rangle = S^{\varphi_H} \) for all \( S \in \mathcal{S}_1 \).

The proof of this lemma is based on the following theorem:

**Theorem 12 ([13]).** Let \( x, \gamma : V \to \Sigma \) be two strings, let \( \Gamma \leq \text{Sym}(V) \) be a \( \Gamma_d \) group, and \( \gamma \in \text{Sym}(V) \). Then one can compute a representation of the set of all permutations \( \gamma' \in \Gamma \gamma \) mapping \( x \) to \( \gamma \) in time \( n^{O((\log d)^c)} \) for some constant \( c \) where \( n := |V| \).

Actually, we shall only need the following corollary. A rooted tree \( T = (V, E, r) \) is \( d \)-ary if every node has at most \( d \) children. An expanded rooted tree is a tuple \( (T, C) \) where \( T = (V, E, r) \) is a rooted tree and \( C : L(T)^2 \to \text{rg}(C) \) is a coloring of pairs of leaves of \( T \) (\( L(T) \) denotes the set of leaves of \( T \)). Isomorphisms between expanded trees \( (T, C) \) and \( (T, C') \) are required to respect the colorings \( C \) and \( C' \).
Corollary 13. Let \( (T, C) \) and \( (T', C') \) be two expanded \( d \)-ary trees and let \( \Gamma \leq \text{Aut}(T) \) and \( \gamma \in \text{Iso}(T, T') \). Then one can compute a representation of the set \( \{ \varphi \in \Gamma_\gamma \mid (T, C)^\varphi = (T', C') \} \) in time \( n^{O((\log d)^c)} \) for some constant \( c \).

Proof. Without loss of generality assume that \( \text{rg}(C) = \text{rg}(C') \). Let \( \Gamma' \leq \text{Sym}(L(T))^2 \) be the group obtained from the natural action of \( \Gamma \) on the set \( L(T)^2 \) and similarly let \( \gamma' : L(T)^2 \to L(T)^2 \) be the bijection obtained from \( \gamma \). Note that \( \text{Aut}(T) \) is a \( \Gamma_d \)-group and thus, \( \Gamma' \) is also a \( \Gamma_d \)-group.

Using Theorem 12 and interpreting \( C \) and \( C' \) as strings, we can compute a representation of the set of all permutations \( \gamma' \in \Gamma' \) mapping \( C \) to \( C' \). From this, we can easily compute the desired set \( \{ \varphi \in \Gamma_\gamma \mid (T, C)^\varphi = (T', C') \} \). \( \square \)

Proof of Lemma 11. For \( i \in \{1, 2\} \) let \( H_i' \) be the graph obtained from \( H_i \) by adding, for each \( S_i \subseteq S_i \), vertices \( (S_i, v) \) for every \( v \in S_i \) connected to the vertex \( S_i \). Note that depth\( (H_i') \leq h + 1 \) and \( \text{fdeg}(w) \leq 2d \) for all \( w \in V(H_i') \). Let \( r_i \) be the root of \( H_i' \).

A branch of the graph \( H_i' \) is a sequence of vertices \( (v_0, \ldots, v_k) \) such that \( v_{j-1}, v_j \in E(H_i') \) for every \( j \in [k] \) and \( \text{dist}(r_i, v_j) = j \) for all \( j \in \{0, \ldots, k\} \) (in particular \( v_0 = r_i \)). We construct a rooted 2\( d \)-ary tree \( T_i \) as follows: Let \( V(T_i) \) be the set of all branches of \( H_i' \) and define \( E(T_i) \) to be the set of all pairs \( \overline{v}, \overline{w} \) such that \( \overline{v} = (v_0, \ldots, v_k) \) and \( \overline{w} = (w_0, \ldots, w_{k+1}) \). The root of \( T_i \) is \( (r_i) \). It is easy to check that \( |V(T_i)| \leq (2d + 1)^{h+1} \).

For a sequence of vertices \( \overline{v} = (v_1, \ldots, v_k) \) let last\( (\overline{v}) = v_k \) be the last entry of the tuple \( \overline{v} \). Let \( \chi_i^0 : V(T_i) \to \mathbb{N} \) be the coloring defined by \( \chi_i^0(\overline{v}) = 1 \) for all \( \overline{v} \) where last\( (\overline{v}) \in V_i \), \( \chi_i^0(\overline{v}) = 2 \) for all \( \overline{v} \) where last\( (\overline{v}) \in V(H_i') \setminus V(H_i) \), \( \chi_i^0(\overline{v}) = 3 + \chi_i(\text{last}(\overline{v})) \) for all \( \overline{v} \) where last\( (\overline{v}) \in S_i \), and \( \chi_i^0(\overline{v}) = 0 \) for all other tuples. Let \( \Gamma_i := \text{Aut}(T_i, \chi_i^0) \). Also let \( \gamma_i \to \gamma_i' \) be an isomorphism from \( (T_i, \chi_i^0) \) to \( (T_i', \chi_i') \) (if no such isomorphism exists, then \( \text{Iso}(T_i) = \emptyset \) and we are done).

Let \( A_i = \{ \overline{v} \in V(T_i) \mid \text{last}(\overline{v}) \in V(H_i') \setminus V(H_i) \} \). Note that every element \( \overline{v} \in A_i \) is a leaf of \( T_i \). Also note that \( A_i \) is invariant under \( \Gamma_i \); that is, \( A_i^\varphi = A_i \) for all \( \varphi \in \Gamma_i \). For \( \overline{v} \in A_i \), let \( \text{par}(\overline{v}) \) be the parent node of \( \overline{v} \) in the tree \( T_i \). Observe that last\( (\text{par}(\overline{v})) \in S_i \) for every \( \overline{v} \in A_i \). Let \( B_i = \{ \text{par}(\overline{v}) \mid \overline{v} \in A_i \} \). For \( \overline{w} \in B_i \) let \( A_i(\overline{w}) = \{ \overline{v} \in A_i \mid \text{par}(\overline{v}) = \overline{w} \} \) and let \( A_i'(\overline{w}) = \{ \text{last}(\overline{v}) \mid \overline{v} \in A_i(\overline{w}) \} = \{ S \} \times S \) where \( S = \text{last}(\overline{w}) \).

Now let \( \gamma \in \Gamma_1 \gamma_1 \to \gamma_2 \) be an isomorphism from \( (T_1, \chi_1^0) \) to \( (T_2, \chi_2^0) \). Then, for every \( \overline{w} \in B_1 \) the isomorphism \( \gamma \) induces a bijection \( \gamma_{\overline{w}} : \text{last}(\overline{w}) \to \text{last}(\overline{w}') \) that is induced by how \( \gamma \) maps \( A_1(\overline{w}) \) to \( A_2(\overline{w}') \). Now let

\[
\Gamma_1'(\gamma_1 \to \gamma_2) = \{ \gamma \in \Gamma_1 \gamma_1 \to \gamma_2 \mid \forall \overline{w} \in B_1 : \gamma_{\overline{w}} \in \Theta_{\text{last}(\overline{w})}(\text{last}(\overline{w}'), \text{last}(\overline{w})) \}.
\]

Observe that the set indeed forms a coset due to property (N). Moreover, a representation for \( \Gamma_1'(\gamma_1 \to \gamma_2) \) can be computed in time polynomial in \( |V(T_i)| \), since the group \( \Gamma_i \) acts independently as a symmetric group on all the sets \( A_i(\overline{w}) \).

Observe that, up to now, we have not enforced any consistency between the maps \( \gamma_{\overline{w}} \). Indeed, there may be elements \( \overline{w}, \overline{w}' \in B_1 \) such that \( \text{last}(\overline{w}) = \text{last}(\overline{w}') \) but \( \gamma_{\overline{w}} \neq \gamma_{\overline{w}'} \). To finish the proof, we need to enforce such a consistency property. Then every isomorphism \( \gamma \in \Gamma_1'(\gamma_1 \to \gamma_2) \) that fulfills the consistency property naturally translates back to an isomorphism of \( T \).

For \( v \in V(H_i') \), we define the label \( \ell(v) = v \) if \( v \in V(H_i) \) and \( \ell(v) = w \) if \( v = (S, w) \in V(H_i') \setminus V(H_i) \). Also, we define colorings \( C_i : L(T_i)^2 \to \mathbb{N} \) in such a way that for all \( \overline{v} = (v_0, \ldots, v_k), \overline{w} = (w_0, \ldots, w_{k+1}) \in L(T_i), \overline{w} = (w_0, \ldots, w_{k+1}) \in L(T_i) \)

\[
C_i(\overline{v}, \overline{w}) = C_j(\overline{w}, \overline{w}') \iff k = \ell \text{ and } k' = \ell'.
\]

and \( \forall i \in [k], j \in [k'] : \left( \ell(v_i) = \ell(v_j) \iff \ell(w_i) = \ell(w_j) \right) \)

and \( \overline{v} \in A_i \iff \overline{w} \in A_j \) and \( \overline{w} \in A_i \iff \overline{w} \in A_j \).
Now let 

$$\Gamma_1^{*} \gamma_{1-2} = \{ \gamma \in \Gamma_1^{*} \gamma_{1-2} \mid (T_1, C_2) \} = (T_2, C_2).$$

By Corollary 13 a representation for this coset can be computed in time $2^{O(h \cdot \log^c d)}$ for some constant $c$. We claim that every $\gamma \in \Gamma_1^{*} \gamma_{1-2}$ naturally defines an isomorphism of $I$ and conversely, every isomorphism of $I$ can be described like this: Let $V'_j = \{ \overline{v} \in V(T_i) \mid \text{last}(\overline{v}) \in V_j \}$. Let $\overline{v}, \overline{v}' \in V'_j$ such that last($\overline{v}$) = last($\overline{v}'$) and $\gamma \in \Gamma_1^{*} \gamma_{1-2}$. Then last($\overline{v}''$) = last($\overline{v}'''$) by properties (2) and (3) of the colorings $C_j$. Hence, every $\gamma \in \Gamma_1^{*} \gamma_{1-2}$ naturally defines a bijection $\gamma^{V_1} : V_1 \rightarrow V_2$.

Additionally, for $\overline{v} \in A_i$ and $\overline{v}' \in V'_j$ such that $\ell(\text{last}(\overline{v})) = \ell(\overline{v}')$ and $\gamma \in \Gamma_1^{*} \gamma_{1-2}$ it holds that last($\overline{v}''$) = last($\overline{v}'''$) by the same argument. With this, we obtain that $\gamma^{V_1}$ is indeed an isomorphism of $I$. The hyperedges $S_1$ and $S_2$ are encoded in the tree structure (and they have to be mapped to each other by the coloring $\chi^i_j$), and the coloring of hyperedges is encoded in the coloring $\chi^i_j$, which has to be preserved. Also, when mapping an element $\overline{w} \in B_1$ to another element $\overline{w}' \in B_2$ the children have to mapped to each other in such a way that the corresponding bijection lies in the corresponding coset. So, overall, $\gamma^{V_1}$ is an isomorphism of $I$.

However, every isomorphism $\varphi \in \text{Iso}(I)$ extends naturally to an isomorphism from $H_1 \rightarrow H_2$ and thus, it also extends to an isomorphism from $H'_1 \rightarrow H'_2$ and from $T_1 \rightarrow T_2$, since all relevant objects are defined in an isomorphism-invariant way. Hence, from a representation for $\Gamma_1^{*} \gamma_{1-2}$, we can easily compute a representation for $\text{Iso}(I)$.

Looking at the properties of the tree decompositions computed in Theorems 6 and 9, we have for every node $t$ that either the adhesion sets to the children are all equal or they are all distinct. Up to this point, we have only considered the problem that all adhesion sets are distinct (i.e., the coset-hypergraph-isomorphism problem). Next, we consider the case that all adhesion sets are equal. Towards this end, we define the following variant:

An instance of multiple-colored-coset-isomorphism is a 6-tuple $I = (V_1, V_2, \chi_1, \chi_2, F, t)$ such that

1. $\chi_1 : [t] \rightarrow \mathbb{N}$ is a coloring.
2. $\mathcal{F} = \{ \Theta_i \mid i \in [t] \}$ containing a group $\Theta_i \leq \text{Sym}(V_1)$ for each $i \in [t]$, and
3. $f = \{ \tau_{i,j} \mid i, j \in [t] \}$ such that $\chi_1(i) = \chi_2(j)$ where $\tau_{i,j} : V_1 \rightarrow V_2$ such that
   a. every $\tau_{i,j} \in f$ is bijective, and
   b. for every color $i \in \mathbb{N}$ and every $j_1, j'_1 \in \chi_1^{-1}(i)$ and $j_2, j'_2 \in \chi_2^{-1}(i)$ and $\theta \in \Theta_{j_1}, \theta' \in \Theta_{j'_1}$ it holds that
      $$\theta \tau_{j_1, j_2} \cdot \tau_{j_1, j'_2}^{-1} \tau_{j'_1, j_2} \cdot \tau_{j'_1, j'_2}^{-1} \in \Theta_{j_1} \cdot \Theta_{j'_1}.$$  (N2)

The instance is solvable if there is a bijective mapping $\varphi : V_1 \rightarrow V_2$ and a $\pi \in \text{Sym}(t)$ such that

1. $\chi_1(i) = \chi_2(\pi(i))$ for all $i \in [t]$, and
2. for every $i \in [t]$ it holds that $\varphi \in \Theta_i \tau_{i, \pi(i)}$.

The set $\text{Iso}(I)$ is defined analogously.

**Lemma 14.** Let $I = (V_1, V_2, \chi_1, \chi_2, F, t)$ be an instance of multiple-colored-coset-isomorphism. Then a representation for the set $\text{Iso}(I)$ can be computed in time

$$\min(2^{O(|V_1| \log |V_1|)|\mathcal{F}|^3}, 2^{O(|\mathcal{F}| \log |\mathcal{F}| + (\log |V_1|)^c)})$$

for some constant $c$.

**Proof.** The first run time can be achieved by brute force in the following way: We iterate through all $\varphi : V_1 \rightarrow V_2$ and check in polynomial time if a $\pi \in \text{Sym}(t)$ exists that satisfies (1) and (2). This can be done by computing, for every $i \in [t]$, the set $A_i$ of those $j \in [t]$ such that $\varphi \in \Theta_i \tau_{i,j}$
and $\chi_1(i) = \chi_2(j)$. Then one needs to find a permutation $\pi \in \text{Sym}(t)$ such that $\pi(i) \in A_i$ for all $i \in [t]$. This can be interpreted as a matching problem in a bipartite graph with $2t$ many vertices. A maximum matching in such a bipartite graph can be computed in time $O(t^3)$. The result is the union of all these $\varphi$.

However, for the second run time, we iterate through all $\pi \in \text{Sym}(t)$ satisfying (1) and compute all corresponding $\varphi : V_1 \to V_2$ satisfying (2), which in turn can be done by iterated coset-intersection. The result is $\text{Iso}(I) = \bigcup_{\pi \in \text{Sym}(t) \text{ satisfying (1)}} \bigcap_{i \in [t]} \Theta_i T_i \pi(i)$. Coset-intersection can be done in quasipolynomial time [2].

### 6 THE ISOMORPHISM ALGORITHM

**Lemma 15.** Let $G = (D, E)$ be a graph of tree-width at most $k$ and let $H$ be a rooted graph such that $D \subseteq V(H)$. Then, one can compute an isomorphism-invariant rooted graph $H'$ such that

1. $H$ is an induced subgraph of $H'$,
2. $\text{fdeg}_{\text{G}(H')}^>(w) \leq \max\{d, k+1\} + 1$ for all $w \in V(H')$ where $d = \max_{w \in V(H)} \text{fdeg}_{\text{G}(H')}^>(w)$,
3. $\text{depth}(H') \leq \text{depth}(H) + k + 2$, and
4. for every clique $C \subseteq D$ in the graph $G$ it holds that $C \in V(H')$.

In time $2^{O(k)} \cdot |V(H)|^{O(1)}$.

Here, isomorphism-invariant means that every isomorphism $\varphi \in \text{Iso}(H_1, H_2)$, which naturally restricts to an isomorphism from $G_1$ to $G_2$, can be extended to an isomorphism from $H_1'$ to $H_2'$ (which also maps the “clique vertices” in the natural way).

**Proof.** Let $G^{(0)} := G$ and for $i > 0$ let $G^{(i)} := G^{(i-1)}[V^{(i)}]$ where $V^{(i)} := \{v \in V(G^{(i-1)}) | \text{deg}_{G^{(i-1)}}^<(v) > k\}$. Since $G$ has tree-width at most $k$ it follows that there is some $i^* > 0$ such that $G^{(i^*)}$ is the empty graph. For every clique $C \subseteq D$, we define $i(C)$ to be the maximal $i \in \mathbb{N}$ such that $C \subseteq V(G^{(i)})$. Moreover, let $a(C) := C \setminus V(G^{(i(C))^+})$. Observe that $a(C) \neq \emptyset$ and for every $v \in a(C)$ it holds $\text{deg}_{G^{(i(C)}}^>(v) \leq k$ and $C \subseteq N_{G^{(i(C)}}^>(v)$.

For $v \in D$ let $i(v)$ be the maximal $i \in \mathbb{N}$ such that $\varphi \in V(G^{(i)})$. Let $A(v) := N_{G^{(i(v))}}^>(v)$. Note that $|A(v)| \leq k + 1$.

For a set $S$, we define the rooted graph $L_S$ to be the graph associated with the subset lattice of $S$; that is, $L_S = (\text{Pow}(S), E_S, \emptyset)$ where $AB \in E_S$ if $A \subseteq B$ and $|B \setminus A| = 1$. Now let $H'$ be the following graph: For every $v \in D$, we attach the graph $L_{A(v)}$ to the vertex $v \in V(H)$; that is, the root node $\emptyset \in V(L_{A(v)})$ is connected to the vertex $v$. It can be easily verified that $H'$ satisfies all requirements. Observe that $C \subseteq A(v)$ for every $v \in a(C)$ where $C$ is a clique of $G$. □

**Theorem 16.** There is an algorithm that, given $k \in \mathbb{N}$ and two connected graphs $G_1, G_2$, either correctly concludes that tw($G_1$) $>$ $k$ or decides whether $G_1$ is isomorphic to $G_2$ in time $2^{O(k \log k \log \gamma)} |V(G_1)|^3$ for some constant $c$.

**Proof.** We describe a dynamic programming algorithm $A$ that has an input $(I_1, I_2)$ where $I_j = (G_j, S_j, T_j, \beta_j, \eta_j)$ where $G_j$ is a graph, $(T_j, \beta_j)$ is a tree decomposition of the graph, $S_j \subseteq V(G_j)$ is a subset of the root bag, and $\eta_j$ is a (partial) function that assigns nodes $t_j \in V(T_j)$ a graph $\eta(t_j)$ that is isomorphism-invariant w.r.t. $T_{i,t_j}, \beta_{i,t_j}, G_{i,t_j}$ and $S_{i,t_j}$ as in Remark 10. The isomorphisms $\text{Iso}(I_1, I_2)$ are all isomorphisms $\varphi : V(G_1) \to V(G_2)$ from $(G_1, S_1)$ to $(G_2, S_2)$ for which there is a bijection between the nodes of the trees $\pi : V(T_1) \to V(T_2)$ such that for all $t \in V(T_1)$, we have that $\varphi(\beta_1(t)) = \beta_2(\pi(t))$. The algorithm computes a coset $A(I_1, I_2) = \{\varphi\mid \varphi \in \text{Iso}(I_1, I_2)\}$ (if $S_1 = \emptyset$, then the algorithm simply decides whether $\text{Iso}(I_1, I_2) \neq \emptyset$). In our recursive procedure, we maintain the following properties of the tree decomposition for each unlabeled bag $\beta_j(t_j)$ ($\eta_j(t_j)$):
(U1) The graph $G^k_j[\beta_j(t_j)]$ is clique-separator free,
(U2) Each adhesion set of $t_j$ and also $S_j$ are cliques in $G^k_j$, and
(U3) The adhesion sets to the children of $t_j$ are all equal to $\beta_j(t_j)$ or the adhesion sets to the children are all distinct.

And for each labeled bag $\beta_j(t_j)$, we require the following:

(L1) The cardinality $|\beta_j(t_j)|$ is bounded by $c_L$,
(L2) $\eta_j(t_j) = H$ is a connected rooted graph such that $\beta_j(t_j) \cup \eta_j(t_j) \subseteq V(H)$ and for each adhesion set $S$ it holds that $S \subseteq V(H)$, depth$(H) \in O(k)$ and $\text{fdeg}(v) \in kO(1)$ for all $v \in H$,
(L3) The adhesion sets to the children of $t_j$ are all equal to $\beta_j(t_j)$ or the adhesion sets to the children are all distinct,
(L4) If the adhesion sets are all equal and $\beta_j(t_j)$ is not a clique in $G^k_j$, then the cardinality $|\beta_j(t_j)|$ is bounded by $c_M$ and the number of children of $t_j$ is bounded by $k$.

The initial input of $A$ consists of the graphs $G_1, G_2$ together with their canonical (unlabeled) clique separator decompositions $(T_j, \beta_j)$ of their $k$-improved graphs $G^k_j$ from Theorem 6. For $S_j$, we choose the empty set. Since the clique separator decomposition is canonical, we have $\text{Iso}(I_1, I_2) = \text{Iso}(G_1, G_2)$ and therefore the algorithm decides whether $G_1$ is isomorphic to $G_2$.

Description of $A$:
For $j = 1, 2$ let $t_{j1}, \ldots, t_{jF}$ be the children of $t_j$ and let $(T_{ji}, \beta_{ji}, \eta_{ji})$ be the decomposition of the subtree rooted at $t_{ji}$. Let $G_{ji}$ be the graph corresponding to $(T_{ji}, \beta_{ji}, \eta_{ji})$. Let $V_{ji} := V(G_{ji})$ and let $S_{ji} := \beta_j(t_j) \cap V_{ji}$ be the adhesion sets between the children and the root node, let $Z_{ji}$ be $V_{ji} \setminus S_{ji}$, and let $I_{ji} := (G_{ji}, S_{ji}, t_{ji}, \beta_{ji}, \eta_{ji})$. We assume that the isomorphisms $A(I_{ji}, I_{j'f})$ have already been computed via dynamic programming.

We have to consider two cases depending on the root $t_j \in V(T_j)$.

Case $\beta_1(t_1)$ and $\beta_2(t_2)$ are both unlabeled: We distinguish two further subcases. First suppose the adhesion sets to the children of $t_j$ are all equal to $\beta_j(t_j)$. In this case $\beta_j(t_j)$ is a clique and thus, $|\beta_j(t_j)| \leq k + 1$. We encode the isomorphism problem $\text{Iso}(I_1, I_2)$ as an instance of multiple-colored-cost-isomorphism as follows: Let $I = (\beta(t_1), \beta(t_2), \chi_1, \chi_2, F, t)$ where $\chi_1(i) = \chi_2(i')$ if and only if $A(I_{1i}, I_{j'f}) = \emptyset$ for $i, i' \in [f]$ and $F, t$ encode the isomorphisms between the children of $t_1$ and $t_2$ (restricted to the adhesion sets). Additionally, the sets $S_j, j = 1, 2$, are also encoded in the instance $I$. Then $\text{Iso}(I)$ is computed using Lemma 14.

We return $A(I_1, I_2) = \{\varphi|_{S_j} | \varphi \in \text{Iso}(I)\}$.

In the other case, all adhesion sets are distinct. Let $v_1$ be a vertex in $\beta_1(t_1)$ of degree at most $k$ in $G^k_1[\beta_1(t_1)]$ and compute the canonical labeled tree decomposition $(T'_1, \beta'_1, \eta'_1)$ of $G^k_1[\beta_1(t_1), v_1)$ from Theorem 9. For each vertex $v_2 \in \beta_2(t_2)$ of degree at most $k$ in $G^k_2[\beta_2(t_2)]$, we compute the canonical labeled tree decomposition $(T'_2, \beta'_2, \eta'_2)$ of $G^k_2[\beta_2(t_2), v_2)$ from Theorem 9. Notice that the bags of $(T'_j, \beta'_j, \eta'_j)$ satisfy (L1)–(L4), because in the case $\text{tw}(G_j) \leq k$, the $k$-improvement does not increase the tree-width as seen in Lemma 5. Notice that the adhesion sets $S_{ji}$ and $S_j$ are cliques in $G^k_j$ and these cliques must be completely contained in one bag. We attach the children $(T_{ji}, \beta_{ji}, \eta_{ji})$ to $(T'_j, \beta'_j, \eta'_j)$ by adding them to the highest possible bag and we choose a new root as the highest possible node $r_j \in V(T'_j)$ such that $S_j \subseteq \beta'_j(r_j)$. By doing this, we obtain $(T''_j, \beta''_j, \eta''_j)$. We need to recompute $\eta''_j(s_j)$ for all nodes $s_j \in V(T_j)$ where new children are attached to preserve Property (L2). We call the algorithm in Lemma 15 with the input $(G_j[\beta_j(s_j)], \eta''_j(s_j))$ and obtain $\eta'''_j(s_j)$. Finally, we obtain $(T''_j, \beta''_j, \eta'''_j)$ and define $I'_j := (G, S, T''_j, \beta''_j, \eta'''_j)$. We show
that (L1)–(L4) remains satisfied. By possibly introducing new bags, we can preserve Property (L3): The adhesion sets are either pairwise-different or all equal. We can preserve Property (L4). Consider a bag $s_j$ in which all adhesion sets are equal and where $\beta''(s_j)$ is not a clique in $G^k$. Then the children are not any of the attached children $t_1, \ldots, t_j$ and therefore the number of children of $s_j$ remains bounded by $k$.

For each of the vertices $v_2 \in \beta_s(t_2)$, we compute the isomorphisms $\mathcal{A}(I'_1, I'_2(v_2))$ recursively. We return the smallest coset that contains the union $\bigcup_{v_2 \in \beta_s(t_2)} \mathcal{A}(I'_1, I'_2(v_2))$. Case $\beta_1(t_1)$ and $\beta_2(t_2)$ are both labeled: We consider two cases depending on whether the adhesion sets of $t_j$ are all different or not.

If all adhesion sets are different, we use the algorithm from Lemma 11 as follows: We encode the isomorphism problem $\text{Iso}(I_1, I_2)$ as an instance of coset-hypergraph-isomorphism. First, we encode the tree decomposition as

$$I := (\beta_1(t_1), \beta_2(t_2), \{S_1\}, \{S_2\}, \chi_1, \chi_2, \mathcal{F}, \tau)$$

and $H_1 := \eta_1(t_1), H_2 := \eta_2(t_2)$ where $\mathcal{F}, \tau$ are the isomorphisms between the children of $t_1$ and the children of $t_2$ restricted to their adhesion sets. We can easily construct $\mathcal{F}, \tau$ with the isomorphisms $\mathcal{A}(I_1, I'_2)|_{\chi_1}$ that have already been computed. The colorings $\chi_1$ are defined as follows: Two $S_1$ and $S_2$ get assigned the same color (i.e., $\chi_1(S_1) = \chi_2(S_2)$) if and only if the set of isomorphisms $\mathcal{A}(I_1, I'_2)$ is not empty. Since $\mathcal{F}, \tau$ consist of isomorphisms, the consistency property (N) is satisfied. Next, we encode the edge relation of edges contained in the root bag as $I' = (\beta_1(t_1), \beta_2(t_2), E(\beta_1(t_1)) \cup \{S_1\}, E(\beta_2(t_2)) \cup \{S_2\}, \chi_1', \chi_2', \mathcal{F}', \tau')$. To construct $\mathcal{F}', \tau'$, we define for all edges $e_1 \in E(\beta_1(t_1)), e_2 \in E(\beta_2(t_2))$ the allowed isomorphisms $\Theta_{e_1, e_2}$ as the set of bijections between $\beta_1(t_1)$ and $\beta_2(t_2)$ that map $e_1$ to $e_2$. Moreover, the set of allowed isomorphisms $\Theta_{S_1, S_2}$ is the set of bijections between $\beta_1(t_1)$ and $\beta_2(t_2)$ that map $S_1$ to $S_2$. We define $\chi_1', \chi_2'$ as a coloring mapping all edges to the integer 0 and mapping $S_1$ and $S_2$ to the integer 1. Finally, we combine both instances $I$ and $I'$ to obtain $I''$ and compute the isomorphisms $\text{Iso}(I'') = \text{Iso}(I) \cap \text{Iso}(I')$ by the algorithm in Lemma 11. We restrict the isomorphisms to the sets $S_1$ and $S_2$ return $\mathcal{A}(I_1, I_2) = \{\varphi|_{S_1} \mid \varphi \in \text{Iso}(I'')\}$.

If all adhesion sets are equal, the procedure is analogous, but we compute $\text{Iso}(I'')$ by calling the algorithm from Lemma 14.

(Correctness.) We consider the first case. Since each isomorphism from $I_1$ to $I_2$ maps the vertex $v_1$ to some vertex $v_2 \in \beta_s(t_2)$, we conclude that $\text{Iso}(I_1, I_2) = \bigcup_{v_2 \in \beta_s(t_2)} \mathcal{A}(I_1, I_2(v_2))$.

We consider the second case. We show that $\{\varphi|_{S_1} \mid \varphi \in \text{Iso}(I_1, I_2)\} = \mathcal{A}(I_1, I_2)$. Let $\varphi \in \text{Iso}(I_1, I_2)$ be an isomorphism, i.e., $\varphi$ is an isomorphism from $(G_1, S_1)$ to $(G_2, S_2)$ for which there is a bijection between the nodes of the trees $\pi : V(T_1) \to V(T_2)$ such that for all $t \in V(T_1)$, we have that $\varphi(\beta_1(t)) = \beta_2(\pi(t))$. Equivalently, since each $\pi$ maps the root of $T_1$ to the root of $T_2$, we have that $\varphi|_{\beta_1(t_1)} \in \text{Iso}(\{G_1[\beta_1(t_1)], S_1\}, \{G_2[\beta_2(t_2)], S_2\})$ and there is a bijection $\pi : \text{Sym}(\ell)$ between the children of $t_1$ and $t_2$ such that $\varphi|_{V(G_{\ell})} \in \text{Iso}(I_1, I_2(\ell))$. The condition $\varphi|_{\beta_1(t_1)} \in \text{Iso}(\{G_1[\beta_1(t_1)], S_1\}, \{G_2[\beta_2(t_2)], S_2\})$ is equivalent to $\varphi|_{\beta_1(t_1)} \in \text{Iso}(I')$. Therefore, $\{\varphi|_{S_1} \mid \varphi \in \text{Iso}(I_1, I_2)\} = \mathcal{A}(I_1, I_2)$.

(Running time.) Let $(T_1, \beta_1)$ be the clique separator decomposition of the $k$-improved graph $G^k_j$ from Theorem 6 computed in the initial step of the algorithm for both $j = 1, 2$. The computation of $G^k_j$ can be done in time $O(k^2 n^3)$ by Lemma 5. Moreover, the computation of $(T_1, \beta_2), (T_2, \beta_2)$ can be performed in time $O(k^2 mn) \subseteq O(k^2 n^2)$ by Theorem 6. Let $s_j := \sum t \in V(T_j)(\beta_j(t))$ for both $j = 1, 2$. We analyze the running time of the algorithm in terms of $s_1$ and $s_2$. Note that $s_j \in O(kn)$ by Theorem 6 where $n := |V(G_1)| = |V(G_2)|$. 

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
Let $t_U(s_1, s_2)$ denote the running time of the algorithm assuming the root bags are unlabeled. Also, let $s_{ji}, i \in [\ell]$, denote the size of the decomposition rooted at $t_{ji}$, i.e., $s_{ji} := \sum t \in V(T_{ji}) |\beta_j(t)|$. Note that $s_j = |\beta_j(t_j)| + \sum_{i \in [\ell]} s_{ji}$. First suppose the adhesion sets to the children of $t_j$ are all equal to $\beta(t_j)$. In this case

$$t_U(s_1, s_2) = \sum_{i=1}^{\ell} \sum_{i' = 1}^{\ell} t_U(s_{1i}, s_{2i'}) + O\left(2^{O(k \log k)} \ell^2\right)$$

by Lemma 14. To analyze the second case, define $t_L(s_1, s_2)$ to be the running time of the algorithm assuming the root bags are labeled. Then

$$t_U(s_1, s_2) = \sum_{i=1}^{\ell} \sum_{i' = 1}^{\ell} t_U(s_{1i}, s_{2i'}) + O\left(2^{O(k \log k)} |\beta_2(t_2)|^2 + 2^{O(k \log k)} |\beta_2(t_2)| + t_L(s_1', s_2')\right).$$

The first part of the sum simply describes the cost of recursively computing isomorphisms between all children of $t_1$ and $t_2$. The second part of the sum describes the running time of the algorithm for an unlabeled root node in the case where the adhesion sets to the children are all distinct. For each $v_2 \in \beta_2(t_2)$ of degree at most $k$ in $G_2^k[\beta_2(t_2)]$ the following steps are performed. First, the algorithm applies Theorem 9 to the pairs $(G_1^k[\beta_1(t_1)], v_1)$ and $(G_2^k[\beta_2(t_2)], v_2)$, which requires time $2^{O(k \log k)} |\beta_2(t_2)|^2$ (assuming $|\beta_1(t_1)| = |\beta_2(t_2)|$), otherwise the instances are not isomorphic. Then the children $(T_{ji}, \beta_{ji}, \eta_{ji})$ are attached to $(T'_j, \beta'_j, \eta'_j)$ by adding them to the highest possible bag. This can be done in time $2^{O(k \log k)} |\beta_2(t_2)|$. Updating a single graph $H = \eta''(s_j)$ in the case where the clique separators were attached is done by Lemma 15 in time $2^{O(k \log k)|V(H)|}$. Notice that the depth of $H$ is bounded by $O(k)$ and the maximum forward-degree is bounded by $k^{O(1)}$. This gives a bound on the number of vertices $|V(H)| \in 2^{O(k \log k)}$. Since the algorithm needs to update at most $\ell$ graphs, all updates can be performed in time $2^{O(k \log k) \ell}$. Afterwards, the solution is computed recursively, which requires time $t_L(s_1', s_2')$. Note that, due to dynamic programming, the algorithm only needs to recursively process the labeled tree decomposition $(T'_j, \beta'_j, \eta'_j)$ computed for the pair $(G_2^k[\beta_2(t_j)], v_j)$. Hence, $s_j' := \sum t \in V(T'_j) |\beta'_j(t)| = O(k \cdot |\beta_j(t_j)|)$. Finally, we need to compute the smallest coset that contains the union $\bigcup_{v_2 \in \beta_2(t_2)} A(I'_1, t'_2(v_2))$. This can be done in time $|S_2|^{O(1)} |\beta_2(t_2)| \subseteq k^{O(1)} |\beta_2(t_2)|$ (see Section 7.1 for more details).

It remains to analyze the recurrence obtained in the case where both root bags are labeled. As before, we first compute the solution for all pairs of children of $t_1$ and $t_2$. Observe that, by the comment above, it suffices to consider the children whose bags are labeled.

Consider the second case in which all adhesion sets are different. The isomorphism subroutine in Lemma 11 runs in time $2^{O(k \log k)}$. So, consider the second case in which all adhesion sets are equal. Then $\beta_j(t_j)$ is not a clique in $G_j^k$. Thus, by Property (L4), the cardinality $|\beta_j(t_j)|$ is bounded by $c_M$ and the number of children of $t_j$ is bounded by $k$. In this case, we have that $|F| \leq k$ for our subroutine, which leads to a running time $2^{O(|F| \log |F| + k \log c(|\beta_1(t_1)|))} \subseteq 2^{O(k \log k)}$.

So, overall,

$$t_L(s_1, s_2) = \sum_{i=1}^{\ell} \sum_{i' = 1}^{\ell} t_L(s_{1i}, s_{2i'}) + 2^{O(k \log k)}.$$
Then \( t_L(s_1, s_2) \in 2^{O(k(\log k)^c)} \cdot \ell^2 \). So,

\[
    t_U(s_1, s_2) = \sum_{i=1}^{\ell} \sum_{i=1}^\ell t_U(s_{1i}, s_{2i}) + 2^{O(k(\log k)^c)}|\beta_2(t_2)|^2 \cdot (|\beta_2(t_2)| + \ell) + O\left(2^{O(k(\log k) \ell^3)}\right).
\]

This recurrence described resolves to a running time \( 2^{O(k(\log k)^c)} \cdot \ell^3 \) as desired.

\[\square\]

**Remark 17.** While the previous theorem only decides whether \( G_1 \) is isomorphic to \( G_2 \), the algorithm can also be used to compute an isomorphism if \( G_1 \) is isomorphic to \( G_2 \). Indeed, by a standard argument, an algorithm can perform a second pass over the tree decomposition to recover an isomorphism proving \( G_1 \) is isomorphic to \( G_2 \).

## 7 Canonization

In this section, we adapt our techniques to obtain an algorithm that computes a canonical form for a given graph of tree-width at most \( k \). Since the isomorphism test for graphs of bounded degree given in [13] cannot be used for canonization purposes, our canonization algorithm has a slightly worse running time. However, our canonization algorithm still significantly improves on the previous best due to Lokshchanov, Pilipczuk, Pilipczuk, and Saurabh [18].

One of the main tasks for building the canonization algorithm out of our isomorphism test is to adapt Lemma 11. To achieve this, we shall use several group theoretic algorithms concerned with canonization. We start by giving the necessary background.

### 7.1 Background on Canonization with Labeling Cosets

Let \( G \) and \( H \) be two graphs such that \( V(G) = V(H) = [n] \) is an initial segment of the natural numbers. Also let \( \Delta \leq \text{Sym}(n) \). We say \( G \) and \( H \) are isomorphic with respect to \( \Delta \), denoted by \( G \equiv_\Delta H \), if there is a bijection \( \delta \in \Delta \) such that \( G^\delta = H \). In this case, we write \( G \equiv_\Delta H \). By \( \text{Aut}_\Delta(G) \), we denote the automorphism group of a graph \( G \) restricted to \( \Delta \), i.e., all \( \delta \in \Delta \) such that \( G^\delta = G \).

In the following, let \( \mathcal{X} \) denote a class of graphs with vertex set \([n]\) for some fixed number \( n \), and suppose \( \mathcal{X} \) is closed under applying permutations (i.e., \( G^\delta \in \mathcal{X} \) for all \( G \in \mathcal{X} \) and \( \delta \in \text{Sym}(\{1, \ldots, n\}) \)).

**Definition 18.** A function \( CF_\Delta : \mathcal{X} \to \mathcal{X} \) is a canonical form with respect to a group \( \Delta \leq \text{Sym}(n) \) for \( \mathcal{X} \) if \( CF_\Delta(G) \equiv_\Delta G \) for all \( G \in \mathcal{X} \), and \( G \equiv_\Delta H \) implies \( CF_\Delta(G) = CF_\Delta(H) \) for all \( G, H \in \mathcal{X} \).

**Definition 19.** Let \( CF_\Delta \) be a canonical form for \( \mathcal{X} \) for some group \( \Delta \leq \text{Sym}(n) \). The canonical labelings of a graph \( G \in \mathcal{X} \) with respect to \( CF_\Delta \) and \( \Delta \) are the elements in \( \Delta \) that bring \( G \) in the canonical form, i.e., \( \text{CL}(G; \Delta) := \{ \delta \in \Delta \mid G^\delta = CF_\Delta(G) \} \).

For the purpose of recursion, it is usually more convenient to compute the entire coset of canonical labelings rather than just a canonical form.

**Lemma 20 ([5]).** The following three conditions characterize canonical labelings:

1. (CL1) \( \text{CL}(G; \Delta) \subseteq \Delta \),
2. (CL2) \( \text{CL}(G; \Delta) = \delta \text{CL}(G^\delta; \Delta) \) for all \( \delta \in \Delta \), and
3. (CL3) \( \text{CL}(G; \Delta) = \text{Aut}_\Delta(G) \pi \) for some (and thus for all) \( \pi \in \text{CL}(G; \Delta) \).

These conditions imply that the assignment of \( G^\pi \) for an arbitrary \( \pi \in \text{CL}(G; \Delta) \) is independent of the choice of \( \pi \) and that \( G^\pi \) is a canonical form of \( G \) with respect to \( \Delta \), justifying the name. Hence, in the following, we shall focus on computing cosets \( \text{CL}(G; \Delta) \) satisfying the conditions of the lemma. Note that the conditions of the lemma are independent of the graph class \( \mathcal{X} \).

Above, we made the assumption that the vertex set of the graph is a fixed linearly ordered set. We can drop this assumption and can define canonical labelings for arbitrary vertex sets by considering...
labeling cosets instead of groups. For a labeling coset $\tau \Delta \leq \text{Label}(V(G))$, we define $\text{CL}(G; \tau \Delta)$ to be the set $\tau \text{CL}(G^\tau; \Delta)$, which is well defined due to Property (CL2). In general, a canonical form $\text{CF}$ for an arbitrary graph class $\mathcal{X}$ (which is closed under isomorphisms) is a function such that $\text{CF}(G) \equiv G$ for all $G \in \mathcal{X}$, and $G \cong H$ implies $\text{CF}(G) = \text{CF}(H)$ for all $G, H \in \mathcal{X}$. Notice that $G^\tau$ defines such a canonical form of a graph $G$ for all $\pi \in \text{CL}(G; \text{Label}(V(G)))$. Thus, given a graph $G \in \mathcal{X}$, it suffices to compute $\text{CL}(G; \text{Label}(V(G)))$ for a function $\text{CL}$ satisfying the conditions provided by Lemma 20.

To compute a canonical labeling coset, we require several algorithmic tools related to group theory. The composition-width of a coset $\tau \Delta$, denoted $\text{cw}(\tau \Delta)$, is the smallest integer $k \in \mathbb{N}$ such that $\tau \Delta \in \Gamma_k$; that is, every composition factor of $\Delta$ is isomorphic to a subgroup of $\text{Sym}(k)$. The composition-width of a collection of cosets $\text{cw}(\mathcal{S})$ is the maximum composition-width among all elements of $\mathcal{S}$. Let $\omega(n)$ be the smallest function such that, if $\Delta \leq \text{Sym}(X)$ is primitive, then $|\Delta| \leq |X|^{\omega(\text{cw}(\Delta))}$. It is known that $\omega(n) \in O(n)$ as stated in [4] (see also [17]).

**Theorem 21 ([5]).** Given a graph $G$ and a labeling coset $\tau \Delta \in \text{Label}(V)$, a canonical labeling coset $\text{CL}(G; \tau \Delta)$ can be computed in time $|V(G)|^{O(\omega(\text{cw}(\Delta)))}$.

Analogously, for a hypergraph $X$, we first define the canonical labelings with respect to a group and an initial segment of $\mathbb{N}$ as the vertex set. As before, canonical labelings of hypergraphs can be characterized by the conditions (CL1)–(CL3) with $X$ in place of $G$. Then, for hypergraphs on arbitrary vertex sets and a given labeling coset, we define $\text{CL}(X; \rho \Delta)$ to be the set $\rho \text{CL}(X^\rho; \Delta)$.

**Theorem 22 ([21], see [24]).** Given a hypergraph $X = (V, E)$ and a labeling coset $\tau \Delta \in \text{Label}(V)$, a canonical labeling coset $\text{CL}(X; \tau \Delta)$ can be computed in time $(|V| + |E|)^{O(\omega(\text{cw}(\Delta)))}$.

We describe several polynomial time operations that can be used in general when dealing with labeling cosets of a set of natural numbers.

We first define an ordering on $\text{Label}([1, \ldots, n]) = \text{Sym}([1, \ldots, n])$. For two such permutations $\pi_1, \pi_2$ define $\pi_1 < \pi_2$ if there is an $i \in [1, \ldots, n]$ such that $\pi_1(i) < \pi_2(i)$ and $\pi_1(j) = \pi_2(j)$ for all $1 \leq j < i$. We extend the definition to labeling cosets of $[1, \ldots, n]$ as follows: Suppose $\tau_1 \Theta_1, \tau_2 \Theta_2 \leq \text{Label}([1, \ldots, n])$ are labeling cosets. Then, we define $\tau_1 \Theta_1 < \tau_2 \Theta_2$ if $|\tau_1 \Theta_1| < |\tau_2 \Theta_2|$ or if $|\tau_1 \Theta_1| = |\tau_2 \Theta_2|$ and the smallest element of $\tau_1 \Theta_1 \setminus \tau_2 \Theta_2$ is smaller (w.r.t. to $<$ on permutations) than the smallest element of $\tau_2 \Theta_2 \setminus \tau_1 \Theta_1$.

Let $\tau_1, \tau_2, \ldots, \tau_t$ be a set of bijections with the same domain and range. We denote by

$$\tilde{\langle} \tau_1, \tau_2, \ldots, \tau_t \tilde{\rangle}$$

the smallest coset containing all $\tau_i$. This coset can be computed in polynomial time (see for example [15, Lemma 9.1]). Indeed, adding the elements one-by-one and keeping the representation of all cosets small, a representation for $\tilde{\langle} \tau_1, \tau_2, \ldots, \tau_t \tilde{\rangle}$ can be computed in time polynomial in the size of the domain and linear in $t$.

**Lemma 23 (see [1, Lemma 6.2]).** There is a polynomial time algorithm that, given a labeling coset $\tau \Theta$ of the set $[1, \ldots, n]$, as $\tau$ and a generating set $S$ of $\Theta$, computes a sequence of at most $n \log n$ elements $(\tau_1, \tau_2, \ldots, \tau_t)$ of $\tau \Theta$ such that

1. $\tau \Theta = \tilde{\langle} \tau_1, \ldots, \tau_t \tilde{\rangle}$,
2. $\tau_i$ is the smallest element of $\tau \Theta$ not contained in $\tilde{\langle} \tau_1, \ldots, \tau_{i-1} \tilde{\rangle}$, and
3. the output of the algorithm only depends on $\tau \Theta$ (and not on $\tau$ or $S$).

**Proof.** It suffices to find a solution for the following task: Given $\tau_1, \tau_2, \ldots, \tau_{t'}$ find the smallest element in $\tau \Theta$ not contained $\tilde{\langle} \tau_1, \tau_2, \ldots, \tau_{t'} \tilde{\rangle}$.

This task can be solved in polynomial time as follows: We find the smallest $j$ such that not all elements of $\tau \Theta$ have the same image of $j$. Otherwise, if each element of $\tau \Theta$ has the same

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
image, then \( \tau \Theta \) consists of only one element and we are done. With standard group theoretic techniques, including the Schreier-Sims algorithm, we can compute the possible images of \( j \) under \( \tau \Theta \), say \( i_1, i_2, \ldots, i_k \). For each \( m \in \{1, \ldots, k\} \), we compute \( (\tau \Theta)_{j \rightarrow i_m} \), the set of those elements in \( \tau \Theta \) that map \( j \) to \( i_m \). Notice that if \( \tau \Theta \) contains an element not in \( \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle \), then there is also a \( (\tau \Theta)_{j \rightarrow i_m} \) that contains an element not in \( \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle \). We find the smallest \( i_m \) such that \( (\tau \Theta)_{j \rightarrow i_m} \) is not contained in \( \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle \). Now, we compute \( \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle j \rightarrow i_m \) and recurse with \( (\tau \Theta)_{j \rightarrow i_m} \).

Each of the operations can be performed in polynomial time with standard techniques (see [15, Section 9]).

Note that the lemma can be used to compute canonical generating sets of a permutation group defined on a linearly ordered set.

**Corollary 24.**

1. Given a labeling coset \( \tau \Theta \) of \( [1, \ldots, n] \), we can compute the smallest element (w.r.t. “\( \prec \)”) in time polynomial in \( n \).
2. Given a collection of labeling cosets \( \{\tau_1 \Theta_1, \ldots, \tau_t \Theta_t\} \) of \( [1, \ldots, n] \) via generating sets, we can compute their order (w.r.t. “\( \prec \)”) in time quadratic in \( t \) and polynomial in \( n \).

### 7.2 Graphs with Coset-labeled Hyperedges

With the results established in the previous subsection, we can now give a variant of Lemma 11 suitable for canonization. For this purpose, we adapt the problem of coset-hypergraph isomorphism to the context of computing canonical labelings. More precisely, we define a new combinatorial object, namely hypergraphs with coset-labeled hyperedges, and canonical labelings for those objects.

A hypergraph with coset-labeled hyperedges is a 4-tuple \( (V, \mathcal{S}, \chi, \mathcal{F}) \) consisting of

1. a vertex set \( V \),
2. the hyperedges \( \mathcal{S} \subseteq \text{Pow}(V) \) that form a collection of subsets of \( V \),
3. a coloring \( \chi: \mathcal{S} \to \mathbb{N} \) of the hyperedges, and
4. a collection \( \mathcal{F} = \{\tau_\Sigma \Theta_\Sigma \mid \Sigma \in \mathcal{S}\} \) containing a labeling coset \( \tau_\Sigma \Theta_\Sigma \leq \text{Label}(\Sigma) \) for every \( \Sigma \in \mathcal{S} \) such that \( \Theta_\Sigma = \Theta_{\Sigma'} \) for all \( \Sigma, \Sigma' \in \mathcal{S} \) with \( \chi(\Sigma) = \chi(\Sigma') \).

For a map \( \varphi: V \to V' \), we define \( (V, \mathcal{S}, \chi, \mathcal{F})^\varphi \) to be the tuple \( (V'^\varphi, S'^\varphi, \chi'^\varphi, \mathcal{F}'^\varphi) \), where \( S'^\varphi = \{S'^\varphi \mid S \in \mathcal{S}\} \), \( \chi'^\varphi = \varphi^{-1} \chi \) and \( \mathcal{F}'^\varphi \) is the collection \( \{\varphi^{-1} \tau_\Sigma \Theta_\Sigma \} \). The automorphism group of a hypergraph with coset-labeled hyperedges \( X \) with respect to a group \( \Delta \leq \text{Sym}(V(X)) \), written \( \text{Aut}_\Delta(X) \), are all permutations \( \delta \in \Delta \) such that \( X^\delta = X \).

Now, we adapt the canonization to our combinatorial object and define canonical labelings for hypergraphs with coset-labeled hyperedges. As before, we assume that \( V(X) = \{|1, \ldots, |V(X)|\} \). A canonical labeling is a map that assigns every pair consisting of a hypergraph with coset-labeled hyperedges \( X \) and a group \( \Delta \leq \text{Sym}(|V|) \) a labeling coset \( \text{CL}(X; \Delta) \) such that

- (CL1) \( \text{CL}(X; \Delta) \subseteq \Delta \),
- (CL2) \( \text{CL}(X; \Delta) = \delta \text{CL}(X^\delta; \Delta) \) for all \( \delta \in \Delta \), and
- (CL3) \( \text{CL}(X; \Delta) = \text{Aut}_\Delta(X) \pi \) for some (and thus for all) \( \pi \in \text{CL}(X; \Delta) \).

As before, we extend the canonical labelings to arbitrary (not necessarily ordered) vertex sets. For a labeling coset \( \tau \Delta \leq \text{Label}(V) \), we write \( \text{CL}(X; \tau \Delta) \) for \( \tau \text{CL}(X^\tau; \Delta) \). This is well defined due to Property (CL2).

We are now ready to formulate the analogue of Lemma 11 for the canonization algorithm.
Lemma 25. There is an algorithm that computes a canonical labeling of a pair \((X; \tau \Delta)\) consisting of a hypergraph with costet-labeled hyperedges \(X = (V, S, \chi, F)\) and a labeling cost \(\tau \Delta \leq \text{Label}(V)\), which has a running time of \(\left| V \right| + \left| S \right| \in \Theta(\min\{\text{cw}(\Delta), \text{cw}(F)\})\).

Proof. Since \(\text{CL}(X; \tau \Delta) = \text{CL}(X^\tau; \Delta)\) by definition, we need to give an algorithm that, given a pair \((X; \Delta)\) with \(X = (V, S, \chi, F)\), computes \(\text{CL}(X; \Delta)\). If \(\Delta\) does not preserve \(S\) and \(\chi\), we proceed as follows: Using Theorem 22, we compute a canonical labeling cost \(\tau \Delta' = \text{CL}((V, S, \chi'; F)\) of the hypergraph so that \(\Delta'\) preserves \(S^\tau\) and \(F^\tau\). We compute and return \(\text{CL}(X; \tau \Delta')\) recursively (notice that \(\text{CL}(X; \tau \Delta')\) is defined as \(\text{CL}(X^\tau; \Delta')\)).

So, we can assume in the following that \(\Delta\) preserves \(S\) and \(\chi\), i.e., \(S^\delta = S\) and \(\chi^\delta = \chi\) for all \(\delta \in \Delta\). Let \(U\) be the set of pairs \(\{(S, s) \mid S \in \mathcal{S}, s \in S\}\). We can think of \(U\) as the disjoint union of all sets \(S \in \mathcal{S}\). For notational convenience, we define \(U_S := \{S\} \times \mathcal{S}\).

Next, we define a labeling \(\hat{\Delta}\) on the disjoint union \(V \cup U\). Roughly speaking, we define a labeling on \(V \cup U\) that orders \(V\) according to a \(\delta \in \Delta\) and orders the sets \(U_S\) according to the ordering of the sets \(S^\delta\). Inside the set \(U_S\) the vertices are ordered according to some \(\rho_S \in \tau_S \Theta_S\). More formally, we define \(\hat{\Delta}\) to be the labeling of \(V \cup U\) consisting of all labelings \(\hat{\delta}\) for which there are \(\rho_S \in \tau_S \Theta_S\) such that

\[
\begin{align*}
(1) \quad & \hat{\delta}|_V = \delta \text{ for some element of } \delta \in \Delta, \\
(2) \quad & (S, s)^{\hat{\delta}} < (S', s')^{\hat{\delta}} \text{ if } S^{\delta} < S'^{\delta} \text{ (the ordering } < \text{ of subsets of } \mathbb{N} \text{ as defined in the preliminaries)}, \text{ and} \\
(3) \quad & (S, s)^{\hat{\delta}} < (S', s')^{\hat{\delta}} \text{ if } \rho_S < \rho_{S'}
\end{align*}
\]

for all \((S, s), (S', s') \in U\).

The cost \(\hat{\Delta}\) is a labeling coset of the set \(V \cup U \) and also, it has composition-width at most \(\max\{\text{cw}(\Delta), \text{cw}(F)\}\).

Consider the bipartite graph \(Y\) defined on \(V \cup U\) by connecting \(v\) and \((S, \nu)\) for each \(S \in \mathcal{S}\) for which \(v \in S\). We return the canonical labeling coset of the graph \(Y\) restricted to \(V\), i.e., \(\text{CL}(Y; \hat{\Delta})|_V\).

(Canonical labeling.) We argue that \(\text{CL}(Y; \hat{\Delta})\) restricted to \(V\) is a canonical labeling coset of \((X; \Delta)\).

(CL1) By definition, all elements of \(\hat{\Delta}\) restricted to \(V\) are in \(\Delta\).

(CL2) We consider the result of applying the algorithm to \((X^{\hat{\delta}}; \Delta)\) for any \(\delta \in \Delta\). In the case that \(\Delta\) does not preserve \(S\) and \(\chi\), the algorithm returns \(\text{CL}(X^{\hat{\delta}}; \tau \Delta') = \delta^{-1} \text{CL}(X; \tau \Delta')\). So, assume that \(\Delta\) already preserves \(S\) and \(\chi\). In this case the definition of \(V \cup U\) remains unchanged, since \(\Delta\) preserves \(S\). In place of \(\hat{\Delta}\), the algorithm computes \(\hat{\Delta}\). In place of \(\hat{\Delta}\) the algorithm computes \(\hat{\chi}^\delta\). Then the result of the computation is \(\text{CL}(Y; \hat{\Delta})|_V = \delta^{-1} \text{CL}(Y; \hat{\Delta})|_V\).

(CL3) Suppose \(\hat{\delta}, \hat{\delta}' \in \text{CL}(Y; \hat{\Delta})\) and \(\hat{\delta}|_V = \hat{\delta}'|_V =: \delta). We need to show that \(\delta \delta'^{-1}\) is an automorphism of \(X\). Since \(\hat{\delta}, \hat{\delta}' \in \Delta\), it is clear that \(\delta \delta'^{-1} \) preserves \(S\) and \(\chi\). It remains to show that \(\delta \delta'^{-1}\) is compatible with \(F\). In other words, we need to show that \(F^\delta = \delta^{-1} \tau_S \Theta_S \) and \(F^\delta = \delta^{-1} \tau_{S'} \Theta_{S'}\) are actually identical.

Choose \(S \in \mathcal{S}\) and let \(S' := S^{\delta'^{-1}} \in \mathcal{S}\). Since \(\chi\) is preserved, we have \(\Theta_S = \Theta_{S'}\) and therefore it suffices to show that \(\delta^{-1} \tau_S \Theta_S = \delta'^{-1} \tau_{S'} \Theta_{S'}\).

Consider the graph \(Y^{\hat{\delta}} = Y^{\hat{\delta'}}\). Let \(\gamma_S\) be the map from \(S^{\hat{\delta}}\) to \((U_S)^{\hat{\delta}}\) that maps each \(v^{\hat{\delta}} \in S^{\hat{\delta}}\) to \((S, v)^{\hat{\delta}}\). Similarly, let \(\gamma_{S'}\) be the map from \(S'\) to \((U_{S'})^{\hat{\delta}}\) that maps each \(v^{\hat{\delta}} \in S^{\hat{\delta}}\) to \((S', v)^{\hat{\delta}}\). In the graph \(Y^{\hat{\delta}}\), for \(s \in S^{\hat{\delta}}\), we have that \(s \tau S\) is the only vertex in \((U_S)^{\hat{\delta}}\) that is a neighbor of \(s\). Similarly, in \(Y^{\hat{\delta'}} = Y^{\hat{\delta}}\), we have that \(s \tau S'\) is the only vertex in \((U_{S'})^{\hat{\delta}}\) that is a neighbor of \(s\). It follows that \(\gamma_S = \gamma_{S'}\).
By construction of $\hat{\Delta}$, we know that there is an integer $m$ and $\rho_S \in \tau_S \Theta_S$ such that for all $v \in V$ it holds $|S_\rho(v) \cdot \rho_S(v) + m|$. Similarly, there are $m', \rho'_S \in \tau'_S \Theta'_S$ with $|S_\rho'(v) \cdot \rho'_S(v) + m'|$ for all $v \in V$. In fact, it must be the case that $m = m'$, since $(U_{S_\rho'})^\delta = (U_S)^\delta$. We conclude that $\delta \delta'^{-1} = \delta Y_{S_\rho'} \delta'^{-1} = \rho_S \rho'^{-1} \rho_S$. Therefore, $\delta \delta'^{-1} \tau'_S \Theta_S = \tau S \Theta_S$ as desired.

(Running time.) We use two subroutines throughout the algorithm. The first one is Miller’s canonization algorithm for hypergraphs from Theorem 22, which runs in time $(|V| + |S|)^{O(\text{cw}(\Delta))}$. The second one is a canonization algorithm for the graph $Y$, which runs in time $(|V| + |S|)^{O(\text{cw}(\Delta) \cdot \text{cw}(Y))}$.

We also need a second combinatorial subproblem that handles the case when $V$ is small but the family $\mathcal{F}$ is possibly large (this is analogous to multiple-colored-coset-isomorphism and Lemma 14).

A set with multiple colored labeling cosets is a tuple $(V, \mathcal{F}, \chi)$ consisting of

1. a vertex set $V$,
2. a collection $\mathcal{F} = \{\tau_1 \Theta_1, \tau_2 \Theta_2, \ldots, \tau_t \Theta_t\}$ of labeling cosets of $V$, and
3. a map $\tau : \{1, \ldots, t\} \to \mathbb{N}$ assigning to every labeling coset in $\mathcal{F}$ a positive integer.

For a map $\varphi : V \to V'$, we define $(V, \mathcal{F}, \chi)^\varphi$ analogously to before. Similarly, the automorphism group and canonical labelings of $(V, \mathcal{F}, \chi)$ are defined analogously to above.

We are now ready to formulate a lemma that is the analogue of Lemma 14.

**Lemma 26.** There is an algorithm that computes a canonical labeling of a pair $(X; \tau \Delta)$ consisting of a set with multiple colored labeling cosets $X = (V, \mathcal{F}, \chi)$ and a labeling coset $\tau \Delta \leq \text{Label}(V)$, which has a running time of $2^{O(|V| \log |V|)} |\mathcal{F}|^2$.

**Proof.** Since $\text{CL}(X; \tau \Delta)$ is $\tau \text{ CL}(X^\tau; \Delta)$ by definition, we need to give an algorithm that computes a canonical labeling for a given group $\Delta \leq \text{Sym}([1, \ldots, |V|])$ and a given set with multiple colored labeling cosets on a ground set satisfying $V = [1, \ldots, |V|]$. Let $\mathcal{F} = \{\tau_1 \Theta_1, \tau_2 \Theta_2, \ldots, \tau_t \Theta_t\}$ be the family of labeling cosets.

For each labeling $\delta \in \Delta$, we do the following: Consider the sets of the form $\delta^{-1} \tau_\delta \Theta_\delta$. We order these sets so that $\delta^{-1} \tau_{\delta_i} \Theta_{\delta_i} \prec \delta^{-1} \tau_{\delta_j} \Theta_{\delta_j}$ if $\chi(\tau_{\delta_i} \Theta_{\delta_i}) < \chi(\tau_{\delta_j} \Theta_{\delta_j})$ or if $\chi(\tau_{\delta_i} \Theta_{\delta_i}) = \chi(\tau_{\delta_j} \Theta_{\delta_j})$ and $\delta^{-1} \tau_{\delta_i} \Theta_{\delta_i} < \delta^{-1} \tau_{\delta_j} \Theta_{\delta_j}$, where $\prec$ is the ordering of labeling cosets of $[1, \ldots, |V|]$ defined in Section 7.1. By Corollary 24 this ordering can be computed in time quadratic in $|\mathcal{F}|$ and polynomial in $|V|$. For every $\delta$, we get an ordered sequence $\delta^{-1} \tau_{\delta_1} \Theta_{\delta_1}, \ldots, \delta^{-1} \tau_{\delta_t} \Theta_{\delta_t}$.

Let $(\delta_1, \ldots, \delta_t)$ be the collection of those $\delta$ for which this ordered sequence is lexicographically minimal. Then, we let $(\delta_1, \ldots, \delta_t)$ be the output of the algorithm.

It follows from the construction that $(\delta_1, \ldots, \delta_t)$ satisfies the properties of a canonical labeling coset.

By Corollary 24, for each $\delta$ the time requirement of the algorithm is quadratic in $|\mathcal{F}|$ and polynomial in $|V|$. There are at most $|V|! \in 2^{O(|V| \log |V|)}$ many elements in $\Delta$, and thus at most equally many choices for $\delta$, giving us the desired time bound.

### 7.3 The Canonization Algorithm

Finally, we have assembled all the tools to describe an algorithm that canonizes graphs of bounded tree-width.

**Theorem 27.** There is an algorithm that, given $k \in \mathbb{N}$ and a connected graph $G$, either correctly concludes that $\text{tw}(G) > k$ or computes a canonical form $\text{CF}(G)$ in time $2^{O(k^2 \log k)} |V(G)|^3$. 

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
We remark that the algorithm is closely related to the one described in Theorem 16. In the isomorphism algorithm the graph $\eta(t)$ serves as a tool to exploit the structure of each bag $t$. As the isomorphism techniques depend on the isomorphism test for graphs of bounded degree given in [13], which does not extend to canonization yet, we will now use the fact that additionally each bag can be guarded with a labeling coset $\alpha(t)$ of bounded composition-width. The labeling coset $\alpha(t)$ serves as a tool for efficient canonization. The main change of the algorithm therefore arises in (L2).

**Proof.** We describe a dynamic programming algorithm $\mathcal{A}$ with input $I = (G, S, T, \beta, \alpha)$ where $(T, \beta)$ is a tree decomposition of a graph $G$ and $S \subseteq V(G)$ is a subset of the root bag, and $\alpha$ is a (partial) function that maps nodes $t \in V(T)$ to labeling cosets $\alpha(t) \subseteq \text{Label}(\beta(t))$. The algorithm computes a non-empty set of labelings $\mathcal{A}(I) \subseteq \Lambda(I)$ such that $\mathcal{A}(I)|_S = \Lambda(I)|_S$ and

- (CL1) $\Lambda(I) \leq \text{Label}(V(G))$ and $\Lambda(I)|_S \leq \text{Label}(S)$,
- (CL2) $\Lambda(I) = \tau\Lambda(I^t)$ for all $\tau \in \text{Label}(V(G))$, and
- (CL3) $\text{Aut}(G, S, T, \beta, \alpha)\pi \subseteq \Lambda(I) \subseteq \text{Aut}(G)\pi$ for some (and thus for all) $\pi \in \Lambda(I)$.

In our recursive procedure, we maintain the following properties of the tree decomposition for each unlabeled bag $\beta(t)$ ($\alpha(t) = \bot$):

- (U1) The graph $G^k[\beta(t)]$ is clique-separator free,
- (U2) Each adhesion set of $t$ and also $S$ are cliques in $G^k$, and
- (U3) The adhesion sets to the children of $t$ are all equal to $\beta(t)$ or the adhesion sets to the children are all distinct.

For each labeled bag $\beta(t)$, we require the following:

- (L1) The cardinality $|\beta(t)|$ is bounded by $c_L$,
- (L2) The number of children of $t$ is bounded by $kc_L^2 + 2^k c_L$ and $\alpha(t) \in \tilde{I}_{k+1}$,
- (L3) The adhesion sets to the children of $t$ are all equal to $\beta(t)$ or the adhesion sets to the children are all distinct, and
- (L4) If the adhesion sets are all equal, then the cardinality $|\beta(t)|$ is bounded by $c_M$.

The initial input of $\mathcal{A}$ is the canonical clique separator decomposition $(T, \beta)$ of the $k$-improved graph $G^k$ from Theorem 6. For $S$, we choose the empty set. Since the clique separator decomposition is canonical, we have $\text{Aut}(G, S, T, \beta, \bot) = \text{Aut}(G)$ and therefore the algorithm computes a canonical labeling $\tau$ of $G$. Thus, $G^\tau$ is a canonical form of $G$.

**Description of $\mathcal{A}$.**

Let $t_1, \ldots, t_t$ be the children of $t$ and let $(T_i, \beta_i, \alpha_i)$ be the decomposition of the subtree rooted at $t_i$. Let $G_i$ be the graph corresponding to $(T_i, \beta_i, \alpha_i)$. Let $V_i := V(G_i)$, let $S_i := \beta(t) \cap V_i$ be the adhesion sets to the children, and let $Z_i$ be $V_i \setminus S_i$. We assume that the canonical labelings $\mathcal{A}(G_i, S_i, T_i, \beta_i, \alpha_i)$ have already been computed via dynamic programming. Also, we pick $\tau_i^* \in \mathcal{A}(G_i, S_i, T_i, \beta_i, \alpha_i)$ and set

$$\tau_i \Theta_i := \mathcal{A}(G_i, S_i, T_i, \beta_i, \alpha_i)|_S = \Lambda(G_i, S_i, T_i, \beta_i, \alpha_i)|_S \leq \text{Label}(S)$$

where $\tau_i = \tau_i^*|_S$. Finally, let $\tau_i^* \Theta_i^* := \Lambda(G_i, S_i, T_i, \beta_i, \alpha_i)$ (recall that the algorithm only computes a subset of $\tau_i^* \Theta_i$).

We have to consider two cases depending on the root $t \in V(T)$.

**Case $\beta(t)$ is unlabeled.** We distinguish two further subcases. First suppose the adhesion sets to the children of $t_i$ are all equal to $\beta(t_i)$. By (U1), $\beta(t_i)$ is a clique and thus, $|\beta(t_i)| \leq k + 1$. Compute a partial linear order on the graphs $G_i$ by comparing the resulting adjacency...
matrices of $G^i_{\tau}$ lexicographically (notice that the order does not depend on the representative $\tau^i_{\tau}$). Define $\chi : \{S_1, \ldots, S_\ell\} \rightarrow \{1, \ldots, h\}$ according to this order where $h$ is the number of equivalence classes. We refine $\chi$ such that $\chi(S_i) = \chi(S_j)$ implies $\Theta_i = \Theta_j$ (according to the linear order defined on labeling cosets on an initial segment of the natural numbers). We compute $\Lambda$ by applying the algorithm of Lemma 26 to the input $X = ((\beta(t), \{\tau_i\Theta_i\}), \chi); \Lambda(\beta(t)))$. Then, in a second step, we compute $\Lambda'$ by applying Theorem 22 to $((\beta(t), \{S\}), \Lambda)$. For each $\lambda \in \Lambda'|_S$, we need to compute an extension to the entire vertex set. This is done analogously to the labeled case described below.

In the other case, all adhesion sets are distinct. For each vertex $v \in \beta(t)$ of degree at most $k$ in $G^k[\beta(t)]$, we compute the canonical coset-labeled tree decomposition $(T', \beta', \alpha')$ of $(G^k[\beta(t)], v)$ from Theorem 9. Notice that the bags of $(T', \beta', \alpha')$ satisfy (L1)–(L4), because in the case $tw(G) \leq k$, the $k$-improvement does not increase the tree-width as seen in Lemma 5(2). Notice that the adhesion sets $S_i$ are cliques in $G^k$ and these cliques must be completely contained in one bag. We attach the children $(T_i, \beta_i, \alpha_i)$ to $(T', \beta', \alpha')$, adding them to the highest possible bag, and we choose a new root as the highest possible node $r \in V(T')$ such that $S \subseteq \beta'(r)$. By doing this, we obtain $(T'', \beta'', \alpha'')$. We ensure Property (L2), because the number of children attached to one bag can be bounded by the number of cliques in $G^k$ that contains the bag. The number of cliques again can be bounded by $2^k c_L$, which is a consequence of Lemma 15. By possibly introducing new bags, we can preserve Property (L3): The adhesion sets are either pairwise-different or all equal. Since the adhesion sets $S_i$ are cliques in $G^k$ and therefore their size is bounded by $k+1$, we also preserve Property (L4).

For each $v$, we compute $\Theta_v := \mathcal{A}(G, S, T'', \beta'', \alpha'')$ recursively. Moreover, we define $\Lambda_v := \Lambda(G, S, T'', \beta'', \alpha'')$ for each $v$, we compute the canonical form obtained by applying an arbitrary element of $\Theta_v$ to the graph. Let $M$ be those $v$ for which the obtained canonsized graph is minimal with respect to lexicographic comparison of the adjacency matrix. Then, we define $\Lambda(I)$ to be the smallest coset containing $\bigcup_{v \in M} \Lambda_v$, and for each $\lambda' \in \Lambda(I)|_S$, we return a single element $\lambda \in \Lambda(I)$ such that $\lambda|_S = \lambda'$. These elements can be computed from the sets $\Theta_v$ (recall that $\Theta_v|_S = \Lambda_v|_S$).

Case $\beta(t)$ is labeled: We apply Lukš’s algorithm from Theorem 21 to get a subcoset that respects the edge relation and get $\psi'\Lambda' := \text{CL}(G[\beta(t)]; \alpha(t))$. We apply Miller’s algorithm from Theorem 22 to get a subcoset that respects $S$ and compute $\psi \Lambda := \text{CL}((\beta(t), \{S\}); \psi'\Lambda')$. Let $\mu \in \psi \Lambda$. We define a map $\kappa : S^\mu \rightarrow \{1, \ldots, |S|\}$ such that $\kappa^k < u^\mu$, if and only if $\kappa^\mu < u^\mu$. This is an isomorphism-invariant renaming of the vertices such that $S^{\mu \kappa} = \{1, \ldots, |S|\}$. Define $\psi \Lambda := \psi \Lambda \kappa$.

Compute a partial linear order on the graphs $G_i$ by comparing the resulting adjacency matrices of $G^i_{\tau}$ lexicographically (notice that the order does not depend on the representative $\tau^i_{\tau}$). Define $\chi : \{S_1, \ldots, S_\ell\} \rightarrow \{1, \ldots, h\}$ according to this order where $h$ is the number of equivalence classes. We refine $\chi$ such that $\chi(S_i) = \chi(S_j)$ implies $\Theta_i = \Theta_j$.

If all adhesion sets are different, we compute a canonical labeling coset $\Lambda$ of the set $\beta(t)$ by applying Lemma 25 to the input $X = ((\beta(t), \{\tau_i\Theta_i\}), \chi); \Lambda(\beta(t)))$. Otherwise, if all adhesion sets are equal, we compute $\Lambda$ by applying the algorithm of Lemma 26 to the input $X = ((\beta(t), \{\tau_i\Theta_i\}), \chi); \Lambda(\beta(t)))$. (Notice that Property (CL1) allows us to restrict ourselves to $S_i$.)

We describe a canonical labeling coset of $G$. For each $\lambda \in \Lambda$, we define extensions to $V(G)$ as follows: For each $i \in [\ell]$, we define $\tau^i_{\Theta^i_{\lambda}}$ to be the subcoset of those maps $\rho_i \in \tau^i_{\Theta^i_{\lambda}}$ for which $\lambda^{-1}\rho_i$ is minimal. (In fact, we have to restrict these maps to fit together

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
\((\lambda^{-1}|_{S_{\lambda}})(\rho|_{S_{\lambda}}))\). In the case that all adhesions are different, we define \(\tilde{\lambda}\) to be the set of labelings \(\tilde{\lambda}\) of \(V(G)\) for which there are \(\rho_1 \in \tau_1'\Theta_1', \ldots, \rho_\ell \in \tau_\ell'\Theta_\ell'\) such that

1. \(\tilde{\lambda}|_{\beta(i)} = \lambda_i\),
2. \(\nu^\lambda_i < \nu^\lambda_j\), if \(v_i \in Z_i, v_j \in Z_j\) and \(S^\lambda_i < S^\lambda_j\) (the ordering \(<\) of subsets of \(\mathbb{N}\) as defined in the preliminaries), and
3. \(\nu^\lambda_i < u^\lambda\), if \(v, u \in Z_i\) and \(v^\rho_i < u^\rho_i\).

In the case that all adhesion sets are equal, we replace (2) by the following two conditions: \(\nu^\lambda_i < \nu^\lambda_j\), if \(v_i \in Z_i, v_j \in Z_j\) and \(\lambda^{-1}\tau_i\Theta_i < \lambda^{-1}\tau_j\Theta_j\). For all \(i \in [\ell]\) it holds \(Z^\lambda_i = Z^{\rho_i} + m\) for some \(\rho_i \in \tau'_i\Theta'_i\) and some integer \(m\).

We define \(\tilde{\Lambda}\) as the smallest coset containing \(\bigcup_{\lambda \in \tilde{\Lambda}} \lambda\) and set \(\Lambda(I) := \tilde{\Lambda}\).

For each \(\lambda' \in \Lambda(I)|_{S}\), we return a single element \(\lambda \in \Lambda(I)\) such that \(\lambda|_{S} = \lambda'\). These elements can be computed from the sets \(\Lambda\) and \(\mathcal{A}(G, S, T, \beta, \alpha)\) (recall that \(\mathcal{A}(G, S, T, \beta, \alpha)|_{S} = \tau'_i\Theta'_i|_{S}\)).

(Canonical labeling.) Property (CL1) is clearly satisfied. We claim that for a partially coset-labeled decomposition \((T, \beta, \alpha)\) the set \(\Lambda(I)\) satisfies (CL2) and (CL3). The correctness of the second part of the first case is easy to see. The returned set is non-empty, because every graph of tree-width at most \(k\) is \(k\)-degenerate and therefore each subgraph has a vertex of degree at most \(k\). The first part of the first case is analogous to the second case, which is considered below.

(CL2) We consider the result of applying the algorithm to \(G^v\) for any \(\tau \in \text{Label}(V(G))\). So, the initial input of the algorithm is \((T, \beta)^v\) instead of \((T, \beta)\). Let \(\tau|_v := \tau|_{V(G^v)}\). The children of the root of \(T^v\) have the decompositions \((T_i, \beta_i, \alpha_i)^{v_i}\) with the corresponding graphs \(G^v_i\) instead. By induction, we assume that (CL2) holds for subgraphs of \(G\). Property (CL2) implies that even for an arbitrary bijection \(\tau|_v\) (whose image is not necessarily an initial segment of \(\mathbb{N}\)) it holds that \((\tau'_i\Theta'_i)^{v_i} = \Lambda(G^v_i, S^v_i, (T_i, \beta_i, \alpha_i)^{v_i})\). Let \(\tau|_\emptyset := \tau|_{\beta(1)}\). We get \(\alpha(t)^{v_{\emptyset}}\) instead of \(\alpha(t)\) and get \((\psi\lambda)^{v_{\emptyset}}\) instead of \(\psi\lambda\). We explain the case when all adhesion sets are different. In this case, we get \(\tau|_{\emptyset}^v\Lambda\) instead of \(\Lambda\). Since \(\lambda^{-1}\rho_i = (\tau|_{v_{\emptyset}}\lambda)^{-1}\tau|_{\emptyset}^{-1}\rho_i\), we chose \(\tau|_{\emptyset}^{-1}\rho_i\) as a minimal element instead of \(\rho_i\) and this leads to \(\tau|_{\emptyset}^{-1}\tau'_i\Theta'_i\) instead of \(\tau'_i\Theta'_i\). Because of (CL2) of the canonical labelings of a hypergraph with coset-labeled hyperedges, we get \(\tau|_{\emptyset}^{-1}\Lambda\) instead of \(\Lambda\). Since \(v^\lambda = v^\tau^{-1}v^\lambda\) for all \(v \in V(G)\), we get \(\tau^{-1}\Lambda|_{\emptyset}\) and finally \(\tau^{-1}\bigcup_{\lambda \in \tilde{\Lambda}} \lambda\). Observe that the smallest coset containing \(\tau^{-1}\bigcup_{\lambda \in \tilde{\Lambda}} \lambda\) is actually \(\tau^{-1}\tilde{\Lambda}\), which shows the correctness. The case in which all adhesion sets are distinct is similar.

(CL3) The first inclusion \(\text{Aut}(G, S, (T, \beta, \alpha))\pi \subseteq \Lambda(I)\) follows from (CL2). It remains to prove \(\Lambda(I) \subseteq \text{Aut}(G)\pi\). Assume we have \(\tilde{\lambda}, \tilde{\lambda}' \in \bigcup_{\lambda \in \Lambda} \tilde{\lambda}\) and \(\tilde{\lambda}|_{\beta(1)} = \lambda\) and \(\tilde{\lambda}'|_{\beta(1)} = \lambda'\). We have to show \(\tilde{\lambda}^{-1}\tilde{\lambda}'^{-1}\) is an automorphism of \(G\). Since both \(\lambda, \lambda' \in \Lambda \leq \text{CL}(G[\beta(t)]; \alpha(t))\), we conclude that \(\lambda^{-1}\lambda'\) preserves the edge relation \(E(\beta(t))\). With the same argument \(\lambda\lambda'^{-1}\) preserves \(S\). We explain the case when all adhesion sets are different. We have \(\tilde{I} = \tilde{I}'\) and therefore \(\lambda\lambda'^{-1}\) preserves \(\{S_i\}, \{\tau_i\Theta_i\}\) and \(\chi\). So, assume that \(\lambda\lambda'^{-1}\) maps \(S_i\) to \(S_j\). It remains to show that \(\tilde{\lambda}^{-1}\tilde{\lambda}'^{-1}\) is an isomorphism from \(G_i\) to \(G_j\). By construction, we have \(\tilde{\lambda}|_{Z_i} = \rho_i + m\) and \(\tilde{\lambda}'|_{Z_j} = \rho'_j + m'\) for \(\rho_i \in \tau'_i\Theta'_i, \rho_j \in \tau'_j\Theta'_j\). In fact, it must be the case that \(m = m'\), because the labelings \(\tilde{\lambda}, \tilde{\lambda}'\) order the sets \(Z_i, Z_j\) according to the set \(S^\lambda_i = S^{\rho_i}\) as required in (2). Since \(m = m'\), we conclude \(\tilde{\lambda}|_{Z_i}(\tilde{\lambda}'|_{Z_j})^{-1} = \rho_i|_{Z_i}(\rho'_j|_{Z_j})^{-1}\). Since \(\lambda\lambda'^{-1}\) preserves \(\{\tau_i\Theta_i\}\), we conclude \(\lambda^{-1}\tau_i\Theta_i = \lambda^{-1}\tau_j\Theta_j\) and we get \(\lambda^{-1}\rho_i|_{Z_i} = \lambda^{-1}\rho_j|_{Z_j}\) by the minimality of the elements in \(\tau'_i\Theta'\) and \(\tau'_j\Theta'\). This gives us...
\( \hat{\lambda}|_{S_i}(\hat{\lambda}'|_{S_j})^{-1} = \lambda|_{S_i}(\lambda'|_{S_j})^{-1} = \rho_i|_{S_i}(\rho'|_{S_j})^{-1}. \)

We already noticed that \( \hat{\lambda}|_{Z_i}(\hat{\lambda}'|_{Z_j})^{-1} = \rho_i|_{Z_i}(\rho'|_{Z_j})^{-1} \), which leads to \( \hat{\lambda}|_{V_i}(\hat{\lambda}'|_{V_j})^{-1} = \rho_i|_{V_i}(\rho'|_{V_j})^{-1}. \) Since \( \chi \) is preserved, we know that that \( G'_i \chi = G'_j \) and therefore \( G_i \hat{\lambda}^{-1} = G_j \), which means that \( \hat{\lambda}^{-1} \) is an isomorphism.

(Running time.) Let \((T, \beta)\) be the clique separator decomposition of the \( k \)-improved graph \( G^k \) from Theorem 6 computed in the initial step of the algorithm. Note that \((T, \beta)\) can be computed in time \( O(k^2 n^3 + k^3 n^3) \) by Lemma 5 and Theorem 6. Also let \( s := \sum_{t \in V(T)} |\beta(t)|. \) Similar to the analysis for Theorem 16, we analyze the running time of the algorithm in terms of \( s \). Let \( t_U(s) \) denote the running time of the algorithm assuming the root bag is unlabeled and let \( t_L(s) \) be the running time of the algorithm assuming the root bag is labeled. Also, let \( s_i, i \in [\ell], \) denote the size of the decomposition rooted at \( t_i \), i.e., \( s_i := \sum_{t \in V(T)} |\beta_i(t)|. \) Note that \( s = |\beta(t)| + \sum_{i \in \ell} s_i. \)

In both cases, one vital subroutine is the lexicographic comparison of adjacency matrices. We first describe a data structure that can be initialized in time \( O(n^2) \) and performs a single comparison of adjacency matrices in time \( O(|E(G)|) \subseteq O(kn) \). The data structure consists of an \( n \times n \) matrix with entries in \( \mathbb{N}^2 \) and a counter \( r \). Initially, \( r := 0 \) and all entries of the matrix are set to \((0, 0)\). For each comparison of two adjacency matrices, we first increment the counter \( r \). Whenever an entry of the matrix is accessed, the first component is updated to \( r \). This way, it can be recognized whether an entry of the matrix has already been accessed during the current computation. We iterate over all edges of the two input graphs and mark the corresponding positions in the matrix (using the second component). By iterating over all edges a second time, we can then find the minimal edge that appears in only one of the graphs.

In \( \beta(t) \) is unlabeled, we distinguish two cases. Both cases are similar to the isomorphism algorithm from Theorem 16. If all adhesion sets to the children of \( t \) are equal to \( \beta(t) \), then

\[
t_U(s) = \sum_{i=1}^{\ell} t_U(s_i) + O(kn\ell \log \ell) + k^{O(1)}\ell^2 + 2^{O(k \log k)}\ell^2 + 2^{O(k \log k)}n
\]

First, we recursively compute \( \mathcal{A}(G_i, S_i, T_i, \beta_i, \alpha_i) \) for all \( i \in [\ell] \). Then, the algorithm computes a partial linear order by comparing the adjacency matrices of the graphs \( G'_i \) lexicographically. This can be done in time \( O(kn \ell \log \ell) \). In the next step, the coloring \( \chi \) is computed in time \( k^{O(1)}\ell^2 \) using Corollary 24.

The application of Lemma 26 requires time \( 2^{O(k \log k)}\ell^2 \). Finally, the last term gives the time required to compute the output given the labeling coset \( \Lambda' \) (all computations are performed on the set \( S \) where the labelings defined on the entire vertex set \( V(G) \) are dragged along). Note that \( |\beta(t)| \leq k + 1 \) and therefore, \( |\text{Label}(S)| \leq (k + 1)! \).

In the other case,

\[
t_U(s) = \sum_{i=1}^{\ell} t_U(s_i) + |\beta(t)| \cdot \left( 2^{O(k^2 \log k)}|\beta(t)|^2 + 2^{O(k \log k)}\ell^2|\beta_2(t_2)| + t_L(s') \right) + 2^{O(k^2 \log k)}|\beta(t)|n.
\]

Here, \( s' \) denotes the maximum size of a tree decomposition of \((G^k[\hat{\beta}(t)], v)\), where \( v \in \beta(t) \) is a vertex of degree at most \( k \) in \( G^k[\hat{\beta}(t)] \), computed by Theorem 9. Note that \( s' = O(k \cdot |\beta(t)|) \).

The last term gives the time required to combine the sets \( \Theta_v \) into the output \( \mathcal{A}(l) \) as before, all computations are performed on the set \( S \) where the labelings defined on the entire vertex set \( V(G) \) are dragged along.)
It remains to analyze the recurrence obtained in the case where the root bag is labeled. The application of Theorem 21 and Theorem 22 requires time $2^{O(k^2 \log k)}$. The computation of the coloring $\chi$ requires time $O(kn \ell \log \ell) + k^{O(1)} \ell^2$ similar to the analysis of the unlabeled case.

Assume that all adhesion sets are different. By Property (L1), the bag size of $\beta(t)$ is bounded by $c_L$; and by Property (L2), the number of children is bounded by $k c_L^2 + 2^k c_L$. By Property (L2), we know that $cw(\alpha(t)) \leq k + 1$ and also $cw(\tau_i \Theta_i |_{S_i}) \leq k + 1$, since $\tau_i \Theta_i |_{S_i}$ is a subcoset of $\alpha(t)$. Therefore, the canonization algorithm for hypergraphs with coset-labeled hyperedges in Lemma 25 runs in time $(c_L k c_L^2 + 2^k c_L)^{O(max\{cw(\alpha(t)), cw(\alpha(t_i))\})} \subseteq 2^{O(k^2 \log k)}$.

Next, consider the second case in which all adhesion sets are equal. In this case, we have that the bag size of $\beta(t)$ is bounded by $c_M$ due to Property (L4) and that the number of children of $t$ is bounded by $k c_M^2 + 2^k c_M$ due to Property (L2). Therefore, the canonization algorithm for the multiple colored cosets in Lemma 26 runs in time $2^{O(c_M \log c_M)} (k c_M^2 + 2^k c_M)^2 \subseteq 2^{O(k^2 \log k)}$.

Finally, in both cases, the canonical labelings of $G$ returned by the algorithm can be computed in time $2^{O(k^2 \log k)} n$.

Hence,

$$t_L(s) = \sum_{i=1}^{\ell} t_L(s_i) + O(kn \ell \log \ell) + k^{O(1)} \ell^2 + 2^{O(k^2 \log k)} + 2^{O(k^2 \log k)} n$$

$$\subseteq \sum_{i=1}^{\ell} t_L(s_i) + 2^{O(k^2 \log k)} n$$

using the fact $\ell \in 2^{O(k^2 \log k)}$ by Property (L2).

The recurrence described above resolves to a running time $2^{O(k^2 \log k)} |V(G)|^3$ as desired. More precisely, $t_L(s) \in 2^{O(k^2 \log k)} n s$. Thus,

$$t_U(s) = \sum_{i=1}^{\ell} t_U(s_i) + 2^{O(k^2 \log k)} n^2 \cdot (|\beta(t)| + \ell).$$

So, overall, $t_U(s) \in 2^{O(k^2 \log k)} n^3$. \qed

8 CONCLUSION

We presented an algorithm solving the isomorphism problem for graphs of tree-width at most $k$ in time $2^{k \log(k)} n^3$ time. Also, we gave an algorithm for canonizing graphs of tree-width at most $k$ with a slightly worse running time $2^{O(k^2 \log k)} n^3$. Naturally, this raises the question whether canonization of graphs of bounded tree-width can be performed in the same time as testing isomorphism. With Babai’s recent quasipolynomial time canonization algorithm [3] it seems plausible that such an improvement for the canonization algorithm can be achieved. However, as a first step in this direction, one would probably require a canonization algorithm for graphs of bounded degree matching the running time of the isomorphism test from [13].

Another question is whether the parameter dependence can be further improved. Can isomorphism for graphs of tree-width at most $k$ be tested in time $2^{O(k)} n^{O(1)}$? In a very recent work [25], the last author of this article provides an algorithm canonizing graphs of tree-width $k$ in time $n^{O(1)}$. With this result, one could even ask for an isomorphism test running in time $2^{O(1)} n^{O(1)}$.

ACM Transactions on Algorithms, Vol. 16, No. 3, Article 34. Publication date: May 2020.
REFERENCES

[1] Eric Allender, Joshua A. Grochow, Dieter van Melkebeek, Cristopher Moore, and Andrew Morgan. 2018. Minimum circuit size, graph isomorphism, and related problems. SIAM J. Comput. 47, 4 (2018), 1359–1372. DOI: https://doi.org/10.1137/17M1157970

[2] László Babai. 2016. Graph isomorphism in quasipolynomial time [extended abstract]. In Proceedings of the 48th ACM SIGACT Symposium on Theory of Computing (STOC’16), Daniel Wichs and Yishay Mansour (Eds.). ACM, 684–697. DOI: https://doi.org/10.1145/2897518.2897542

[3] László Babai. 2019. Canonical form for graphs in quasipolynomial time: Preliminary report. In Proceedings of the 51st ACM SIGACT Symposium on Theory of Computing (STOC’19), Moses Charikar and Edith Cohen (Eds.). ACM, 1237–1246. DOI: https://doi.org/10.1145/3313276.3316356

[4] László Babai, William M. Kantor, and Eugene M. Luks. 1983. Computational complexity and the classification of finite simple groups. In Proceedings of the 24th Symposium on Foundations of Computer Science. IEEE Computer Society, 162–171. DOI: https://doi.org/10.1109/SFCS.1983.10

[5] László Babai and Eugene M. Luks. 1983. Canonical labeling of graphs. In Proceedings of the 15th ACM Symposium on Theory of Computing, 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, 171–183. DOI: https://doi.org/10.1145/800061.808746

[6] Hans L. Bodlaender. 1990. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. J. Algor. 11, 4 (1990), 631–643. DOI: https://doi.org/10.1016/0196-6774(90)90013-5

[7] Hans L. Bodlaender. 2003. Necessary edges in k-chordalisations of graphs. J. Comb. Optim. 7, 3 (2003), 283–290. DOI: https://doi.org/10.1023/A:1027320705349

[8] François Clautiaux, Jacques Carlier, Aziz Moukrim, and Stéphane Négre. 2003. New lower and upper bounds for graph treewidth. In Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA’03). 70–80. DOI: https://doi.org/10.1007/3-540-44867-5_6

[9] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2015. Parameterized Algorithms. Springer. DOI: https://doi.org/10.1007/978-3-319-21275-3

[10] Michael Elberfeld and Pascal Schweitzer. 2017. Canonizing graphs of bounded tree width in logspace. ACM Trans. Comput. Theory 9, 3 (2017), 12:1–12:29. DOI: https://doi.org/10.1145/3132720

[11] Martin Grohe and Dániel Marx. 2015. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. SIAM J. Comput. 44, 1 (2015), 114–159. DOI: https://doi.org/10.1137/120892234

[12] Martin Grohe and Daniel Neuen. 2019. Canonisation and definability for graphs of bounded rank width. In Proceedings of the 34th ACM/IEEE Symposium on Logic in Computer Science (LICS’19). IEEE, 1–13. DOI: https://doi.org/10.1109/LICS.2019.8785682

[13] Martin Grohe, Daniel Neuen, and Pascal Schweitzer. 2018. A faster isomorphism test for graphs of small degree. In Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS’18), Mikkel Thorup (Ed.). IEEE Computer Society, 89–100. DOI: https://doi.org/10.1109/FOCS.2018.00018

[14] Martin Grohe and Pascal Schweitzer. 2015. Isomorphism testing for graphs of bounded rank width. In Proceedings of the IEEE 56th Symposium on Foundations of Computer Science (FOCS’15), Venkatesan Guruswami (Ed.). IEEE Computer Society, 1010–1029. DOI: https://doi.org/10.1109/FOCS.2015.66

[15] Martin Grohe and Pascal Schweitzer. 2015. Isomorphism testing for graphs of bounded rank width. CoRR abs/1505.03737 (2015).

[16] Hans-Georg Leimer. 1993. Optimal decomposition by clique separators. Discrete Math. 113, 1-3 (1993), 99–123. DOI: https://doi.org/10.1016/0012-365X(93)90051-Z

[17] Martin W. Liebeck and Aner Shalev. 1999. Simple groups, permutation groups, and probability. J. Amer. Math. Soc. 12, 2 (1999), 497–520. DOI: https://doi.org/10.1090/S0894-0347-99-00288-X

[18] Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. 2017. Fixed-parameter tractable canonical isomorphism and isomorphism test for graphs of bounded treewidth. SIAM J. Comput. 46, 1 (2017), 161–189. DOI: https://doi.org/10.1137/140999980

[19] Eugene M. Luks. 1982. Isomorphism of graphs of bounded valence can be tested in polynomial time. J. Comput. Syst. Sci. 25, 1 (1982), 42–65. DOI: https://doi.org/10.1016/0022-0000(82)90009-5

[20] Rudolf Mathon. 1979. A note on the graph isomorphism counting problem. Inf. Proc. Lett. 8, 3 (1979), 131–132. DOI: https://doi.org/10.1016/0019-9780(79)90004-8

[21] Gary L. Miller. 1983. Isomorphism testing and canonical forms for k-contractable graphs (A generalization of bounded valence and bounded genus). In Foundations of Computation Theory. Springer Berlin, 310–327. DOI: https://doi.org/10.1007/3-540-12689-9_114
An Improved Isomorphism Test for Bounded-tree-width Graphs

[22] Daniel Neuen. 2016. Graph isomorphism for unit square graphs. In Proceedings of the 24th European Symposium on Algorithms (ESA’16), (LIPIcs), Piotr Sankowski and Christos D. Zaroliagis (Eds.), Vol. 57. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 70:1–70:17. DOI: https://doi.org/10.4230/LIPIcs.ESA.2016.70

[23] Yota Otachi and Pascal Schweitzer. 2014. Reduction techniques for graph isomorphism in the context of width parameters. In Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT’14), (Lecture Notes in Computer Science), R. Ravi and Inge Li Gørtz (Eds.), Vol. 8503. Springer, 368–379. DOI: https://doi.org/10.1007/978-3-319-08404-6_32

[24] Ilia N. Ponomarenko. 1991. The isomorphism problem for classes of graphs closed under contraction. J. Soviet Math. 55, 2 (01 June 1991), 1621–1643. DOI: https://doi.org/10.1007/BF01098279

[25] Daniel Wiebking. 2019. Graph isomorphism in quasipolynomial time parameterized by treewidth. CoRR abs/1911.11257 (2019).

Received April 2019; revised January 2020; accepted February 2020.