Polynomial-time Algorithm for Isomorphism of Graphs with Clique-width at most Three

Bireswar Das*, Murali Krishna Enduri** and I. Vinod Reddy

IIT Gandhinagar, India
{bireswar,endurimuralikrishna,reddy_vinod}@iitgn.ac.in

Abstract. The clique-width is a measure of complexity of decomposing graphs into certain tree-like structures. The class of graphs with bounded clique-width contains bounded tree-width graphs. We give a polynomial time graph isomorphism algorithm for graphs with clique-width at most three. Our work is independent of the work by Grohe et al. [16] showing that the isomorphism problem for graphs of bounded clique-width is polynomial time.

1 Introduction

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection $f : V_1 \rightarrow V_2$ such that $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$. Given a pair of graphs as input the problem of deciding if the two graphs are isomorphic is known as graph isomorphism problem ($\text{GI}$). Despite nearly five decades of research the complexity status of this problem still remains unknown. The graph isomorphism problem is not known to be in $\text{P}$. It is in $\text{NP}$ but very unlikely to be $\text{NP}$-complete [5]. The problem is not even known to be hard for $\text{P}$. Recently Babai [2] designed a quasi-polynomial time algorithm to solve the GI problem improving the previously best known $2^{O(\sqrt{n \log n})}$ time algorithm [1,25]. Although the complexity of the general graph isomorphism problem remains elusive, many polynomial time algorithms are known for restricted classes of graphs e.g., bounded degree [20], bounded genus [22], bounded tree-width [3], etc.

The graph parameter clique-width, introduced by Courcelle et al. in [7], has been studied extensively. The class of bounded clique-width graphs is fairly large in the sense that it contains distance hereditary graphs, bounded tree-width graphs, bounded rank-width graphs [18], etc. Fellows et al. [14] shows that the computing the clique-width of a graph is NP-hard. Oum and Seymour [23] gave an elegant algorithm that computes a $(2^{3k+2} - 1)$-expression for a graph $G$ of clique-width at most $k$ or decides that the clique-width is more than $k$.

The parameters tree-width and clique-width share some similarities, for example many $\text{NP}$-complete problems admit polynomial time algorithms when the tree-width or the clique-width of the input graph is bounded. A polynomial time...

* Part of the research was done while the author was a DIMACS postdoctoral fellow.
** Supported by Tata Consultancy Services (TCS) research fellowship.
an isomorphism algorithm for bounded tree-width graphs has been known for a long
time [3]. Recently Lokhstanov et al. [19] gave an fpt algorithm for GI parameterized by tree-width. The scenario is different for bounded clique-width graphs.

The complexity of GI for bounded clique-width graphs is not known. Polynomial
time algorithm for GI for graphs with clique-width at most 2, which coincides with the class of co-graphs, is known probably as a folklore. The complexity of recognizing graphs with clique-width at most three was unknown until Corneil et al. [6] came up with the first polynomial time algorithm. Their algorithm (henceforth called the CHLRR algorithm) works via an extensive study of the structure of such graphs using split and modular decompositions. Apart from recognition, the CHLRR algorithm also produces a 3-expression for graphs with clique-width at most three. For fixed $k > 3$, though algorithms to recognize graphs with clique-width at most $k$ are known [23], computing a $k$-expression is still open. Recently in an independent work by Grohe et al. [16] designed an isomorphism algorithm for graphs of bounded clique-width subsuming our result. Their algorithm uses group theory techniques and has worse runtime. However our algorithm has better runtime and uses different simpler intuitive techniques.

In this paper we give isomorphism algorithm for graphs with clique-width
at most three with runtime $O(n^3m)$. Our algorithm works via first defining a notion of equivalent $k$-expression and designing $O(n^3)$ algorithm to test if two input $k$-expressions are equivalent under this notion. Next we modify the
CHLRR algorithm slightly to output a linear sized set $parseG$ of 4-expressions for an input graph $G$ of clique-width at most three which runs in $O(n^3m)$ time. Note that modified CHLRR algorithm will not output a canonical expression. However we show that for two isomorphic graphs $G$ and $H$ of clique-width at most three, $parseG$ contains an equivalent $k$-expression for each $k$-expression in $parseH$ and vice versa. Moreover, if $G$ and $H$ are not isomorphic then no pair in $parseG \times parseH$ is equivalent.

2 Preliminaries

In this paper, the graphs we consider are without multiple edges and self loops.
The complement of a graph $G$ is denoted as $\overline{G}$. The connected components of $G$ are the connected components of $\overline{G}$. We say that a vertex $v$ is universal to a vertex set $X$ if $v$ is adjacent to all vertices in $X \setminus \{v\}$. A biclique is a bipartite graph $(G, X, Y)$, such that every vertex in $X$ is connected to every vertex of $Y$. A labeled graph is a graph with labels assigned to vertices such that each vertex has exactly one label. In a labeled graph $G$, $lab(v)$ is the label of a vertex $v$ and $lab(G)$ is the set of all labels. We say that a graph is bilabeled (trilabeled) if it is labeled using exactly two (three) labels. The set of all edges between vertices of label $a$ and label $b$ is denoted $E_{ab}$. We say $E_{ab}$ is complete if it corresponds to a biclique.

The subgraph of $G$ induced by $X \subseteq V(G)$ is denoted by $G[X]$, the set of vertices adjacent to $v$ is denoted $N_G(v)$. The closed neighborhood $N_G[v]$ of $v$ is $N_G(v) \cup \{v\}$. We write $G \cong_f H$ if $f$ is an isomorphism between graphs.
G and H. For labeled graphs G and H, we write $G \cong_j^f H$ if $G \cong_j^f H$ and \( \pi : \text{lab}(G) \to \text{lab}(H) \) is a bijection such that for all \( x \in V(G) \) if \( \text{lab}(x) = i \) then \( \text{lab}(f(x)) = \pi(i) \). The set of all isomorphisms from G to H is denoted \( \text{ISO}(G, H) \).

**Definition 1.** The **clique-width** of a graph G is defined as the minimum number of labels needed to construct G using the following four operations:

1. \( v(i) \): Creates a new vertex \( v \) with label \( i \)
2. \( G_1 \oplus G_2 \cdots \oplus G_l \): Disjoint union of labeled graphs \( G_1, G_2, \ldots, G_l \)
3. \( \eta_{i,j} \): Joins each vertex with label \( i \) to each vertex with label \( j \) \((i \neq j)\)
4. \( \rho_{i \to j} \): Renames all vertices of label \( i \) with label \( j \)

Every graph can be constructed using the above four operations, which is represented by an algebraic expression known as a **k-expression**, where \( k \) is the number of labels used in expression. The **clique-width** of a graph G, denoted by \( \text{cwd}(G) \), is the minimum \( k \) for which there exists a k-expression that defines the graph G. From the k-expression of a graph we can construct a tree known as **parse tree** of G. The leaves of the parse tree are vertices of G with their initial labels, and the internal nodes correspond to the operations \( (\eta_{i,j}, \rho_{i \to j} \text{ and } \oplus) \) used to construct G. For example, \( C_5 \) (cycle of length 5) can be constructed by

\[
\eta_{1,3}(\rho_{3 \to 2}(\eta_{2,3}((\eta_{1,2}(a(1) \oplus b(2))) \oplus (\eta_{1,3}(c(3) \oplus d(1)))))) \oplus e(3)).
\]

The k-expression for a graph need not be unique. The clique-width of any induced subgraph is at most the clique-width of its graph \( G \).

Now we describe the notions of modular and split decompositions. A set \( M \subseteq V(G) \) is called a **module** of G if all vertices of M have the same set of neighbors in \( V(G) \setminus M \). The trivial modules are \( V(G) \) and \( \{v\} \) for all \( v \). In a labeled graph, a module is said to be a **l-module** if all the vertices in the module have the same label. A **prime (l-prime) graph** is a graph (labeled graph) in which all modules (l-modules) are trivial. The modular decomposition of a graph is one of the decomposition techniques which was introduced by Gallai [15]. The **modular decomposition** of a graph G is a rooted tree \( T_M^G \) that has the following properties:

1. The leaves of \( T_M^G \) are the vertices of G.
2. For an internal node \( h \) of \( T_M^G \), let \( M(h) \) be the set of vertices of G that are leaves of the subtree of \( T_M^G \) rooted at \( h \). \( M(h) \) forms a module in G).
3. For each internal node \( h \) of \( T_M^G \) there is a graph \( G_h \) (representative graph) with \( V(G_h) = \{h_1, h_2, \ldots, h_r\} \), where \( h_1, h_2, \ldots, h_r \) are the children of \( h \) in \( T_M^G \) and for \( 1 \leq i < j \leq r \), \( h_i \) and \( h_j \) are adjacent in \( G_h \) iff there are vertices \( u \in M(h_i) \) and \( v \in M(h_j) \) that are adjacent in G.
4. \( G_h \) is either a clique, an independent set, or a prime graph and \( h \) is labeled **Series** if \( G_h \) is clique, **Parallel** if \( G_h \) is an independent set, and **Prime** otherwise.

James et al. [17] gave first polynomial time algorithm for finding a modular decomposition which runs in \( O(n^4) \) time. Linear time algorithms to find modular decompositions are proposed in [10][24].
A vertex partition \((A, B)\) of a graph \(G\) is a split if \(\hat{A} = A \cap N(B)\) and \(\hat{B} = B \cap N(A)\) forms a biclique. A split is trivial if \(|A|\) or \(|B|\) is one. Split decomposition was introduced by Cunningham [11]. Loosely it is the result of a recursive process of decomposing a graph into components based on the splits. Cunningham [11] showed that a graph can be decomposed uniquely into components that are stars, cliques, or prime (i.e., without proper splits). This decomposition is known as the skeleton. For details see [12]. A polynomial time algorithm for computing the skeleton of a graph is given in [21].

Theorem 1. [13] (see [6]) Let \(G\) be a connected graph. Then the skeleton of \(G\) is unique, and the proper splits of \(G\) correspond to the special edges of its skeleton and to the proper splits of its complete and star components.

Organization of the paper: In Section 3 we discuss GI-completeness of prime graph isomorphism. In Section 4 we define a notion of equivalence of parse trees called structural isomorphism, and give an algorithm to test if two parse trees are structurally isomorphic. We give an overview of the CHLRR algorithm in Section 5. In Section 6 we present the isomorphism algorithm for prime graphs of clique-width at most three. In Appendix, we show that the CHLRR algorithm can be modified suitably to output structurally isomorphic parse trees for isomorphic graphs.

3 Completeness of Prime Graph Isomorphism

It is known that isomorphism problem for prime graphs is GI-complete [4]. There is an easy polynomial time many-one reduction from GI to prime graph isomorphism described in Lemma 9 of the Appendix. Unfortunately, this reduction does not preserve the clique-width. We also give a clique-width preserving Turing reduction from GI to prime graph isomorphism which we use in our main algorithm. The reduction hinges on the following lemma.

Lemma 1. \(\mathcal{G}\) \(G\) is a graph of clique-width at most \(k\) iff each prime graph associated with the modular decomposition of \(G\) is of clique-width at most \(k\).

We next show that if we have an oracle for GI for colored prime graphs of clique-width at most \(k\) then there is a GI algorithm for graphs with clique-width at most \(k\).

Theorem 2. Let \(A'\) be an algorithm that given two colored prime graphs \(G'\) and \(H'\) of clique-width at most \(k\), decides if \(G' \cong H'\) via a color preserving isomorphism. Then there exists an algorithm \(A\) that on input any colored graphs \(G\) and \(H\) of clique-width at most \(k\) decides if \(G \cong H\) via a color preserving isomorphism.

Proof. Let \(G\) and \(H\) be two colored graphs of clique-width at most \(k\). The algorithm is similar to [13], which proceeds in a bottom up approach in stages

\footnote{In fact, it is an AC^0 reduction}
starting from the leaves to the root of the modular decomposition trees $T_G$ and $T_H$ of $G$ and $H$ respectively. Each stage corresponds to a level in the modular decomposition. In every level, the algorithm $A$ maintains a table that stores whether for each pair of nodes $x$ and $y$ in $T_G$ and $T_H$ the subgraphs $G[x]$ and $H[y]$ induced by leaves of subtrees of $T_G$ and $T_H$ rooted at $x$ and $y$ are isomorphic. For the leaves it is trivial to store such information. Let $u$ and $v$ be two internal nodes in the modular decomposition trees of $T_G$ and $T_H$ in the same level. To decide if $G[u]$ and $H[v]$ are isomorphic $A$ does the following.

If $u$ and $v$ are both series nodes then it just checks if the children of $u$ and $v$ can be isomorphically matched. The case for parallel node is similar. If $u$ and $v$ are prime nodes then the vertices of representative graphs $G_u$ and $H_u$ are colored by their isomorphism type i.e., two internal vertices $u_1$ and $u_2$ of the representative graphs will get the same color iff subgraphs induced by leaves of subtrees of $T_G$ (or $T_H$) rooted at $u_1$ and $u_2$ are isomorphic. To test $G[u] \cong H[v]$, $A$ calls $A'(G_u, H_v)$, where $G_u$ and $H_v$ are the colored copies of $G_u$ and $H_v$ respectively. At any level if we can not find a pairwise isomorphism matching between the internal nodes in that level of $T_G$ and $T_H$ then $G \cong H$. In this manner we make $O(n^2)$ calls to algorithm $A'$ at each level. The total runtime of the algorithm is $O(n^3)T(n)$, where $T(n)$ is run time of $A'$. Note that by Lemma 4, clique-width of $G_u$ and $H_v$ are at most $k$.

4 Testing Isomorphism between Parse Trees

In this section we define a notion of equivalence of parse trees called structural isomorphism, and we give an algorithm to test if two given parse trees are equivalent under this notion. As we will see, the graphs generated by equivalent parse trees are always isomorphic. Thus, if we have two equivalent parse trees for the two input graphs, the isomorphism problem indeed admits a polynomial time algorithm. In Section 3 we prove that the CHLRR algorithm can be tweaked slightly to produce structurally isomorphic parse trees for isomorphic graphs with clique-width at most three and thus giving a polynomial-time algorithm for such graphs.

Let $G$ and $H$ be two colored graphs. A bijective map $\pi : V(G) \to V(H)$ is color consistent if for all vertices $u$ and $v$ of $G$, $\text{color}(u) = \text{color}(v)$ iff $\text{color}(\pi(u)) = \text{color}(\pi(v))$. Let $\pi : V(G) \to V(H)$ be a color consistent mapping, define $\pi/\text{color} : \text{color}(G) \to \text{color}(H)$ as follows: for all $c$ in $\text{color}(G)$, $\pi/\text{color}(c) = \text{color}(\pi(v))$ where $\text{color}(v) = c$. It is not hard to see that the map $\pi/\text{color}$ is well defined. Recall that the internal nodes of a parse tree are $\eta_{i,j}$, $\rho_{i\to j}$, and $\oplus$ operations. The levels of a parse tree correspond to $\oplus$ nodes. Let $T_g$ be a parse tree of $G$ rooted at $\oplus$ node $g$. Let $g_1$ be descendant of $g$ which is neither $\eta$ nor $\rho$. We say that $g_1$ is an immediate significant descendant of $g$ if there is no other $\oplus$ node in the path from $g$ to $g_1$. For an immediate significant descendant $g_1$ of $g$, we construct a colored quotient graph $Q_{g_1}$ that corresponds to graph operations appearing in the path from $g$ to $g_1$ performed on graph $G_{g_1}$, where $G_{g_1}$ is graph generated by parse tree $T_{g_1}$. The vertices of $Q_{g_1}$ are labels of
Let \( T_g \) and \( T_h \) be two parse trees of \( G \) and \( H \) rooted at \( \oplus \) nodes \( g \) and \( h \) respectively. We say that \( T_g \) and \( T_h \) are structurally isomorphic via a label map \( \pi \) (denoted \( T_g \cong^\pi T_h \))

1. If \( T_g \) and \( T_h \) are single node\(^2\) or inductively,
2. If \( T_g \) and \( T_h \) are rooted at \( g \) and \( h \) having immediate significant descendants \( g_1, \ldots, g_r \) and \( h_1, \ldots, h_r \), and there is a bijection \( \gamma : [r] \rightarrow [r] \) and for each \( i \) there is a \( \pi_i \in \text{ISO}(Q_{g_i}, Q_{h_{\gamma(i)}}) \) such that \( T_{g_i} \cong^\pi T_{h_{\gamma(i)}} \) and \( \pi_i/\text{color} = \pi_j/\text{color} \), where \( T_{g_1}, \ldots, T_{g_r} \) and \( T_{h_1}, \ldots, T_{h_r} \) are the subtrees rooted at \( g_1, \ldots, g_r \) and \( h_1, \ldots, h_r \) respectively.\(^3\)

We say that \( T_g \) and \( T_h \) are structurally isomorphic if there is a \( \pi \) such that \( T_g \cong^\pi T_h \).

The structural isomorphism is an equivalence relation: reflexive and symmetric properties are immediate from the above definition. The following lemma shows that it is also transitive.

**Lemma.** Let \( T_{g_1}, T_{g_2} \) and \( T_{g_3} \) be the parse trees of \( G_1, G_2 \) and \( G_3 \) respectively such that \( T_{g_1} \cong^\pi T_{g_2} \) and \( T_{g_2} \cong^\pi T_{g_3} \) then \( T_{g_1} \cong^\pi T_{g_3} \).

**Proof.** The proof is by induction on the height of the parse trees. The base case trivially satisfies the transitive property. Assume that \( g_1 \), \( g_2 \) and \( g_3 \) are nodes of height \( d + 1 \). Let \( g_{1i} \) be an immediate significant descendant of \( g_1 \). Since \( T_{g_1} \cong^\pi T_{g_2} \), there is an immediate significant descendant \( g_{2j} \) of \( g_2 \) and \( \pi_{1i} \in \text{ISO}(Q_{g_{1i}}, Q_{g_{2j}}) \) such that \( \pi_{1i}/\text{color} = \pi_j/\text{color} \) and \( T_{g_{1i}} \cong^\pi T_{g_{2j}} \). Similarly, \( g_{2j} \) will be matched to some immediate significant descendant \( g_{3k} \) of \( g_3 \) via \( \pi_{2j} \in \text{ISO}(Q_{g_{2j}}, Q_{g_{3k}}) \) such that \( \pi_{2j}/\text{color} = \pi_j/\text{color} \) and \( T_{g_{2j}} \cong^\pi T_{g_{3k}} \). The nodes \( g_{1i}, g_{2j} \) and \( g_{3k} \) has height at most \( d \). Therefore, by induction hypothesis \( T_{g_{1i}} \cong^\pi T_{g_{3k}} \). By transitivity of isomorphism we can say \( \pi_{2j}\pi_{1i} \in \text{ISO}(Q_{g_{2j}}, Q_{g_{3k}}) \). To complete the proof we just need to show \( \pi_{2j}\pi_{1i}/\text{color} = \pi_{2j}\pi_{1i}/\text{color} \). This can be inferred from the following two facts:

1. \( \pi_{2j}\pi_{1i}/\text{color} = \pi_j/\text{color} \)
2. \( \pi_{2j}\pi_{1i}/\text{color} = \pi_j/\text{color} \). \( \square \)

\(^2\) In this case they are trivially structurally isomorphic via \( \pi \).

\(^3\) Notice that this definition implies that \( G_g \) and \( H_{h_{\gamma(i)}} \) are isomorphic via the label map \( \pi_i \) where \( G_g \) and \( H_{h_{\gamma(i)}} \) are graphs generated by the parse trees \( T_g \) and \( T_{h_{\gamma(i)}} \) respectively.
**Algorithm to Test Structural Isomorphism:** Next we describe an algorithm that given two parse trees $T_G$ and $T_H$ tests if they are structurally isomorphic. From the definition if $T_G \cong T_H$ then we can conclude that $G$ and $H$ are isomorphic. We design a dynamic programming algorithm that basically checks the local conditions 1 and 2 in Definition 2.

The algorithm starts from the leaves of parse trees and proceeds in levels where each level corresponds to $\oplus$ operations of parse trees. Let $g$ and $h$ denote the $\oplus$ nodes at level $l$ of $T_G$ and $T_H$ respectively. At each level $l$, for each pair of $\oplus$ nodes $(g, h) \in (T_G, T_H)$, the algorithm computes the set $R_l^{g,h}$ of all bijections $\pi : \text{lab}(G_g) \rightarrow \text{lab}(H_h)$ such that $G_g \cong f H_h$ for some $f$, and stores in a table indexed by $(l, g, h)$, where $G_g$ and $H_h$ are graphs generated by sub parse trees $T_g$ and $T_h$ rooted at $g$ and $h$ respectively. To compute $R_l^{g,h}$, the algorithm uses the already computed information $R_{l+1}^{g,h_1}$ where $g_i$ and $h_j$ are immediate significant descendants of $g$ and $h$.

The base case corresponds to finding $R_l^{g,h}$ for all pairs $(g, h)$ such that $g$ and $h$ are leaves. Since in this case $G_g$ and $H_h$ are just single vertices, it is easy to find $R_l^{g,h}$. For the inductive step let $g_1, \ldots, g_r$ and $h_1, \ldots, h_r$, be the immediate significant descendants of $g$ and $h$ respectively. If $r \neq r'$ then $R_l^{g,h} = \emptyset$. Otherwise we compute $R_l^{g,h}$ for each pair $(g, h)$ at level $l$ with help of the already computed information up to level $l + 1$ as follows.

For each $\pi : \text{lab}(G_g) \rightarrow \text{lab}(H_h)$ and pick $g_1$ and try to find a $h_i$ such that $T_{g_1} \cong T_{h_i}$ for some $\pi_1 \in \text{ISO}(Q_{g_1}, Q_{h_i}) \cap R_{l+1}^{g_1,h_i}$ such that $\pi_1/\text{color} = \pi|_{\text{color}(Q_{g_1})}$. We do this process to pair $g_2$ with some unmatched $h_i$. Continue in this way until all immediate significant descendants are matched. By Lemma 3 we know that this greedy matching satisfies the conditions of Definition 2. If all the immediate significant descendants are matched we add $\pi$ to $R_l^{g,h}$. It is easy to see that if $R_l^{g,h} \neq \emptyset$ then the subgraphs $G_g \cong f H_h$ for $\pi \in R_l^{g,h}$. From the definition of structurally isomorphic parse trees it is clear that if $R_0^{g,h} \neq \emptyset$ then $G \cong H$. The algorithm is polynomial time as the number of choices for $\pi$ and $\pi_i$ is at most $k!$ which is a constant, where $|\text{lab}(G)| = k$.

Note that for colored graphs, by ensuring that we only match vertices of same color in the base case, the whole algorithm can be made to work for colored graphs. In Lemma 3 we prove that structural isomorphism satisfies transitivity. In fact, structural isomorphism satisfies a stronger notion of transitivity as stated in the following lemma.

**Lemma 3.** Let $T_g$ and $T_h$ be two parse trees of graphs $G$ and $H$. Let $g_i$ and $g_2$ be two immediate significant descendants of $g$, and $h_1$ and $h_2$ be two immediate significant descendants of $h$. Suppose for $i = 1, 2$, $T_{g_i} \cong T_{h_i}$ for some $\pi_i \in \text{ISO}(Q_{g_i}, Q_{h_i})$ with $\pi_i/\text{color} = \pi|_{\text{color}(Q_{g_i})}$. Also assume that $T_{g_1} \cong T_{h_2}$ where $\pi_3 \in \text{ISO}(Q_{g_1}, Q_{h_2})$ and $\pi_3/\text{color} = \pi|_{\text{color}(Q_{g_1})}$. Then, $T_{g_2} \cong T_{h_1}$ where $\pi_1 \pi_3^{-1} \pi_2 \in \text{ISO}(Q_{g_2}, Q_{h_1})$ and $\pi_1 \pi_3^{-1} \pi_2/\text{color} = \pi|_{\text{color}(Q_{g_2})}$.

**Proof.** By Lemma 2 $T_{g_2} \cong T_{h_1}$. The rest of the proof is similar to the proof of the inductive case of Lemma 2. \[\square\]
5 Overview of the CHLRR Algorithm

Corneil et al. [6] gave the first polynomial time algorithm (the CHLRR algorithm), to recognize graphs of clique-width at most three. We give a brief description of their algorithm in this section. We mention that our description of this fairly involved algorithm is far from being complete. The reader is encouraged to see [6] for details. By Lemma 7 we assume that the input graph $G$ is prime.

To test whether clique-width of prime graph $G$ is at most three the algorithm starts by constructing a set of bilabelings and trilabelings of $G$. In general the number of bilabelings and trilabelings are exponential, but it was shown (Lemma 8 and 9 in [6]) that it is enough to consider the following linear size subset denoted by $\text{Lab}_G$.

1. For each vertex $v$ in $V(G)$
   - $[B_1]$ Generate the bilabeling $\{v\}$ and add it to $\text{Lab}_G$.
   - $[B_2]$ Generate the bilabeling $\{x \in N(v) \mid N[x] \subseteq N[v]\}$ and add it to $\text{Lab}_G$.
2. Compute the skeleton of $G$ search this skeleton for the special edges, clique and star components.
   - $[T_1]$ For each special edge $s$ (corresponds to a proper split), generate the trilabeling $\tilde{X}$, $\tilde{Y}$, $V(G) \setminus (\tilde{X} \cup \tilde{Y})$ where $(X,Y)$ is the split defined by $s$ and add it to $\text{Lab}_G$.
   - $[B_3]$ For all clique components $C$, generate the bilabeling $C$ and add it to $\text{Lab}_G$.
   - $[B_4]$ For all star components $S$, generate the bilabeling $\{c\}$, where $c$ is the special center of $S$, and add it to $\text{Lab}_G$.

Lemma 4. [6] Let $G$ be a prime graph. Clique-width of $G$ is at most three if and only if at least one of the bilabelings or trilabelings in $\text{Lab}_G$ has clique-width at most three.

By Lemma 4 the problem of testing whether $G$ is of clique-width at most three is reduced to checking one of labeled graph in $\text{Lab}_G$ is of clique-width at most three. To test if a labeled graph $A$ taken from $\text{Lab}_G$ is of clique-width at most three, the algorithm follows a top down approach by iterating over all possible last operations that arise in the parse tree representation of $G$. For example, for each vertex $x$ in $G$ the algorithm checks whether the last operation must have joined $x$ with its neighborhood. In this case the problem of testing whether $G$ can be constructed using at most three labels is reduced to test whether $G \setminus \{x\}$ can be constructed using at most three lables. Once the last operations are fixed the original graph decomposes into smaller components, which can be further decomposed recursively.

For each $A$ in $\text{Lab}_G$, depending on whether it is bilabeled or trilabeled the algorithm makes different tests on $A$ to determine whether $A$ is of clique-width

---

4 *bilabeling* of a set $X \subseteq V$ indicates that all the vertices in $X$ are labeled with one label and $V \setminus X$ is labeled with another label.
at most three. Based on the test results the algorithm either concludes clique-width of \( A \) is more than three or returns top operations of the parse tree for \( A \) along with some connected components of \( A \) which are further decomposed recursively.

If \( A \) in \( LabG \) is connected, trilabeled (with labels \( l_1, l_2, l_3 \)) and \( l \)-prime then by the construction of \( LabG \), \( A \) corresponds to a split (possibly trivial). If \( A \) has a proper split then there exists \( a \neq b \) in \( \{l_1, l_2, l_3\} \) such that \( A \) will be disconnected with the removal of edges \( E_{a\overline{b}} \). This gives a decomposition with top operations \( \eta_{a,b} \) followed by a \( \oplus \) node whose children are connected components of \( A \setminus E_{a\overline{b}} \). If \( A \) has a universal vertex \( v \) (trivial split) labeled \( a \) in \( A \) then by removing edges \( E_{a\overline{b}} \) and \( E_{ac} \) we get a decomposition with top operations \( \eta_{a,b} \) and \( \eta_{a,c} \) followed by a \( \oplus \) operation with children connected components of \( A \setminus (E_{a\overline{b}} \cup E_{ac}) \).

To describe the bilabeled case we use \( V_i \) to denote the set of vertices of \( A \) with label \( i \). If \( A \) in \( LabG \) is connected, bilabeled (with labels \( l_1, l_2 \)) and \( l \)-prime, then the last operation is either \( \eta_{l_1,l_2} \) (otherwise \( A \) will have a \( l \)-module) nor \( \oplus \) (\( A \) is connected). So the last operation of the decomposition must be a relabeling followed by a join operation i.e., we have to introduce a third label set \( V_3 \) such that all the edges are present between the two of three labeled sets.

After introducing third label if there is only one join to undo, then we have a unique way to decompose the graph into smaller components. If there are more than one possible join to be removed, then it is enough to consider one of them and proceed (see Section 5.2 in [6]). There are four ways to introduce the third label to decompose the graph, but they might correspond to overlapping cases. To overcome this the algorithm first checks whether \( A \) belongs to any of three simpler cases described below.

**PC1**: \( A \) has a universal vertex \( x \) of label \( l \in \{l_1, l_2\} \). In this case relabel vertex \( x \) with \( l_3 \) and remove the edges \( E_{l_3l_2} \) and \( E_{l_3l_1} \) to decompose \( A \). This gives a decomposition with \( \rho_{l_3\overrightarrow{l_1}}, \eta_{l_3,l_2}, \eta_{l_3,l_1} \) followed by \( \oplus \) operation with children \( x \) and \( A \setminus \{x\} \).

**PC2**: \( A \) has a vertex \( x \) of label \( l \in \{l_1, l_2\} \) that is universal to all vertices of label \( l' \in \{l_1, l_2\} \), but is not adjacent to all vertices with the other label, say \( l'' \). In this case relabel vertex \( x \) with \( l_3 \) and remove the edges \( E_{l_3l_{l''}} \). This gives a decomposition with \( \rho_{l_3\overrightarrow{l_1}}, \eta_{l_3,l_{l''}} \) above a \( \oplus \) operation with children \( x \) and \( A \setminus \{x\} \).

**PC3**: \( A \) has two vertices \( x \) and \( y \) of label \( l \), where \( y \) is universal to everything other than \( x \), and \( x \) is universal to all vertices of label \( l \) other than \( y \), and non-adjacent to all vertices with the other label \( l \). In this case the algorithm relabels vertices \( x \) and \( y \) with \( l_3 \), and by removing edges \( E_{l_3,l} \) disconnects the graph \( A \), with two connected components \( x \) and \( A \setminus \{x\} \). Now in graph \( A \setminus \{x\} \) again remove the edges \( E_{l_3,l} \) to decompose the graph into two parts \( y \) and \( A \setminus \{x,y\} \).

If \( A \) does not belongs to any of above three simpler cases then there are four different ways to introduce the third label set to decompose the graph as described below.

Let \( E \) be the set of all connected, bilabeled, \( l \)-prime graphs with clique-width at most three and not belonging to above three simpler cases. For \( l \in \{1, 2\} \) we define the following four subsets of \( E \).
1. $\mathcal{U}_l$: $V^a \neq \emptyset$ and removing the edges between the $V^a$ and $V_l$ disconnects the graph.

2. $\mathcal{D}_l$: $V_l$ is not connected and removing the edges between the coconnected components of $V_l$ disconnects the graph.

In these four cases the algorithm introduces a new label $l_3$ and removes the edges $E_{l_3}$, $l \in \{l_1, l_2\}$ to disconnect $A$. This gives a decomposition with $\rho_{l_3 \to l}$ and $\eta_{l_3}$, followed by $\oplus$ operation with children that are the connected components of $A \setminus E_{l_3}$. For more details about decomposition process when $A$ is in $\mathcal{U}_l$ or $\mathcal{D}_l$, $l \in \{1, 2\}$ the reader is encouraged to see Section 5.2 in [6].

The following Lemma shows that there is no other possible way of decomposing a clique-width at most three graphs apart from the cases described above.

**Lemma 5.** $E = \mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{D}_1 \cup \mathcal{D}_2$, and this union is disjoint.

In summary, for any labeled graph $A$ in $\text{Lab}G$ the CHLRR algorithm tests whether $A$ belongs to any of the above described cases, if it is then it outputs suitable top operations and connected components. The algorithm continues the above process repeatedly on each connected component of $A$ until it either returns a parse tree or concludes clique-width of $A$ is more than three.

### 6 Isomorphism Algorithm for Prime Graphs of Clique-width at most Three

In Section 4 we described algorithm to test structural isomorphism between two parse trees. In this Section we show that given two isomorphic prime graphs $G$ and $H$ of clique-width at most three, the CHLRR algorithm can be slightly modified to get structurally isomorphic parse trees. We have used four labels in order to preserve structural isomorphism in the modified algorithm. The modified algorithm is presented in Appendix. Recall that the first step of the CHLRR algorithm is to construct a set $\text{Lab}G$ of bilabelings and trilabelings of $G$ as described in Section 5.

**Definition 3.** We say that $\text{Lab}G$ is equivalent to $\text{Lab}H$ denoted as $\text{Lab}G \equiv \text{Lab}H$ if there is a bijection $g : \text{Lab}G \to \text{Lab}H$ such that for all $A \in \text{Lab}G$, there is an isomorphism $f : V(A) \to V(g(A))$ and a bijection $\pi : \text{lab}(A) \to \text{lab}(g(A))$ such that $A \cong^f \pi g(A)$.

**Lemma 6.** $\text{Lab}G \equiv \text{Lab}H$ iff $G \cong^f H$.

**Proof.** The proof follows from the construction of sets $\text{Lab}G$ and $\text{Lab}H$ from input prime graphs $G$ and $H$ and it is presented in Appendix. \hfill $\square$

**Lemma 7.** Let $A \in \text{Lab}G$ and $B \in \text{Lab}H$. If $A \cong^f B$ for some $f$ and $\pi$ then parse trees generated from $\text{Decompose}$ function (Algorithm 2) for input graphs $A$ and $B$ are structurally isomorphic. More specifically, $\text{Decompose}(A) \cong^f \text{Decompose}(B)$. 

10
Proof. Follows from Lemma 11 and Lemma 12 described in Appendix. The major modifications are done in PC2 case, where we have used four labels in order to preserve structural isomorphism between parse trees.

Isomorphism Algorithm

For two input prime graphs $G$ and $H$ the algorithm works as follows. Using modified CHLRR algorithm, first a parse tree $T_G$ of clique-width at most three is computed for $G$. The parse tree $T_G$ of $G$ is not canonical but from Lemma 6 and 7, we know that if $G \cong H$ then there exists parse tree $T_H$ of $H$, structurally isomorphic to $T_G$. Therefore we compute parse tree of clique-width at most three for each labeled graph in $LabH$. For each such parse tree $T_H$, the algorithm uses the structural isomorphic algorithm described in Section 4 to test the structural isomorphism between parse trees $T_G$ and $T_H$. If $T_G \cong T_H$ for some $T_H$, then we conclude that $G \cong H$. If there is no parse tree of $H$ which is structurally isomorphic to $T_G$ then $G$ and $H$ can not be isomorphic.

Computing a parse tree $T_G$ of $G$ takes $O(n^2m)$ time. As there are $O(n)$ many labeled graphs in $LabH$, computing all possible parse trees for labeled graphs in $LabH$ takes $O(n^3m)$ time. Testing structural isomorphism between two parse trees need $O(n^3)$ time. Therefore the running time to check isomorphism between two prime graphs $G$ and $H$ of clique-width at most three is $O(n^3m)$.

The correctness of the algorithm follows from Lemma 8 and Theorem 3. Lemma 8 shows that if $G \cong H$ then we can always find two structurally isomorphic parse trees $T_G$ and $T_H$ using the modified CHLRR algorithm.

Lemma 8. Let $G$ and $H$ be prime graphs with clique-width at most three. If $G \cong H$ then for every $T_G$ in $parseG$ there is a $T_H$ in $parseH$ such that $T_G$ is structurally isomorphic to $T_H$ where $parseG$ and $parseH$ are the set of parse trees generated by Algorithm 7 on input $LabG$ and $LabH$ respectively.

Proof. If $G \cong H$ then from Lemma 6 we have $LabG \equiv LabH$ i.e., for every $A$ in $LabG$ there is a $B = g(A)$ in $LabH$ such that $A \cong_f B$ for some $f$ and $\pi$. On input such $A$ and $B$ to Lemma 7 we get two parse trees $T_A$ and $T_B$ which are structurally isomorphic.

Theorem 3. Let $G$ and $H$ be graphs with clique-width at most three. Then there exists a polynomial time algorithm to check whether $G \cong H$.

Proof. The proof follows from the prime graph isomorphism of graphs with clique-width at most three described in Lemma 8 and Theorem 2.

References

1. Babai, L.: Moderately exponential bound for graph isomorphism. In: Fundamentals of Computation Theory. pp. 34–50. Springer (1981)
2. Babai, L.: Graph isomorphism in quasipolynomial time. arXiv preprint arXiv:1512.03547 (2015)
3. Bodlaender, H.L.: Polynomial algorithms for graph isomorphism and chromatic index on partial $k$-trees. Journal of Algorithms 11(4), 631–643 (1990)
4. Bonamy, M.: A small report on graph and tree isomorphism. http://bit.ly/1ySeNBn (2010)
5. Boppana, R.B., Hastad, J., Zachos, S.: Does co-NP have short interactive proofs? Information Processing Letters 25(2), 127–132 (1987)
6. Corneil, D.G., Habib, M., Lanlignel, J.M., Reed, B., Rotics, U.: Polynomial-time recognition of clique-width 3 graphs. Discrete Applied Mathematics 160(6), 834–865 (2012)
7. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. Journal of computer and system sciences 46(2), 218–270 (1993)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory of Computing Systems 33(2), 125–150 (2000)
9. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101(1), 77–114 (2000)
10. Cournoyer, A., Habib, M.: A new linear algorithm for modular decomposition. In: Trees in Algebra and Programming CAAP’94, pp. 68–84. Springer (1994)
11. Cunningham, W.H.: A combinatorial decomposition theory. Canadian Journal of Mathematics 32(3), 734–765 (1980)
12. Cunningham, W.H.: Decomposition of directed graphs. SIAM Journal on Algebraic Discrete Methods 3(2), 214–228 (1982)
13. Das, B., Enduri, M.K., Reddy, I.V.: Logspace and fpt algorithms for graph isomorphism for subclasses of bounded tree-width graphs. In: WALCOM: Algorithms and Computation, pp. 329–334. Springer (2015)
14. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is NP-complete. SIAM Journal on Discrete Mathematics 23(2), 909–939 (2009)
15. Gallai, T.: Transitiv orientierbare graphen. Acta Mathematica Hungarica 18(1), 25–66 (1967)
16. Grohe, M., Schweitzer, P.: Isomorphism testing for graphs of bounded rank width. In: Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on. pp. 1010–1029. IEEE (2015)
17. James, L.O., Stanton, R.G., Cowan, D.D.: Graph decomposition for undirected graphs. In: Proceedings of 3rd Southeastern Conference on Combinatorics, Graph Theory, and Computing. pp. 281–290 (1972)
18. Kamiński, M., Lozin, V.V., Milanič, M.: Recent developments on graphs of bounded clique-width. Discrete Applied Mathematics 157(12), 2747–2761 (2009)
19. Lokshtanov, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. In: (FOCS), IEEE 55th Annual Symposium on. pp. 186–195 (2014)
20. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences 25(1), 42–65 (1982)
21. Ma, T.H., Spinrad, J.: An O(n^2) algorithm for undirected split decomposition. Journal of Algorithms 16(1), 145–160 (1994)
22. Miller, G.: Isomorphism testing for graphs of bounded genus. In: Proceedings of 12th annual ACM symposium on Theory of computing. pp. 225–235. ACM (1980)
23. Oum, S.i., Seymour, P.: Approximating clique-width and branch-width. Journal of Combinatorial Theory, Series B 96(4), 514–528 (2006)
24. Tedder, M., Corneil, D., Habib, M., Paul, C.: Simpler linear-time modular decomposition via recursive factorizing permutations. In: Automata, Languages and Programming, pp. 634–645. Springer (2008)
25. Zemlyachenko, V., Konieko, N., Tyshkevich, R.: Graph isomorphism problem (russian). In: The Theory of Computation I, Notes Sci. Sem. LOMI 118 (1982)
Lemma 9. Given two connected graphs \( G_1 \) and \( G_2 \), \( G_1 \cong G_2 \) if and only if \( \hat{G}_1 \cong \hat{G}_2 \).

Proof. Let \( \hat{G}_1 \) and \( \hat{G}_2 \) be graphs obtained after adding new vertices to \( G_1 \) and \( G_2 \) respectively. If \( G_1 \cong G_2 \) then we can find an isomorphism between \( \hat{G}_1 \) and \( \hat{G}_2 \) by extending \( f \) to newly added vertices such that for every new vertex \( y \in \hat{G}_1 \) having neighbor \( x \), \( f \) maps \( y \) to \( z \), where \( z \) is the newly added neighbor of \( f(x) \) in \( G_2 \). For the other direction when \( \hat{G}_1 \cong \hat{G}_2 \), as there are no old vertices of degree one in \( \hat{G}_1 \) and \( \hat{G}_2 \) any isomorphism \( f \) from \( \hat{G}_1 \) to \( \hat{G}_2 \) must map the old vertices of \( \hat{G}_1 \) to the old vertices of \( \hat{G}_2 \). The restriction of \( f \) to the old vertices of \( \hat{G}_1 \) and \( \hat{G}_2 \) is an isomorphism from \( G_1 \) to \( G_2 \).

Lemma 10. \( LabG \equiv LabH \) iff \( G \equiv_f H \).

Proof. It is easy to see that if \( LabG \equiv LabH \) then \( G \equiv_f H \) from the definition. For the other direction, given two graphs \( G \) and \( H \) isomorphic via \( f \), we need to prove that there is a bijection \( g : LabG \rightarrow LabH \) such that for all \( A \in LabG \), there is an isomorphism \( f : V(A) \rightarrow V(g(A)) \) and a bijection \( \pi : lab(A) \rightarrow lab(g(A)) \) such that \( A \equiv_f^g g(A) \).

The proof is divided into five cases based on how bilabelings and trilabelings are generated by CHLRR algorithm described in Section 3.

1. Let \( A \in LabG \) be generated at \( B_1 \) in CHLRR algorithm. Therefore, \( A \) has bilabeling \( \{v\} \). Since \( G \equiv_f H \), there is a graph \( B \in LabH \) which has bilabeling \( \{f(v)\} \). Define \( g(A) = B \) and a bijection \( \pi : lab(A) \rightarrow lab(B) \) such that if \( lab(v) = i \) then \( \pi(i) = lab(f(v)) \) so that \( A \equiv_f^g B \).

2. Let \( A \in LabG \) be generated at \( B_2 \). Thus, \( A \) has bilabeling \( P = \{x \in N(v) \mid N[x] \subseteq N[v]\} \). As \( G \equiv_f H \), there is a graph \( B \in LabH \) with bilabeling \( f(P) = \{f(x) \in N(f(v)) \mid N[f(x)] \subseteq N[f(v)]\} \). Define \( g(A) = B \) and a bijection \( \pi : lab(A) \rightarrow lab(B) \) such that if \( lab(P) = i \) then \( \pi(i) = lab(f(P)) \) so that \( A \equiv_f^g B \).

3. Let \( A \in LabG \) be generated at \( T_1 \) for a special edge \( s \) in the skeleton of \( G \) with trilabeling \( \hat{X}, \hat{Y}, V(G) \setminus (\hat{X} \cup \hat{Y}) \). As \( G \equiv_f H \) and the skeleton of graph is unique (from Theorem 1), we can find a \( B \in LabH \) which is generated for the special edge \( f(s) \) in skeleton of \( H \) which corresponds to trilabeling \( f(\hat{X}), f(\hat{Y}), V(H) \setminus (f(\hat{X} \cup \hat{Y})) \). Define \( g(A) = B \) and a bijection \( \pi : lab(A) \rightarrow lab(B) \) such that if \( lab(\hat{X}) = i_1 \) then \( \pi(i_1) = lab(f(\hat{X})) \), if
The function Decompose is described in following Section and Appendix.

Let \( G \in LabG \) be generated at \( B_3 \) for a clique component \( C \) with bilabeling \( C \). As \( G \cong_f H \), there is a \( B \in LabH \) which is generated for a clique component \( f(C) \) with bilabeling \( f(C) \). Define \( g(A) = B \) and a bijection \( \pi : lab(A) \rightarrow lab(B) \) such that if \( lab(C) = i \) then \( \pi(i) = lab(f(C)) \) so that \( A \cong_f B \).

5. Let \( A \in LabG \) be generated at \( B_4 \) for a star component \( S \) with bilabeling \( \{c\} \), where \( c \) is a special center of \( S \). As \( G \cong_f H \), there is a graph \( B \in LabH \) which is generated for a star component \( f(S) \) with bilabeling \( f(c) \), where \( f(c) \) is a special center of \( f(S) \). Define \( g(A) = B \) and a bijection \( \pi : lab(A) \rightarrow lab(B) \) such that if \( lab(c) = i \) then \( \pi(i) = lab(f(c)) \) so that \( A \cong_f B \). \( \Box \)

8 Generating Structurally Isomorphic Parse Trees

In this section we prove that the modified CHLRR algorithm generates structurally isomorphic parse trees on two isomorphic input graphs. To prove that we also show that the supporting subroutines do the same.

**Algorithm 1:** Finding parse trees for labeled graphs of clique-width at most three in \( LabG \)

Input: \( LabG \) a set of bilabelings and trilabelings of \( G \)

Output: \( parseG = \{ T_A \mid T_A \) is a parse tree of graph \( A \) in \( LabG \) of clique-width at most three \} \)

1. begin
2. \( parseG := \emptyset \)
3. for all \( A \in LabG \) do
4. \( A.parse-tree := null \)
5. \( A.parse-tree = Decompose(A) \)
6. if \( A.parse-tree \neq null \) then
7. \( Add A.parse-tree to parseG \)
8. return (parseG)

The function Decompose \((P)\) in Algorithm 1 finds parse tree of \( P \) if \( cwd(P) \leq 3 \) and it is described in following Section and Appendix.

8.1 Decomposing Trilabeled Graphs

The function Decompose-leaf-TI (Algorithm 3) decomposes trilabeled graph from \( LabG \). It can be check that this function is always called with inputs coming from \( LabG \). In other words it is only called in the first level of the recursion.

**Lemma 11.** Let \( A \) in \( LabG \) and \( B \) in \( LabH \) be trilabeled and \( l \)-prime connected graphs. If \( A \cong_f B \) for some \( f \) and \( \pi \) then Algorithm 3 generates top operations of parse trees for \( A \) and \( B \) such that \( \pi \in ISO(Q_a, Q_b) \) with \( A_a \cong_f B_b \), where \( A_a \) and \( B_b \) are the graphs described in Algorithm 3

**Proof.** Let \( A \) and \( B \) are trilabeled with \( l_1, l_2, l_3 \) and \( l'_1, l'_2, l'_3 \) respectively. If \( A \) has a trivial split (see Figure) then it has a universal vertex \( x \) of some label \( l_1 \). Then
Algorithm 2: Function Decompose

Input: A bi or trilabled l-prime connected graph $P$
Output: A parse tree of $P$ or null parse tree if $\text{cud}(P) > 3$

begin
parse-tree := a trivial parse tree with $P$ as the unique leaf
/* parse-tree may contain connected components as leaves but as the algorithm proceeds */
Leaves := $\{P\}$
while Leaves $\neq \emptyset$ do
flag := true, tree := null
Extract $\Gamma$ from Leaves
if $\Gamma$ has no more than three vertices then
Find a canonical parse tree, tree
Replace $\Gamma$ by tree in parse-tree
else
if flag is true then
flag := false, tree := null
Leaves := null
end
if flag is false then
Parse-tree := Decompose-leaf-BI($\Gamma$)
Leaves := $\{\Gamma\}$
end
if flag is true then
Leaves := $\{\Gamma\}$
end
end
end
return (parse-tree)

the algorithm removes the edges $E_{l_1l_2}, E_{l_1l_3}$ from $A$ and gives a decomposition with top operations $\eta_{l_1l_2}$ and $\eta_{l_1l_3}$ above a $\oplus$ operation whose children are $x$ and connected components $A_{a_1}, \cdots, A_{a_k}$ ($A_a = x \oplus A_{a_1} \oplus \cdots \oplus A_{a_k}$). If $A \cong f B$, then there is a universal vertex $y$ in $B$ of label $l'_1$ such that $f(x) = y$ and $\pi(l_1) = l'_1$. To decompose $B$, the algorithm removes the edges $E_{l'_1l'_2}$, $E_{l'_2l'_3}$ from $B$ to get the decomposition with top operations $\eta_{l'_1l'_2}$ and $\eta_{l'_2l'_3}$ above a $\oplus$ operation whose children are $y$ and connected components $B_{b_1}, \cdots, B_{b_k}$ ($B_b = y \oplus B_{b_1} \oplus \cdots \oplus B_{b_k}$). In fact $B_{b_1}, \cdots, B_{b_k}$ are images of $A_{a_1}, \cdots, A_{a_k}$ under $f$ in some order. The quotient graphs $Q_a$ and $Q_b$ have three vertices corresponding to top two consecutive $\eta$ operations. If $A \cong f B$ the quotient graphs are isomorphic via $\pi$ and $A_a \cong f B_b$.

If $A$ corresponds to a nontrivial split (see Figure 2) then there are two labels $l_1, l_2$ such that $E_{l_1l_2}$ is complete. We get a decomposition with $\eta_{l_1l_2}$ operation above a $\oplus$ operation whose children are connected components $A_{a_1}, \cdots, A_{a_k}$ ($A_a = A_{a_1} \oplus \cdots \oplus A_{a_k}$) of $A$ after the $E_{l_1l_2}$ edges are removed. If $A \cong f B$, then there exists a nontrivial split in $B$ and two labels $l'_1, l'_2$ such that $E_{l'_1l'_2}$ is complete and $\pi(l_1, l_2) = (l'_1, l'_2), \pi(l_3) = l'_3$. To decompose $B$, the algorithm removes the edges $E_{l'_1l'_2}$ to get the decomposition with top operations $\eta_{l'_1l'_2}$ above a $\oplus$ operation whose children are connected components $B_{b_1}, \cdots, B_{b_k}$ obtained from $B$ after $E_{l'_1l'_2}$ edges are removed. The quotient graphs $Q_a$ and $Q_b$ build from the top operations are isomorphic via $\pi$ and $A_a \cong f B_b$. □
Algorithm 3: Function \text{Decompose-leaf-TI} [6]

\textbf{Input:} A trilabeled, \(l\)-prime and connected graph \(G\)
\textbf{Output:} true with top operations of parse tree or false if \(cwd(G) > 3\)

1. begin
2. \(\text{tree} := \text{null}\)
3. if \(G\) has a universal vertex \(x\) of label \(l_1\) then
4. \(\text{Let } G_x = x \oplus \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } x, G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_12}, E_{l_13}\} \land \) \(l_1, l_2, l_3\) are labels of \(G\) */
5. \(\text{tree} = \eta_{i_1} \oplus \eta_{i_2} \oplus \eta_{i_3} (x \oplus \bigoplus_{i=1}^{k} G_{y_i})\)
6. return (true, \text{tree})
7. if \(G\) has two labels \(l_1, l_2\) such that \(E_{l_12}\) is complete then
8. \(\text{Let } G_{\eta} = \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_12}\} \land \)
9. \(\text{tree} = \eta_{i_1} \oplus \eta_{i_2} (\bigoplus_{i=1}^{k} G_{y_i})\)
10. return (true, \text{tree})
11. return (false, \text{tree}) (i.e., \(cwd(G) > 3\))

Algorithm 4: Function \text{Decompose-leaf-BI} (cf., [6])

\textbf{Input:} A bilabeled, \(l\)-prime and connected graph \(G\)
\textbf{Output:} true with top operations of parse tree or false if \(cwd(G) > 3\)

1. begin
2. \(\text{tree} := \text{null}\)
3. if \(G\) in \(PC_1\) then
4. \(\text{if } G\) has a universal vertex (say \(x\)) then
5. \(\text{Let } G_x = x \oplus \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } x, G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_1}\} \land \) \(l_1, l_2\) are labels of \(G\) */
6. \(\text{tree} = \eta_{i_2} \lor \eta_{l_1} \lor \eta_{t_i} (x \oplus \bigoplus_{i=1}^{k} G_{y_i})\)
7. return (true, \text{tree})
8. if \(G\) in \(PC_2\) then
9. Compute a set \(S\) number of vertices in \(G\) which are universal to one label class but not adjacent to other label class
10. if \(|S|\) equal to 1 (say x) then
11. \(\text{Let } G_x = x \oplus \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } x, G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_1}\} \land \)
12. \(\text{tree} = \eta_{i_2} \lor \eta_{i_1} (x \oplus \bigoplus_{i=1}^{k} G_{y_i})\)
13. return (true, \text{tree})
14. if \(|S|\) equal to 2 (say \(x_1\) and \(x_2\)) then
15. \(\text{Let } G_x = x_1 \equiv x_2 \oplus \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } x_1, x_2, G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_1}\} \land \)
16. \(\text{tree} = \eta_{i_2} \lor \eta_{i_1} (x_1 \equiv x_2 \oplus \bigoplus_{i=1}^{k} G_{y_i})\)
17. return (true, \text{tree})
18. if \(G\) in \(PC_3\) then
19. \(\text{Let } G_x = x \oplus y \oplus \bigoplus_{i=1}^{k} G_{y_i}, \text{ where } x, y, G_{\eta_i} \text{'s are connected components of} \ G \setminus \{E_{l_1}\} \land \)
20. \(\text{tree} = \eta_{i_1} \lor \eta_{i_2} \lor \eta_{i_3} (x \oplus \eta_{i_2} (y \oplus \bigoplus_{i=1}^{k} G_{y_i}))\)
21. return (true, \text{tree})
22. Compute the coconnected components of \(V_{l_1}\) and \(V_{l_2}\) and test membership of \(G\) in \(U_{l_1}, U_{l_2}, D_{l_1}\) and \(D_{l_2}\)
23. if \(G\) in \(U_{l_1}\) then return (Decompose-leaf-\(U_{l_1}(G)\))
24. if \(G\) in \(U_{l_2}\) then return (Decompose-leaf-\(U_{l_2}(G)\))
25. if \(G\) in \(D_{l_1}\) then return (Decompose-leaf-\(D_{l_1}(G)\))
26. if \(G\) in \(D_{l_2}\) then return (Decompose-leaf-\(D_{l_2}(G)\))
27. if \(G\) not in \(U_{l_1}, U_{l_2}, D_{l_1}\) and \(D_{l_2}\) then return (false, \text{tree}) (i.e., \(cwd(G) > 3\))

16
Fig. 1 Trivial split: $x$ is a universal vertex of label $l_1$ in a trilabeled graph $A$. We use the bold edge between two sets of vertices to indicate that all edges are present between two vertex sets.

Fig. 2 Nontrivial split: $V_i$ represents set of vertices in $A$ that have label $i$.

### 8.2 Decomposing Bilabeled Graphs

Our modification to the CHLRR algorithm is in Decompose-leaf-BI (Algorithm 4), where we use four labels instead of three to find structural isomorphic parse trees. If $G$ is a bilabeled, $l$-prime and connected graph of clique-width at most three, then either $G \in PC_i$ where $i \in \{1, 2, 3\}$ or $G \in \mathcal{U}_i$, $\mathcal{D}_i$ where $i \in \{1, 2\}$ (See Proposition 29 in [6]). From here on wards we assume that $G$ and $H$ are bilabeled with $l_1$, $l_2$ and $l'_1$, $l'_2$ respectively.

Lemma 12. Let $G$ and $H$ be bilabeled, $l$-prime and connected graphs. If $G \cong_f^\pi H$ for some $f$ and $\pi$ then Algorithm 4 generates top operations of parse trees $G$ and $H$ such that there is a $\pi_i \in ISOM (Q_g, Q_h)$ with $G_g \cong_f^\pi H_h$ and $\pi_i/\text{color} = \pi|_{\text{color}(Q_g)}$, where $G_g$ and $H_h$ are the graphs described in Algorithm 4.

Proof. There are three simple cases that can be handled easily. These simple cases denoted as PC1, PC2 and PC3. The other cases $\mathcal{U}_{l_1}$ ($\mathcal{U}_{l_2}$) and $\mathcal{D}_{l_1}$ ($\mathcal{D}_{l_2}$) are described in Algorithms 5 and 6 in Appendix.
PC1: If \( G \in PC1 \) then \( G \) has a universal vertex of label \( l \in \{l_1, l_2\} \) (see Figure 3). Note that in this case \( G \) can not have more than two universal vertices of same label, otherwise those universal vertices form an \( l \)-module.

To decompose \( G \) the algorithm relabels vertex \( x \) with \( l_3 \) and removes the edges \( E_{l_3 l_2} \) and \( E_{l_3 l_1} \). Then we get the decomposition with \( \rho_{l_3 \rightarrow l} \), \( \eta_{l_3, l_2} \), \( \eta_{l_3, l_1} \) above a \( \oplus \) operation with children \( x \) and connected components \( G_{g_1}, \ldots, G_{g_k} \). \( (G_g = x \oplus G_{g_1} \oplus \cdots \oplus G_{g_k}) \). If \( G \cong_f H \) then algorithm finds the unique universal vertex \( y \) in \( H \) of label \( l' \in \{l'_1, l'_2\} \) such that \( f(x) = y \) and \( \pi(l) = l' \). To decompose \( H \) the algorithm relabels the vertex \( y \) with \( l'_3 \) and removes the edges \( E_{l'_3 l'_2} \) and \( E_{l'_3 l'_1} \) to get the decomposition with \( \rho_{l'_3 \rightarrow l'} \), \( \eta_{l'_3, l'_2} \), \( \eta_{l'_3, l'_1} \) above a \( \oplus \) operation with children \( y \) and connected components \( H_{h_1}, \ldots, H_{h_k} \) (these are images of \( G_{g_1}, \ldots, G_{g_k} \) under \( f \) in some order). The quotient graphs \( Q_g \) and \( Q_h \) build from the top operations are isomorphic via \( \pi_i \), where \( \pi_i(l_3) = l'_3 \) and \( \pi_i(l) = \pi(l) \) if \( l \in \{l_1, l_2\} \). It is clear that, \( G_g \cong_f H_h, \) \( \pi_i/\text{color} = \pi |_{\text{color}(Q_g)} \).

Fig. 3 PC1: Decomposition of a bilabeled graph \( G \) with a universal vertex \( x \).

Suppose \( G \) has two universal vertices \( x_1 \) and \( x_2 \) of label \( l_1 \) and \( l_2 \) respectively. In this case we apply above procedure consecutively two times first taking \( x_1 \) as a universal vertex in graph \( G \), second taking \( x_2 \) as a universal vertex in graph \( G \setminus \{x_1\} \). Note that the order in which we consider \( x_1 \) and \( x_2 \) does not effect the structure of the parse tree.

PC2: If \( G \in PC2 \) then \( G \) can have one or two vertices of different labels which are universal to vertices of one label class but not to other label class. Let \( l_1 \) and \( l_2 \) be the labels of \( G \). In this case the algorithm finds the decomposition of \( G \) described as follows:

Case-1: Suppose \( G \) has a single vertex \( x \) of label \( l \) (see Figure 4a) that is universal to all vertices of label \( l' \in \{l_1, l_2\} \), but not adjacent to all vertices of label \( l' \in \{l_1, l_2\} \setminus l' \). To decompose \( G \), the algorithm relabels \( x \) with a label \( l_3 \notin \{l_1, l_2\} \) and removes the edges \( E_{l_3 l'} \), which gives the decomposition with
top operations $\rho_{l_3 \rightarrow l}$, $\eta_{l_3,l'}$ above a $\oplus$ operation with children $x$ and connected components $G_{g_1}, \ldots, G_{g_k}$ ($G_g = x \oplus G_{g_1} \oplus \cdots \oplus G_{g_k}$). If $G \cong_f H$, the algorithm finds a vertex $y$ in $H$ of label $m$ which is universal to all vertices of label $m'$ but not adjacent to all vertices of label $\bar{m}'$ such that $f(x) = y$ and $\pi(l) = m$. To decompose $H$ the algorithm relabels $y$ with a label $l_3' \not\in \{l_1', l_2'\}$ and removes the edges $E_{l_3m'}$, which gives the decomposition with top operations $\rho_{l_3m'}, \eta_{l_3,m'}$ above a $\oplus$ operation whose children are $y$ and the connected components $H_{h_1}, \ldots, H_{h_k}$ (these are images of $G_{g_1}, \ldots, G_{g_k}$ under $f$ in some order). The quotient graphs $Q_g$ and $Q_h$ build from top operations are isomorphic via $\pi_i$, where $\pi_i(l) = m$, $\pi(l) = \bar{m}$ and $\pi_i(l_3) = l_3'$. Moreover, $G_g \cong_f H_h$ and $\pi_i/color = \pi_{color(Q_g)}$.

**Fig. 4** Decomposing a bilabeled graph $G$, having one or two vertices of different labels which are universal to vertices of one label class but not to other label class. We use the zigzag edge to indicate the presence of some edges between the two sets of vertices.

**Case-2:** Suppose $G$ has two vertices $x_1$ and $x_2$ of label $l \in \{l_1, l_2\}$ (see Figure 4b) and $\bar{l} \in \{l_1, l_2\} \setminus l$ such that $x_1$ ($x_2$) is universal to all vertices of label $l' \in \{l_1, l_2\}$ ($\bar{l}'$), but not adjacent to all vertices of label $\bar{l}'$ ($l'$). Then the algorithm relabels vertices $x_1$ and $x_2$ with $l_3$ and $l_4$ respectively and removes edges $E_{l_3l'}$, $E_{l_4l'}$ to get the decomposition of $G$ with $\rho_{l_3l'}$, $\eta_{l_3l'}$, $\rho_{l_4l'}$, $\eta_{l_4l'}$ above a $\oplus$ operation with children $x_1$, $x_2$ and connected components $G_{g_1}, \ldots, G_{g_k}$ ($G_g = x_1 \oplus x_2 \oplus G_{g_1} \oplus \cdots \oplus G_{g_k}$). If $G \cong_f H$, the algorithm finds vertices $y_1$ and $y_2$ in $H$ of label $m \in \{l_1', l_2'\}$ and $\bar{m} \in \{l_1', l_2'\} \setminus m$ such that $y_1(y_2)$ is universal to all vertices of label $m'$ ($\bar{m}'$), but not adjacent to all the vertices of label $\bar{m}'$.
(m') and f(x_1) = y_1, f(x_2) = y_2. Then algorithm relabels vertices y_1 and y_2 with l_3' and l_4' respectively and removes edges E_{l_3',m'}, E_{l_4',m'} to get the decomposition of H with top operations μ_{l_3'→m'}, μ_{l_4'→m'}, μ_{l_3'→m'}, μ_{l_4'→m'} above a ⊕ operation whose children are y_1, y_2 and connected components H_{h_1}, ..., H_{h_k} (these are images of G_{g_1}, ..., G_{g_k} under f in some order). The quotient graphs Q_g and Q_h built from the top operations are isomorphic via π, where π(l') = π(l') = m', π_i(l_3) = l_3' and π_i(l_4) = l_4'. It is clear that, G_g ≃_f H_h and π_i/color = π_i/color(Q_g).

**PC3:** If G ∈ PC3 then G has two vertices x and y of label l, where y is universal to everything other than x, and x is universal to all vertices of label l other than y, and non-adjacent to all vertices of the other label l̸ as shown in Figure 5. To decompose G the algorithm relabels the vertices x and y with l_3 and removes the edges E_{l_3,l} to get the decomposition of G with top operations μ_{l_3→l}, μ_{l_3→l} and a ⊕ with the connected components of G_g = x ⊕ G \ {x} as children. Again the algorithm removes the edges E_{l_3,l} from G \ {x} to get the decomposition with top operations μ_{l_3→l} and a ⊕ with the connected components of G_g, = y ⊕ G \ {x,y} as children. If G ≃_f H, the algorithm finds the vertices x', y' ∈ H of label l' such that f(x) = x', f(y) = y' and π(l) = l', where y' is universal to everything other than x', and x' is universal to all vertices of label l' other than y', and non-adjacent to all vertices of label l'. Then it relabels vertices x' and y' with l_3' and removes the edges E_{l_3',l} to get the decomposition of H with top operations μ_{l_3→l'}, μ_{l_3→l'} and a ⊕ with the connected components of H_h = x' ⊕ H \ {x'} as children. Again the algorithm removes the edges E_{l_3',l} from H \ {x'} to get the decomposition with top operations μ_{l_3→l'} and a ⊕ with the connected components of H_{h_1} = y' ⊕ H \ {x', y'} as children. In this case the generated parse tree has two levels.

In first level the quotient graphs Q_g and Q_h build from top operations are isomorphic via π_{l_3} where π_{l_3}(l) = π(l) if l ∈ {l_1, l_2}, π_{l_3}(l_3) = l_3'. It is clear that, G_g ≃_f H_h and π_{l_3}/color = π_{l_3}/color(Q_g). In second level the quotient graphs Q_{g_1} and Q_{h_1} build from top operations are isomorphic via π_{l_3}, where π_{l_3}(l) = π_{l_3}(l) if l ∈ {l_1, l_2, l_3}, and G_{g_1} ≃_f H_{h_1}, π_{l_3}/color = π_{l_3}/color(Q_{g_1}). The remaining part of proof follows from Lemma 13 and 14.

**8.3 Function Decompose U_1:**

We next describe the case U_1. The case U_2 is omitted from here because it is similar to U_1. Let l_1 and l_2 be the vertex labels. The vertex set V_{l_1} consisting of vertices with label l_1 can be partitioned as follows: The set of vertices adjacent to all vertices of V_{l_2} is denoted V_{l_1}^a. The set of vertices adjacent to some of vertices of V_{l_2} is denoted V_{l_1}^b. The set of vertices adjacent to none of vertices of V_{l_2} is denoted V_{l_1}^c. Similarly we can define the sets V_{l_2}^a, V_{l_2}^b and V_{l_2}^c. For l ∈ {l_1, l_2} we say, G ∈ U_l if V_l^a ≠ ∅ and removing the edges between V_l^a and V_l disconnects G.

**Lemma 13.** Let G and H be bilabeled, l-prime and connected graphs. If G ≃_f H for some f and π then Algorithm 5 generates top operations of parse trees G
and $H$ such that there is a $\pi_i \in \text{ISO}(Q_g, Q_h)$ with $G_g \cong^\pi_i H_h$ and $\pi_i/\text{color} = \pi_{|\text{color}(Q_g)}$, where $G_g$ and $H_h$ are the graphs described in Algorithm 7.

Proof. $G \in \mathcal{U}_{l_1}$ if $V_{l_1}^a \neq \emptyset$ and removing the edges between $V_{l_1}^a$ and $V_{l_2}$ disconnects $G$. The proof is divided into two cases based on connected components (partial and non partial) of $V_1$.

A connected component of $V_1$ that contains at least one vertex of $V_{l_1}^a$ is called partial.

---

**Algorithm 5: Function Decompose-leaf-$\mathcal{U}_{l_1}$**

```
Input: A bilabeled, $l$-prime and connected graph $G$
Output: true with top operations of parse tree or false if $\text{cwd}(G) > 3$

begin
  tree := null
  if $G$ has good non partial connected component $C$ then
    Let $G_g = \oplus_{i=1}^k G_{g_i}$, where $G_{g_i}$'s are connected components of $G \setminus \{E_{l_1}, l_2\}$
    tree = $\rho_{l_3 \rightarrow l_1} \eta_{l_3, l_2} (\oplus_{i=1}^k G_{g_i})$
    return (true, tree)
  if $G$ has only partial connected components then
    Let $G_g = \oplus_{i=1}^k G_{g_i}$, where $G_{g_i}$'s are connected components of $G \setminus \{E_{l_1}, l_2\}$
    tree = $\rho_{l_3 \rightarrow l_1} \eta_{l_3, l_2} (\oplus_{i=1}^k G_{g_i})$
    return (true, tree)
  return (false, tree) (i.e., $\text{cwd}(G) > 3$)
```

---

21
If there is at least one good connected component $C$ (see Section 5.2.1 in [6]) in graph $G$ then the algorithm relabels all vertices of $V_i^n$ in good connected components with $l_3$ and removes the edges $E_{l_3}$ from $G$ to get the decomposition with top operations $\rho_{l_3} \rightarrow_{l_1}$ and $\eta_{l_3} \rightarrow_{l_2}$ above a $\oplus$ operation with the connected components $G_{g_1}, \cdots, G_{g_k}$ as children ($G_g = G_{g_1} \oplus \cdots \oplus G_{g_k}$). If $G \cong H$, up to a permutation of labels $H$ may be in $U_t$ or $U_2$, but this does not effect the structure of the decomposition as in both the cases the set of edges deleted are same. The algorithm finds at least one good connected component $C'$ in $H$ and relabels all vertices of $V_i^n$ in good connected components with $l_3'$ and removes the edges $E_{l_3'}$ from $G$ to get the decomposition with top operations $\rho_{l_3'} \rightarrow_{l_1'}$ and $\eta_{l_3'} \rightarrow_{l_2'}$ above a $\oplus$ operation with connected components $H_{h_1}, \cdots, H_{h_k}$ as children (these are images of $G_{g_1}, \cdots, G_{g_k}$ under $f$ in some order). The quotient graphs $Q_g$ and $Q_h$ build from top operations are isomorphic via $\pi_i$, where $\pi_i(l) = \pi(l)$ if $l \in \{l_1, l_2\}$, $\pi_i(l_3) = l_3'$. It is clear that, $G_g \cong H_h$ and $\pi_i/\text{color} = \pi|_{\text{color}(Q_g)}$.

If there are only partial component (see Section 5.2.1 in [6]) in graph $G$ then the algorithm relabels all the vertices $V_i^n$ with $l_3$ and removes the edges $E_{l_3}$ from $G$ to get the decomposition with top operations $\rho_{l_3} \rightarrow_{l_1}$ and $\eta_{l_3} \rightarrow_{l_2}$ above a $\oplus$ operation with the connected components $G_{g_1}, \cdots, G_{g_k}$ as children ($G_g = G_{g_1} \oplus \cdots \oplus G_{g_k}$). If $G \cong H$, the algorithm relabels all the vertices $V_i^n$ in $H$ with $l_3'$ and removes the edges $E_{l_3'}$ to get the decomposition with top operations $\rho_{l_3'} \rightarrow_{l_1'}$ and $\eta_{l_3'} \rightarrow_{l_2'}$ above a $\oplus$ operation with the connected components $H_{h_1}, \cdots, H_{h_k}$ as children (these are images of $G_{g_1}, \cdots, G_{g_k}$ under $f$ in some order). The quotient graphs $Q_g$ and $Q_h$ build from top operations are isomorphic via $\pi_i$, where $\pi_i(l) = \pi(l)$ if $l \in \{l_1, l_2\}$, $\pi_i(l_3) = l_3'$, and $G_g \cong H_h$ and $\pi_i/\text{color} = \pi|_{\text{color}(Q_g)}$.

Lemma 30, 31 in [6] shows that if $G \in U_t_3$ apart from above two ways there is no other way to continue to find the decomposition for graphs of clique-width at most three.

8.4 Function Decompose $D_t$:

Let $V_i$ be the set of vertices with label $l$. For $l \in \{l_1, l_2\}$ we say, $G \in D_t$ if $V_i$ is not connected and removing edges between the coconnected components of $V_i$ disconnects $G$.

Lemma 14. Let $G$ and $H$ be bilabeled, t-prime and connected graphs. If $G \cong H$ for some $f$ and $\pi$ then Algorithm (6) generates top operations of parse trees $G$ and $H$ such that there is a $\pi_i \in \text{ISO}(Q_g, Q_h)$ with $G_g \cong H_h$ and $\pi_i/\text{color} = \pi|_{\text{color}(Q_g)}$, where $G_g$ and $H_h$ are the graphs described in Algorithm (6).

Proof. The proof is divided into three cases depending on the structure of the graph.

---

6 A non-partial connected component $C$ of $V_i$ is good (respectively, bad), if $G$ is of clique-width at most three implies that the bilabeled graph obtained from $C$ by relabeling all the vertices of $V_i^C \cup C$ with three is of clique-width at most three (respectively of clique-width more than three).
Algorithm 6: Function Decompose-leaf-$\mathcal{T}_l$.

\begin{algorithm}
\textbf{Input:} A bilaabeled, 1-prime and connected graph $G$
\textbf{Output:} true with top operations of parse tree or false if $cud(G) > 3$
\begin{itemize}
  \item[1] begin
  \item[2] tree := null
  \item[3] if $G$ has only two coconnected components then
  \item[4] \hspace{1em} Let $G_g = \bigoplus_{i=1}^h G_{g_i}$, where $G_{g_i}$'s are connected components of $G \setminus \{E_{l_1}l_1\}$
  \item[5] \hspace{1em} tree = $\rho_{l_1-l_1}(\bigoplus_{i=1}^h G_{g_i})$
  \item[6] \hspace{1em} return (true, tree)
  \item[7] if $G$ has proper partition then
  \item[8] \hspace{1em} Let $G_g = G_{g_1} \oplus G_{g_2}$, where $G_{g_1}$ and $G_{g_2}$ are connected components of $G \setminus \{E_{l_1}l_1\}$
  \item[9] \hspace{1em} tree = $\rho_{l_1-l_1}(\eta_{l_1-l_1}(G_{g_1} \oplus G_{g_2})$
  \item[10] \hspace{1em} return (true, tree)
  \item[11] if $G$ is eligible then
  \item[12] \hspace{1em} Let $G_y = y \oplus (G_{g_1} \setminus y) \oplus_{i=2}^h G_{g_i}$, where $G_{g_i}$'s are connected components of $G \setminus \{E_{l_1}l_1\}$
  \item[13] \hspace{1em} tree = $\rho_{l_1-l_1}(\eta_{l_1-l_1}(y \oplus G_{g_1} \setminus y) \oplus_{i=2}^h G_{g_i}$)
  \item[14] \hspace{1em} return (true, tree)
  \item[15] return(false, tree) \hspace{1em} (i.e., $cud(G) > 3$)
\end{itemize}
\end{algorithm}

If there are only two coconnected components $CCC_1$ and $CCC_2$ of $V_{l_1}$, then the algorithm relabels one of $CCC_1$ or $CCC_2$ at random with $l_3$ and removes the edges $E_{l_3}l_1$ to get the decomposition with top operations $\rho_{l_3-l_1}$ and $\eta_{l_3-l_1}$ above a $\oplus$ operation with the connected components $G_{g_1}, \ldots, G_{g_h}$ as children ($G_g = G_{g_1} \oplus \cdots \oplus G_{g_h}$). If $G \cong H$, up to a permutation of labels $H$ may be in $\overline{D}_{l_1'}$ or $(\overline{D}_{l_1'})^\prime$, without loss of generality assume $H$ is in $\overline{D}_{l_1'}$. In $H$ the algorithm relabels one of the coconnected component of $V_{l_1}$ at random with $l_3'$ and removes the edges $E_{l_3}l_1$ to get the decomposition with top operations $\rho_{l_3-l_1}'$ and $\eta_{l_3-l_1}'$ above a $\oplus$ operation with the connected components $H_{h_1}, \ldots, H_{h_h}$ as children (these are images of $G_{g_1}, \ldots, G_{g_h}$ under $f$ in some order). The quotient graphs $Q_g$ and $Q_h$ build from top operations are isomorphic via $\pi_i$, where $\pi_i(l_2) = l_2'$, $\pi_i(l_1) = l_1'$ or $l_1''$, $\pi_i(l_3) = l_3'$ or $l_3''$, and $G_g \cong H; \pi_i/color = \pi_{|color(Q_g)}$.

If $G$ has a proper partition\footnote{partition of the coconnected components of $V_{l_1}$ into two sides such that the vertices of $V_{l_2}$ also partitions into two sides but no connected component of $V_{l_2}$ has vertices in both sides} then algorithm relabels one side of $V_{l_1}$ with $l_3$ and removes the edges $E_{l_3}l_1$ to get the decomposition with top operations $\rho_{l_3-l_1}$ and $\eta_{l_3-l_1}$ above a $\oplus$ operation with the connected components $G_{g_1}$ and $G_{g_2}$ as children. If $G \cong H$, the algorithm relabels one side of $V_{l_1}$ with $l_3''$ and removes the edges $E_{l_3}l_1$ to get the decomposition with top operations $\rho_{l_3-l_1}'$ and $\eta_{l_3-l_1}'$ above a $\oplus$ operation with the connected components $H_{h_1}$ and $H_{h_2}$ as children. The quotient graphs $Q_g$ and $Q_h$ build from top operations are isomorphic via $\pi_i$, where $\pi_i(l_2) = l_2'$, $\pi_i(l_1) = l_1'$ or $l_1''$, $\pi_i(l_3) = l_3'$ or $l_3''$, and $G_g \cong H; \pi_i/color = \pi_{|color(Q_g)}$.

If $G$ is eligible \hspace{1em} (see Section 5.2.2 in \cite{6} for definition) then to decompose $G$ the algorithm relabels vertices in coconnected component $CCC_d$ with $l_3$ and
removes the edges $E_{l_3l_1}$ to get the decomposition with top operations $\rho_{l_3\to l_1}$, $\eta_{l_3,l_1}$ and a $\oplus$ with the connected components $G_{g_1}, \ldots, G_{g_k}$ as children ($G_g = G_{g_1} \oplus \cdots \oplus G_{g_k}$). Again the algorithm removes the edges $E_{l_3l_2}$ from $G_{g_1}$ to get the decomposition with top operations $\rho_{l_3\to l_2}$, $\eta_{l_3,l_2}$ and a $\oplus$ with the connected components $G_{g_1}$ as children ($G_{g_1} = \bigoplus G_{g_i}$). Again the algorithm removes the edges $E_{l_3l_2}$ from $G_{g_1}$ to get the decomposition with top operations $\rho_{l_3\to l_2}$, $\eta_{l_3,l_2}$ and a $\oplus$ with the connected components $G_{g_1}, \ldots, G_{g_k}$ as children ($G_{g_1} = G_{g_1} \oplus \cdots \oplus G_{g_k}$). Again the algorithm removes the edges $E_{l_3l_2}$ from $G_{g_1}$ to get the decomposition with top operations $\rho_{l_3\to l_2}$, $\eta_{l_3,l_2}$ and a $\oplus$ with the connected components $G_{g_1}, \ldots, G_{g_k}$ as children ($G_{g_1} = G_{g_1} \oplus \cdots \oplus G_{g_k}$).

Lemma 32 in [6] shows that if $G \in \overline{\mathcal{D}}_{l_1}$, apart from the above three ways there is no other way to continue to find the decomposition for graphs of clique-width at most three. 

\[\square\]