Polynomial-time recognition of clique-width ≤ 3 graphs

Derek G. Corneil\textsuperscript{a}, Michel Habib\textsuperscript{b}, Jean-Marc Lanlignel\textsuperscript{c}, Bruce Reed\textsuperscript{d}, Udi Rotics\textsuperscript{e,∗}

\textsuperscript{a} Department of Computer Science, University of Toronto, Toronto, Ontario, Canada
\textsuperscript{b} LIAFA, UMR 7089 CNRS & Université Paris Diderot - Paris 7, UFR d’Informatique - case 7014, F-75205 Paris Cedex 13, France
\textsuperscript{c} LIRMM, UMR CNRS-Université Montpellier II, 161 rue Ada, 34392 Montpellier cedex 5, France
\textsuperscript{d} Canada Research Chair in the Combinatorics of Complex Networks, School of Computer Science, McGill University, Montreal, Quebec, Canada
\textsuperscript{e} School of Mathematics and Computer Science, Netanya Academic College, P.O. Box 120, 42100 Netanya, Israel

\textbf{A R T I C L E   I N F O}

\textbf{Article history:}
Received 30 April 2010
Received in revised form 10 February 2011
Accepted 15 March 2011
Available online 8 May 2011

\textbf{Keywords:}
Clique-width computation
Polynomial-time algorithm

\textbf{A B S T R A C T}

Clique-width is a relatively new parameterization of graphs, philosophically similar to treewidth. Clique-width is more encompassing in the sense that a graph of bounded treewidth is also of bounded clique-width (but not the converse). For graphs of bounded clique-width, given the clique-width decomposition, every optimization, enumeration or evaluation problem that can be defined by a monadic second-order logic formula using quantifiers on vertices, but not on edges, can be solved in polynomial time.

This is reminiscent of the situation for graphs of bounded treewidth, where the same statement holds even if quantifiers are also allowed on edges. Thus, graphs of bounded clique-width are a larger class than graphs of bounded treewidth, on which we can resolve fewer, but still many, optimization problems efficiently.

One of the major open questions regarding clique-width is whether graphs of clique-width at most $k$, for fixed $k$, can be recognized in polynomial time. In this paper, we present the first polynomial-time algorithm ($O(n^2m)$) to recognize graphs of clique-width at most 3.

© 2012 Elsevier B.V. with parts of the article © Bruce Reed. All rights reserved.

1. Introduction

Recently, considerable attention has been given to the notion of graph composition/decomposition schemes and the sweeping algorithmic results one can get from such a scheme. The most important of these is treewidth, introduced by Robertson and Seymour. (See [2] for an introductory overview of treewidth.) One of the major results in this area is that any problem expressible in monadic second-order logic (which includes many NP-complete graph problems), when restricted to graphs of bounded treewidth $k$, has a linear-time algorithm (albeit with a constant that grows exponentially with $k$).

Although this is a very powerful result, it is somewhat dissatisfying, since many classes of tame graphs, most notably cliques, have arbitrarily high treewidth, and yet have simple linear-time algorithms for most of the graph problems expressible in monadic second-order logic.

The clique-width of a graph $G$, denoted by $\text{cwd}(G)$, is another attempt to parameterize the composition/decomposition of a graph so that similar sweeping claims can be made about the graph's tractability for problems that are intractable in general. This notion was first introduced by Courcelle et al. in [12], and it has been studied extensively in recent years. The
clique-width of a graph G is defined as the minimum number of labels needed to construct G, using the four graph operations: creation of a new vertex v with label i (denoted v(i)), disjoint union (⊕), connecting vertices with specified labels (η), and renaming labels (ρ). More details are given in Section 2. The construction of a graph G using the above four operations is represented by an algebraic expression called a k-expression, where k is the number of labels used in the expression.

Clique-width is more powerful than treewidth in the sense that if a class of graphs is of bounded treewidth then it is also of bounded clique-width [14]. In particular, for every graph G, \( \text{cwd}(G) \leq 3 \times 2^{\text{twd}(G)-1} \), where \( \text{twd}(G) \) denotes the treewidth of G. For example, cographs (graphs with no induced \( P_4 \)s) are exactly the graphs of clique-width at most 2; trees and distance-hereditary graphs (for all vertices \( x, y \) in G, their distance in G is the same as their distance in every induced subgraph in which \( x \) and \( y \) stay in the same connected component) have clique-width at most 3 [14,24]. On the other hand, some simple families of graphs such as unit-interval and permutation graphs have unbounded clique-width [24]. In a series of papers, Brandstädt, Le, Mosca, and others have established whether various families of graphs defined by small restricted induced subgraphs are either of bounded or unbounded clique-width (see [4,7,5,6,8]).

As with treewidth, there is a general theorem indicating when a large set of NP-complete problems have polynomial-time algorithms for graphs of bounded clique-width. In particular, problems defined by monadic second-order logic formulas, using quantifiers on vertices but not on edges (denoted LinMS(\( \tau_1 \))), can be solved in linear time on any class of graphs \( E \) of clique-width at most \( k \), for some fixed \( k \), assuming that the k-expression defining the input graph is given. For details, see [13]. Note that the corresponding theorem for treewidth allows quantification on edges. In addition, polynomial-time algorithms can be obtained for other problems such as chromatic number, edge dominating set [30], ID\( _2 \)-partition problems [22], partition into independent sets/cliques, and other vertex and edge separation problems [19], on any class of graphs \( E \) of clique-width at most \( k \), for some fixed \( k \), again assuming that the k-expression defining the input graph is given. Recently, Oum [34], improving [35], presented an \( O(n^2) \)-time algorithm that for given graph G and fixed \( k \) either decides that \( \text{cwd}(G) > k \) or produces a \( k \)-expression defining G, where \( k' = 8^k - 1 \). Thus all the above problems can be solved in polynomial time (with complexity at least \( O(n^3) \)) on graphs of clique-width at most \( k \) without knowing the k-expression for the input graph.

These results raise the following two complexity questions. First, given G and integer \( k \), what is the complexity of determining whether \( \text{cwd}(G) \leq k \)? Recently, Fellows et al. showed that this problem is NP-complete [21]. Second, and more important from our perspective, what is the complexity of recognizing graphs of clique-width at most \( k \), for fixed \( k \)? As noted above, if we have a polynomial-time recognition algorithm we also need a corresponding k-expression. It is easy to see that graphs of clique-width 1 are graphs with no edges. As mentioned above, the graphs of clique-width at most 2 are precisely the cographs [14]. In this paper, we present a polynomial-time algorithm \( (O(n^m m)) \) to determine if a graph has clique-width at most \( 3 \). For graphs of clique-width \( \leq 3 \), the algorithm also constructs the 3-expression (denoted as a parse tree) which defines the graph. In other words, we prove the following theorem.

**Theorem 1.** There exists an algorithm of complexity \( O(n^2 m) \) that recognizes the graphs of clique-width \( \leq 3 \), and outputs a parse tree if clique-width \( \leq 3 \).

The question whether there exists a polynomial-time algorithm that recognizes the class of graphs of clique-width at most \( k \) remains open for any fixed \( k \geq 4 \).

The graph parameter “NLC-width”, introduced by Wanke [37], is closely related to clique-width. The NLC-width of a graph is not greater than its clique-width, and the clique-width of a graph is at most twice its NLC-width [28]. The computation of the NLC-width of a given graph is NP-complete [25]. Graphs of NLC-width 1 are cographs. Graphs of NLC-width at most 2 can be recognized in polynomial time [29,31]. The question of whether there exists a polynomial-time algorithm which recognizes the class of graphs of NLC-width at most \( k \) is still open for any fixed \( k \geq 3 \).

1.1. Overview of the paper

The second section of the paper introduces the notation and definitions used throughout the paper. In Section 2.2, an overview of the algorithm for recognizing graphs of clique-width \( \leq 3 \) is presented. One of the fundamental concepts used in our algorithm is split decomposition (first introduced in [16]), which is described in Section 3. The algorithm uses Functions LABEL and DECOMPOSE, which are presented in Sections 4 and 5, respectively. Function DECOMPOSE is quite complicated, and its correctness is proved in Section 6. Concluding remarks appear in Section 7.

2. Preliminaries

2.1. Notation and definitions

The graphs we consider in this paper are undirected and loop free. We use graph terminology that can be found in standard books (for example, see [23,38]). For a graph G, we denote by \( V(G) \), of cardinality \( n \), the set of vertices of G. Similarly, \( E(G) \), of cardinality \( m \), denotes the set of edges of G. For \( X \subseteq V(G) \), we define \( G[X] \) to be the subgraph of G induced on \( X \). Throughout the paper, \( X \) will often refer to \( G[X] \); for example, the subgraph of G induced on \( V(G) \setminus X \) is denoted by \( G \setminus X \). Similarly, \( \bar{X} \) refers to \( G[X] \), where graph \( \bar{G} \) is the complement of G, i.e., \( V(\bar{G}) = V(G) \) and \( E(\bar{G}) = \{uv \mid uv \notin E(G) \} \).
Fig. 1. The coconnected components appear inside solid ellipses and the connected subcomponents are illustrated by dashed ellipses.

and \( u \neq v \). Unless explicitly stated to the contrary, all subgraphs in this paper will be assumed to be induced subgraphs. \( \mathcal{N}_C(v) \) denotes the neighborhood of \( v \) in \( G \), i.e., the set of vertices in \( G \) adjacent to \( v \). \( \mathcal{N}_C[v] \) denotes the closed neighborhood of \( v \), i.e., \( \mathcal{N}_C[v] = \mathcal{N}_C(v) \cup \{v\} \). Similarly, \( \mathcal{N}_C[X] \) denotes the union of the closed neighborhoods of the vertices in \( X \) and \( \mathcal{N}_C(X) = \mathcal{N}_C[X] \setminus X \). For convenience, we often omit \( G \) from terms such as \( V(G) \), \( E(G) \), \( \mathcal{N}_C(v) \), etc.

We say that vertex \( v \) is universal to \( X \) if \( v \) is adjacent to all vertices in \( X \setminus \{v\} \) and that \( v \in V(G) \) is universal in \( G \) if \( v \) is universal to \( V(G) \setminus \{v\} \). We say that \( v \) sees \( X \) if \( v \) is adjacent to at least one vertex in \( X \); similarly, \( v \) sees all \( X \) if \( v \) is universal to \( X \). On the other hand, \( v \) misses \( X \) if \( v \) misses all (i.e., is not adjacent to any) vertices in \( X \). \( X \) is a clique (respectively, a stable set) in \( G \) if \( G[X] \) is a complete graph (respectively, an edgeless graph). \( X \) is a biclique in \( G \) if there is a partition of \( X \) into disjoint sets \( X_1 \) and \( X_2 \) such that all edges between vertices in \( X_1 \) and vertices in \( X_2 \) exist in \( G \). A star is a graph composed of a stable set (the rays) and a universal vertex (the centre of the star).

The coconnected components of \( G \) are the connected components of \( \overline{G} \). The connected subcomponents of \( G \) are the connected components (in \( G \)) of its coconnected components. Fig. 1 illustrates the connected subcomponents of a graph. In this and subsequent figures we use the convention of drawing a bold edge between two sets of vertices \( X \) and \( Y \) to indicate that all edges exist between vertices in \( X \) and vertices in \( Y \).

For a labeled graph \( G \), \( V_i(G) \) denotes the set of vertices in \( G \) that have label \( i \). Such subsets of vertices are also called label classes. We denote by \( E_{ij}(G) = \{e \in E(G) \mid e = ab, a \in V_i, b \in V_j\} \) the set of edges of type \( i \neq j \). For \( i \neq j \) we say that \( E_{ij} \) is complete if all these edges are present, i.e., if there are no non-edges of type \( i \neq j \). A subset \( X \) of \( V(G) \) (or the graph induced on \( X \)) is said to be unlabeled if all its elements either bear the same label, or no label; if they use exactly two different labels; if they use exactly three different labels, etc. For a labeled or unlabeled graph, we often calculate the partial degrees of the vertex \( x \), namely the number of neighbors that \( x \) has in each of the label classes.

We now give more details of the definition of clique-width presented above. The clique-width of a graph \( G \), denoted by \( cwnd(G) \), is defined as the minimum number of labels needed to construct \( G \), using the following four graph operations.

- \( \bullet v(i) \): this operation creates a new vertex \( v \) with label \( i \).
- \( \oplus \): this operation realizes the union of several disjoint graphs. Note that \( V_i \) of the new graph is the union of the \( \{V_i\} \) of the component graphs.
- \( \eta_{i,j}\): this unary operation joins the vertices labeled \( i \) to the vertices labeled \( j \); this is how edges must be created. This operation does not create multiple edges (i.e., it does not affect the pairs of adjacent vertices). Note that \( i \) must be different from \( j \); we cannot make a clique from a set of vertices with the same label.
- \( \rho_{i,j} \): under this unary operation all vertices with label \( i \) are relabeled to have label \( j \). This allows us to merge two label classes into one, thus freeing a label for later use.

Note that, once two vertices have the same label, they will be indistinguishable with respect to subsequent operations performed on them. Thus they will gain the same new neighbors. An expression built from the above four operations using \( k \) labels is called a \( k \)-expression. Each \( k \)-expression \( t \) uniquely defines a labeled graph \( graph(t) \), where the label of each vertex (in \( \{1, \ldots, k\} \)) indicates its label after all operations of the \( k \)-expression have been performed. The definition of clique-width can be naturally extended to unlabeled graphs, since unlabeled graphs and unlabeled graphs are identical objects from the clique-width perspective; moreover, since we can always merge label classes, we say for convenience that a \( k \)-expression \( t \) defines a graph \( G \) if \( G \) is equal to the graph obtained from the labeled graph \( graph(t) \) after removing its labels.

For a \( k \)-expression \( t \) defining a graph \( G \), we denote by \( tree(t) \) the parse tree constructed from \( t \) in the usual way. The leaves of this tree are the vertices of \( G \) with their initial labels, and the internal nodes correspond to the operations of \( t \), and can either have more than one child corresponding to \( \oplus \), or are unary, corresponding to \( \eta \) or \( \rho \). If \( k = cwnd(G) \), we say that \( tree(t) \) is a (clique-width) parse tree for \( G \). For each internal node \( x \) in a parse tree \( tree(t) \), the labeled graph defined by the subtree rooted at \( x \) is called the associated graph of \( x \). Fig. 2 illustrates a clique-width \( \leq 3 \) graph, its corresponding
3-expression and parse tree, and the graphs associated with some of the internal nodes of the parse tree. We let $cw(k)$ denote the set of graphs with clique-width $\leq k$.

A $k$-expression is not unique; for instance, $\eta_{1,2}$ and $\eta_{1,3}$ commute, and two consecutive $\eta_{1,2}$ operations have the same effect as one such operation. Such useless operations prevent a parse tree from having a bounded size; it is not difficult to restrict the definition of an acceptable parse tree (disallowing useless operations between any two disjoint unions) so that it has size linear in that of the graph. But still, the decomposition is not unique (consider the $P_4$ in Fig. 3 and its two different parse trees) and we do not know how the different possible parse trees are related. Not having a unique parse tree is one source of difficulty for solving the recognition problem for $k \geq 3$.

We now state a well-known lemma (see [14]) that shows that we may assume that no edges in a graph are generated more than once.

**Lemma 2** ([14]). Let $G$ be a graph of clique-width $k$. Then there is a $k$-parse tree of $G$ such that no $\eta_{i,j}$ operation recreates edges that have already been established.

We now define the notions of modules, prime graphs, and modular decomposition. A set $M$ of vertices of a graph $G$ is called a module of $G$ if every vertex outside $M$ is either universal to all vertices in $M$ or misses all vertices in $M$. Modules
The leaves of $T(G)$ are the vertices of $G$.

- For an internal node $h$ of $T(G)$, let $M(h)$ be the set of vertices of $G$ that are leaves of the subtree of $T(G)$ rooted at $h$; $M(h)$ forms a module in $G$.

- For each internal node $h$ of $T(G)$ there is a graph $G_h$ called the representative graph of $h$, with the following properties.
  - $V(G_h) = \{h_1, \ldots, h_r\}$, where $h_1, \ldots, h_r$ are the children of $h$ in $T(G)$.
  - The set of edges $E(G_h)$ is defined such that two vertices $u \in M(h_i)$ and $v \in M(h_j)$, for $1 \leq i, j \leq r, i \neq j$, are adjacent in $G$ if and only if $h_i$ and $h_j$ are adjacent in $G_h$.
  - $G_h$ is either a clique, a stable set, or a prime graph.
  - $h$ is labeled “Series” if $G_h$ is a clique, “Parallel” if $G_h$ is a stable set, and “Prime” otherwise.

The set of prime graphs $\{G_h : h \text{ is an internal node of } T(G) \text{ labeled “Prime”}\}$ is denoted as the set of prime graphs associated with the modular decomposition of $G$. For given graph $G$, the modular decomposition of $G$ can be obtained in linear time [15,33,36]. For more details of the modular decomposition of graphs, see, for example, [9,15,18,33,36].

The following lemma shows that, in order to calculate the clique-width of a graph $G$, it is enough to consider the prime graphs associated with the modular decomposition of $G$.

**Lemma 3** ([13]). For every graph $G$ which contains at least one edge, $G$ is of clique-width $\leq k$ if and only if each of the prime graphs associated with the modular decomposition of $G$ is of clique-width $\leq k$. 
In the proof of Lemma 3, it is shown how to construct a parse tree for \( G \), using the modular decomposition of \( G \), and the parse trees of the prime graphs associated with it. The idea is to scan the modular decomposition tree \( T(G) \) from bottom to top, and at each internal node \( h \) to construct a parse tree, denoted \( T_h \), using the following rule.

- Let \( h_1, \ldots, h_i \) be the children of \( h \) in \( T(G) \) and let \( A \) be the clique-width parse tree for the representative graph \( G_{h_i} \). Each leaf of \( A \) is a node \( h_i \) with an initial label \( l_i \). The parse tree \( T_{h_i} \) is obtained by replacing each leaf \( h_i \) of \( A \) with the parse tree \( T_{h_i} \) with the addition of a \( \rho_{l_i} \) operation at the top.

Now it is easy to see that, when the root \( R \) of \( T(G) \) is reached, the parse tree \( T_R \) constructed by the above process is a parse tree for \( G \). Clearly, when \( G \) is not a stable set, the clique-width of this parse tree is no greater than the clique-width of the prime graphs associated with the modular decomposition of \( G \).

Lemma 3 allows us to assume that the input graph is prime (and is therefore connected). Furthermore, we may also assume that the size of the input graph is at least 3, or else we immediately know that its clique-width is at most 3. The importance of prime graphs is captured in the following lemma.

**Lemma 4.** In a parse tree of a prime graph \( G \), if the graph associated with an internal non-root node is unilabeled, then it has only one vertex (it is a leaf).

**Proof.** The vertices of this unilabeled graph behave in the same fashion during the construction of the whole graph; they gain the same neighbors, and thus form a module of \( G \). Since \( G \) is prime, this module must be trivial. \( \square \)

Since we will concentrate on clique-width \( \leq 3 \) graphs, in the rest of the paper when we say that \( G \) has a parse tree we mean that \( G \) has a clique-width \( \leq 3 \) parse tree, i.e., a parse tree that defines \( G \) using at most three labels.

### 2.2. Overview of the algorithm

We assume that we are given connected graph \( \tilde{G} \) and we want to determine whether \( \text{cwd}(\tilde{G}) \leq 3 \). As explained above, a linear-time preprocessing step allows us to consider each non-labeled (this is equivalent to considering the graph to be unilabeled) prime graph \( G \) in turn. The algorithm proceeds in a top-down fashion. The first step in this approach is to iterate on the possible final sequence of operations that resulted in the creation of \( G \). As we shall see, this is the same as iterating on the different labelings of \( G \) where the final sequence of operations result in \( G \). For example, we will examine each vertex \( x \) in \( G \), and in each case determine if our last operation could have joined \( x \in V(G) \) with \( \mathcal{L}_G(x) \). The left parse tree in Fig. 3 is an example of this operation, where \( x \) is a leaf. In such a case, we have to determine whether it is possible to generate, with at most three labels, the bilabeling of \( H = G \setminus \{x\} \) where vertices in \( \mathcal{L}_G(x) \) have one label and the vertices not adjacent to \( x \) have a second label. Another example, representing a more difficult situation, is illustrated in the right parse tree in Fig. 3, where the final join operation is \( i_{2,3} \) adding the edge \( bc \). Fortunately, such a decomposition (known as a split decomposition) has been well studied. In such a case, the problem now is to determine whether it is possible to generate, with at most three labels, the trilabeling of \( H = G \) where the edges to be added are between vertices labeled 2 and 3, respectively, and all other vertices are labeled 1. Continuing in this fashion, the problem of determining whether \( G \) can be generated by using at most three labels is reduced to determining whether one of the cases, as represented by a set of bilabeled or trilabeled graphs \( \{H\} \), can be generated using at most three labels. If we are successful, we immediately have a clique-width \( \leq 3 \) decomposition of \( G \). We will show that, if none of the \( H \) can be generated, then \( \text{cwd}(G) > 3 \), which in turn will imply that \( \text{cwd}(\tilde{G}) > 3 \).

As a general comment, we note that determining whether a bilabeled or trilabeled graph can be generated with at most three labels is a non-trivial problem, whose solution consumes a good part of the paper.

At first glance, it seems somewhat surprising that such an algorithm has a polynomial-time bound. First we will prove that the number of cases for the last steps of \( G \)’s composition is bounded by \( O(n) \). Second, although there may be an exponential number of splits for a given graph, we will see that we only have to examine \( O(n) \) such splits. Third, given a bilabeled or trilabeled graph \( H \), we will show that we do not have to use backtracking to determine if \( H \) can be constructed using at most three labels. In fact, this algorithm can be achieved in time \( O(nm^3) \).

The overall algorithm is presented in Algorithm 1. The first step is to compute \( \mathcal{G} \), the set of prime graphs of the modular decomposition of the given graph \( \tilde{G} \). For each \( G \in \mathcal{G} \), we consider all possible last steps in building a parse tree for \( G \). As described above, this is captured by a set of bilabelings and trilabelings. Function LABEL generates LabG, this set of possible bilabelings and trilabelings of \( G \) that are to be tested. LABEL is described in Section 4. Given a bilabeling or a trilabeling \( H \in \text{LabG} \), Function DECOMPOSE determines whether \( H \) can be generated using at most three labels, and if so, generates Tree, a parse tree for \( H \). If \( H \) cannot be generated using at most three labels, then Tree is null. Function DECOMPOSE is described in Section 5. If one of the \( H \in \text{LabG} \) can be generated using at most three labels, then we modify Tree to produce a parse tree for \( G \), which is stored in \( G \).parse-tree. Finally, if any \( G \in \mathcal{G} \) cannot be generated, then we conclude that \( \text{cwd}(G) > 3 \). However, if all prime graphs in \( \mathcal{G} \) can be generated, then we use the modular decomposition of our given graph \( \tilde{G} \) as well as the parse trees for each \( G \in \mathcal{G} \) to build a parse tree for \( \tilde{G} \) that uses at most three labels.

Before presenting the descriptions of these functions, we study split decompositions in Section 3.
Algorithm 1 Clique-width ≤ 3 recognition algorithm

Input: a given connected graph \( \tilde{G} \)
Output: either a parse tree of \( \tilde{G} \) if \( \text{cwd}(\tilde{G}) \leq 3 \) or a message that \( \text{cwd}(\tilde{G}) > 3 \)

Begin

Compute the modular decomposition of \( \tilde{G} \); Let \( \mathcal{G} \) be the set of prime graphs of \( \tilde{G} \)

for all \( G \in \mathcal{G} \) do

\[ \text{Lab} G := \text{LABEL}(G); \] /* \( \text{Lab} G \) is a set of bilabelings or trilabelings of \( G \) */
\[ G.\text{parse-tree} := \text{null}; \]
while \( G.\text{parse-tree} = \text{null} \) and \( \text{Lab} G \neq \emptyset \) do

Extract \( H \) from \( \text{Lab} G \);
\[ \text{Tree} := \text{DECOMPOSE}(H); \] /* \( \text{Tree} \) is either a parse tree of \( H \) (if \( \text{cwd}(H) \leq 3 \)) or \( \text{null} \) */
if \( \text{Tree} \neq \text{null} \) then \( G.\text{parse-tree} := \rho_{i \rightarrow j} \circ \text{Tree}; \) /* for a suitable \( \rho_{i \rightarrow j} \) */
if \( G.\text{parse-tree} = \text{null} \) then return \( (\text{cwd}(\tilde{G}) > 3) \);
return (parse tree of \( \tilde{G} \))

End

COMPLEXITY ANALYSIS:

Theorem 5. Our clique-width ≤ 3 algorithm (see Algorithm 1) can be implemented to run in time \( O(n(\tilde{G})^2m(\tilde{G})) \) for given connected graph \( \tilde{G} \).

Proof. The modular decomposition can be computed in \( O(m(\tilde{G})) \) time. From Claim 10 in Section 4.3, Function \( \text{LABEL}(G) \) can be computed in time \( O(n(G)m(G)) \); there are \( O(n(G)) \) bilabelings or trilabelings of \( G \) in \( \text{Lab} G \). The \( \text{while} \) loop is executed \( O(n(G)) \) times, and, from Claim 11 in Section 5, \( \text{DECOMPOSE}(H) \) takes \( O(n(H)m(H)) \) time. Thus the \( \text{for all} \) loop has complexity \( O(\sum_{G} n(G)^2m(G)) \) that is bounded by \( O(n(G)^2m(G)) \). The building of the parse tree of \( G \), in the case that \( G \) has clique-width \( \leq 3 \), takes \( O(m(G)) \), thereby completing the proof of the theorem. □

3. Split decomposition

We now turn our attention to splits. Let \( p = (X, Y) \) be a partition of the vertices of a graph. The border of \( p \) is the set \( \tilde{p} = \tilde{X} \cup \tilde{Y} \) of vertices that have neighbors on the other side; thus \( \tilde{X} = X \cap N(Y) \) and \( \tilde{Y} = Y \cap N(X) \). Cunningham [16] defined a split of a graph to be a partition of its vertices into two sides \( X \) and \( Y \) such that all the edges are present at the border, i.e., all edges are present between \( \tilde{X} \) and \( \tilde{Y} \) (see Fig. 4; note that here, and in subsequent figures, thick edges indicate that all edges are present, whereas thin edges indicate the presence of some edges). From the point of view of the edges, a split corresponds to a disconnecting biclique. (Note the natural connection between clique-width and splits; in particular, if we are able to construct \( X \) using at most three labels such that \( X \setminus \tilde{X} \) is left with label 1 and \( X \) is left with label 2, and we can construct \( Y \) in a similar fashion where \( Y \setminus \tilde{Y} \) is left with label 1 and \( Y \) is left with label 3, then a join operation with children \( X \) and \( Y \) followed by an \( n_{2,3} \) operation will construct the original graph with at most three labels.) Since we will only decompose connected prime graphs, we restrict our attention to connected graphs. A partition with at most one vertex on one side is always a split, called a trivial split; unless stated otherwise, we restrict our attention to proper splits, ones that have at least two vertices on each side.

Our goal is to obtain a description of all the splits of a graph. To do this, we define the split decomposition, introduced by Cunningham [16], with some clarifications by Bouchet [3] (see also [32,17] for various applications and algorithms). Note that if there is no chance of confusion with other types of decompositions such as modular decompositions, we will just refer to the split decomposition as a decomposition. The basic step is to decompose a graph according to a proper split of the graph, in the following manner (see Fig. 5). First, add two special vertices, denoted by squares, adjacent respectively to \( \tilde{X} \) and \( \tilde{Y} \). Then, remove the edges of the split, between \( X \) and \( Y \). We now have a graph with two connected components (since the given graph was connected), called the components of the current decomposition. Finally, join the two special vertices by a special edge, denoted by a dashed edge in Fig. 5.
It is easy to verify that the splits of a component of a decomposition correspond to splits of the original graph. We thus recursively identify a split in one of the components, and decompose this component according to this split, until all the components are graphs without proper splits (see Fig. 6, where all the splits have been identified in the course of this process). Let us call the resulting object a “total decomposition” of the original graph, the intermediate steps being simply “decompositions”.

Some properties of a total decomposition of a graph are easy to check by induction. The special edges induce an unrooted tree structure on the components. Two original vertices are neighbors in the original graph if and only if there is an alternating path joining them, i.e., a path whose edges alternate between normal edges and special edges; the first and last edges must be normal edges and there are possibly zero special edges. For example, see the total decomposition in Fig. 6. A special edge \( e \) corresponds to a split of the original graph; \( e \) separates the tree into two subtrees, the original vertices of which form each side of the partition and the edges of the disconnecting biclique correspond to the alternating paths that contain \( e \). A total decomposition has linear size; it contains at most \( n - 3 \) special edges.

Finally, it is important to understand that a special vertex stands for some original vertices, namely, the vertices of one of the borders of that split. In particular, each special vertex is representing the vertices that can be reached from it via an alternating path. For example, in Fig. 6, consider the two special vertices joined by the special edge \( x \). One of the special vertices can reach 2, 4, and 8 by alternating paths, whereas the other special vertex can reach 3 and 7; \{2, 4, 8\} and \{3, 7\} are the boundaries of cut \( x \). The vertices of the two components of the cut are merely the sets of vertices in the two subtrees of the total decomposition after edge \( x \) is removed. In our example, the two sets are \{1, 2, 4, 5, 8, 9\} and \{3, 6, 7\}. A special vertex thus acts as an abstraction for original vertices, which can be found in a total decomposition by following the alternating paths (starting with the special vertex). We will use this fact in Section 4, since it will allow us to assign labels to the vertices of a component without having to make a distinction between original vertices and special vertices.

It is interesting to note the similarity between a total modular decomposition and a total split decomposition. A total modular decomposition builds a graph from graphs without proper modules, and the canonical modular decomposition is obtained from any total one by merging its cliques and stable sets; the canonical modular decomposition tree describes all the modules of the original graph. The total split decomposition behaves in a similar way; we will obtain the canonical decomposition by merging clique components and another special type of component. We cannot encounter stable sets, since the introduction of the special vertices prevents the components from being disconnected; instead, we will encounter stars, namely complete bipartite graphs where one side is a single vertex. Any partition of a clique or a star is a split. Following [1], a skeleton of \( G \) is defined as the decomposition of \( G \) obtained from a total decomposition of \( G \) by regrouping adjacent cliques or star components into larger clique or star components until no such operation is possible. Note that, in the terminology of Cunningham [16], the skeleton of \( G \) is called the minimal decomposition of \( G \).

We now state a fundamental theorem due to Cunningham [16].
Theorem 6 ([16]). 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 clique and star components.

Thus, in a skeleton there is no special edge between two clique components, and no special centre-ray edge between two star components. For instance, the tree in Fig. 6 is a skeleton. Note that there are at most $n - 3$ special edges, cliques, and stars with at least four vertices (there is no proper split in a clique or star of three vertices); the possibly exponential number of splits in a graph can be accounted for by a small number of cliques and stars, in a similar fashion to the number of modules in a graph.

A skeleton can be computed in linear time and space [17], but this algorithm is very difficult. For our purposes, we can use the easier $O(n^3)$ algorithm [32], since the computation of the skeleton will not be a bottleneck for the complexity of our algorithm.

Note that the distance-hereditary graphs, which are those graphs that contain only cliques and stars in their skeleton, are of clique-width $\leq 3$ [24].

4. The Function LABEL

We now examine Function LABEL, outlined in Algorithm 2. Recall that, in Algorithm 1, the input to LABEL is a prime graph $G$ which, as stated before, we may assume to be connected and of size $\geq 3$. The output of LABEL is Lab$G$, a set of bilabelings and trilabelings of $G$ such that $cwd(G) \leq 3$ if and only if there is an element $H$ in Lab$G$ such that $cwd(H) \leq 3$. Function DECOMPOSE (described in the next section) will take each $H$ (in Lab$G$) in turn and determine whether the clique-width of $H$ is $\leq 3$. As seen in Algorithm 2, the bilabelings and trilabelings that are placed in Lab$G$ are derived from considering each vertex individually, as well as from the skeleton of $G$. Note that each bilabeling is described by specifying one subset $X$ of $V(G)$; it is understood that the other set in the bilabeling is $V(G) \setminus X$.

In the first subsection, we assume that $cwd(G) \leq 3$ and examine the last few operations in the parse tree of clique-width at most 3 that produced $G$. This results in possible bilabelings $H$ of $G$. The following two subsections deal, respectively, with the situations of whether $H$ does or does not have a trilabeled connected component. Each case produces a set of bilabelings or trilabelings that are added to Lab$G$. We will also show that $cwd(G) \leq 3$ if and only if at least one of the bilabelings or trilabelings in Lab$G$ has clique-width at most 3.

Algorithm 2 Function LABEL

Input: a prime connected unlabeled graph $G$

Output: Lab$G$, a set of bilabelings or trilabelings, such that at least one of them is of clique-width $\leq 3$ if and only if $G$ is of clique-width $\leq 3$

Begin

Lab$G$ := null;
for all $v \in V(G)$ do
  Generate the bilabeling $\{v\}$ and add it to Lab$G$;
  Generate the bilabeling $\{x \in N(v) \mid N[x] \subseteq N[v]\}$ and add it to Lab$G$;
Compute the skeleton of $G$;
Search this skeleton for the special edges, the clique components and the star components;
for all special edges $s$ do Generate the trilabeling $X, \tilde{Y}, (V(G) \setminus (X \cup \tilde{Y}))$ where $(X, Y)$ is the split defined by $s$ and add it to Lab$G$;
for all clique components $C$ do Generate the bilabeling $C$ and add it to Lab$G$;
for all star components $S$ do Generate the bilabeling for $\{c\}$, where $c$ is the special centre of $S$, and add it to Lab$G$;
return (Lab$G$)
End

4.1. First steps

We assume that the connected prime graph $G$ is labeled with 1s and $cwd(G) \leq 3$. We now examine the top few operations of a parse tree that established that $cwd(G) \leq 3$.

Lemma 7. If $G$ is of clique-width $\leq 3$, there exists a parse tree of $G$ of the form $\rho_{2\to 1} \circ \rho_{3\to 1} \circ \eta_{2,3} \oplus (H)$, where $H$ is trilabeled, disconnected, and has no edges of type $2 \neq 3$. (See Fig. 7.)

Proof. Let $G$ be a prime clique-width $\leq 3$ unlabeled graph. We look at a parse tree of $G$. As $G$ is unlabeled with 1 and is connected, its root must be a $\rho_i \to 1$, for $i \in \{2, 3\}$. Up to a permutation of the labels, we can assume that it is a $\rho_{2\to 1}$. 

The preceding node cannot be an \( \eta_{1,2} \), or else \( G \) would not be prime (it would not be coconnected and thus it would give rise to a series decomposition). Thus a third label must be introduced, and this node is a \( \rho_{2 \rightarrow 1} \) or a \( \rho_{3 \rightarrow 1} \). As the next node is a \( \rho_{2 \rightarrow 1} \), these two possibilities are the same, and we have \( G = \rho_{2 \rightarrow 1} \circ \rho_{3 \rightarrow 1}(G') \), for a trilabeled \( G' \).

The preceding node must be an \( \eta_{1,2} \). Up to a permutation of the labels in the rest of the tree, we can assume it is an \( \eta_{2,3} \), i.e., \( G' = \eta_{2,3}(H) \). By Lemma 2, we may assume that there are no edges of type \( 2 \ast 3 \) in \( H \).

Finally, the preceding node cannot be an \( \eta_{1,2} \), for if it were, for instance, an \( \eta_{1,2} \), then \( V_1(H) \cup V_2(H) \) would be a module of \( G \) (of neighborhood exactly \( V_2(H) \)); thus both \( V_1(H) \cup V_3(H) \) and \( V_2(H) \) would be single vertices contradicting the fact that \( G \) (and thus \( H \)) has at least three vertices. So this node must be a \( \oplus \). \( \Box \)

At this point, it looks as though we have to look at all possible trilabelings \( G' \) of \( G \), such that \( G' = \eta_{2,3} \oplus (H) \) for a disconnected trilabeled \( H \). In fact the number of such trilabelings is exponential, and yet we will show that there is a polynomial-sized subset that covers all of them. The solution is that, for \( X \subseteq V(G) \), when we generate a bilabeling \( G_2 \), where \( V_1 = X \), \( V_2 = V(G) \setminus X \) (which is achieved by the statement “generate the bilabeling \( X' \)”), in fact we try all the trilabelings \( G_3 \) that use one label for \( X \) and the other two labels for \( V(G) \setminus X \). If some \( G_2 \) is of clique-width \( \leq 3 \), then \( G' = \rho_{2 \rightarrow 1}(G_2) \), for example, is of clique-width \( \leq 3 \), and if \( G_2 \) is of clique-width \( \leq 3 \), then the parse tree of \( G_2 \) that Function DECOMPOSE will find can be extended to a parse tree of \( G = \rho_{2 \rightarrow 1}(G_2) \). We say that the bilabeling captures the trilabelings. So, instead of having to know the three label classes of the \( G' \) we are looking for, in many cases, it is enough to know one, and to generate that set; this is why the following proofs will just aim at determining one label class. It will turn out that this technique will allow us to treat a clique or a star with just one bilabeling.

At this point, we know that \( H \) is disconnected, and we look at its connected components. In the next two subsections we deal with the cases of whether \( H \) does or does not have a trilabeled connected component.

4.2. \( H \) has a trilabeled connected component

As a first case, we look for the candidate trilabelings \( G' = \eta_{2,3} \oplus (H) \) such that \( H \) has a trilabeled connected component. This is the case that does not correspond to a split, and it turns out to be easy.

Recall that, when identifying a bilabeling, we will do so by just mentioning one of the two label classes. We generate the following sets of bilabelings:

B1: for all \( u \in V(G) \), the singleton \([u]\);  
B2: for all \( u \in V(G) \), \( \{x \in \mathcal{N}(u) \mid \mathcal{N}[x] \not\subseteq \mathcal{N}[u]\} \) (i.e., the set of vertices in \( \mathcal{N}(u) \) that have a neighbor in \( G \setminus \mathcal{N}[u] \)), if this set is not empty.

**Lemma 8.** If \( G \) admits a trilabeling \( G' \) of clique-width \( \leq 3 \) such that \( H \) has a trilabeled connected component, then it is captured by B1 or B2.

**Proof.** Let \( C \) be a trilabeled connected component of such an \( H \). Then, by considering the subtree of the parse tree of \( G \) restricted to \( C \), \( \text{cwd}(C) \leq 3 \). Since we have already undone an \( \eta_{2,3} \), by Lemma 2 we may assume there is no edge of type \( 2 \ast 3 \). So the first node of a parse tree of \( C \) is an \( \eta_{1,2} \) or an \( \eta_{1,3} \). (If the first node is a \( \rho \), then there must have been at least four labels used; if the first node is a \( \oplus \), \( C \) is disconnected.) Up to swapping labels 2 and 3, we may assume this is an \( \eta_{1,2} \).
For \( i \in \{1, 2, 3\} \), we let \( C_i \) denote \( V_i \cap C \). Now, if \( |C_2| > 1 \), then \( C_2 \) is a module, with neighborhood \( V_2 \cup C_1 \), contradicting \( G \) being module free. It is thus a single vertex, which we denote \( v \) (see Fig. 8). Note that a circle containing a number illustrates a set of vertices labeled with that number, and that dashed lines indicate that edges may or may not be present (but \( C \) must be connected). We now have two cases, depending on whether there are vertices labeled 2 outside \( C \).

If \( v \) is the only vertex of label 2, then the bilabeling \( \{v\} \) (in B1) captures this case. Now, assume there are vertices of \( V_2 \) outside \( C \); these vertices are not adjacent to \( v \), since \( v \in C \). If any vertex \( x \neq v \) satisfies \( N[x] \subseteq N[v] \), then \( x \in C_1 \). (Note that \( x \notin (V_2 \cup V_1) \setminus C \) since this implies that \( vx \notin E(G); x \notin V_1 \) since this implies that \( x \) is adjacent to vertices in \( V_2 \setminus C \)). Similarly, \( x \in C_1 \) implies that \( N[x] \subseteq N[v] \), and thus \( C_1 = \{x \mid x \neq v \text{ and } N[x] \subseteq N[v]\} \).

Now, \( V_2 = N_C(v) \setminus C_1 = \{x \in N_C(v) \mid N[x] \not\subseteq N[v]\}, \) and thus the bilabeling \( \{x \in N(v) \mid N[x] \not\subseteq N[v]\} \) in (B2) captures this case, thereby completing the proof. □

In fact, as we now see, we also capture as a side-effect some other cases (for example, the bilabeling \( \{v\} \) may be of clique-width \( \leq 3 \) even if the corresponding \( H \) has no trilabeled connected components).

4.3. \( H \) has no trilabeled connected component

We now study the other case, which is when there is a parse tree of \( G \) whose \( H \) does not have a trilabeled connected component. The idea is to consider the sets

\[
X = \{v \in H \mid v \text{ is in a connected component that contains 2s}\}
\]

and \( Y = \{v \in H \mid v \text{ is in a connected component that contains 3s}\} \).

Since in \( H \) there are no edges between the 2s and the 3s, a connected component of \( H \) that contains both 2s and 3s must also contain 1s, a contradiction to the assumption that \( H \) has no trilabeled connected components. Thus \( X \cap Y = \emptyset \). Now, every connected component of \( H \) must contain 2s or 3s, because a component consisting only of 1s would still be a connected component of \( G \) (it would not gain neighbors), and \( G \) would not be a connected prime graph. Thus \( X \cup Y = V(G) \), and we have a partition of the vertices. Since the borders of this partition are \( \tilde{X} = X \cap V_2(H) \) and \( \tilde{Y} = Y \cap V_3(H) \), we have a split.

If this split is trivial, e.g., \( X = \{x\} \), then, in \( G \), \( V_2 = \{x\} \) and \( V_1 = N(x) \), and thus \( \rho_{3-1}(G') \) (the bilabeling \( \{x\} \) in B1) is of clique-width \( \leq 3 \). As B1 was already generated during the previous case, the case of trivial splits is already taken into account, and we can restrict our attention to proper splits.

If, for every proper split, we generate the trilabeling \( V_2 = \tilde{X}, V_3 = \tilde{Y}, \) and \( V_1 = V(G) \setminus (\tilde{X} \cup \tilde{Y}) \), we will capture all the remaining \( H \)'s; but, as we have seen, the number of proper splits is not polynomial in general, because of possible cliques and stars. We have to do something else for them; namely, we will generate a bilabeling that captures all these trilabelings. Recall that the border of a split is the union of the borders on each side.

The splits of a clique \( C \) all have the same border, which is the whole clique (remember that the special vertices of this clique stand for the original vertices to which there is an alternating path); among the trilabelings that correspond to these splits, the only thing that changes is how this border is distributed between \( V_2 \) and \( V_3 \). So all these trilabelings can be captured by the bilabeling consisting of all vertices of \( G \) either in \( C \) or represented by special vertices in \( C \).

The splits of a star \( S \) have in common that one side of the border is exactly the centre \( c \) of the star (or the original vertices that it stands for if it is special). So here again we can easily capture all the corresponding trilabelings by the bilabeling consisting of all vertices of \( G \) represented by the special vertex \( c \). (If \( c \) is a vertex of \( G \), then we have already generated this singleton.)

Thus, we compute the skeleton of \( G \), and we capture all the remaining cases by generating the following bilabelings and trilabelings.

T1: for each special edge \( s \), the trilabeling \( (\tilde{X}, \tilde{Y}, V(G) \setminus (\tilde{X} \cup \tilde{Y})) \), where \( (X, Y) \) is the split defined by \( s \);

B3: for each clique component \( C \), the bilabeling determined by \( C \) (i.e., the vertices of \( G \) that are in \( C \) or represented by special vertices in \( C \));

B4: for each \( c \), the special centre of a star component, the bilabeling determined by \( c \) (i.e., the vertices of \( G \) represented by \( c \)).
Since, by Theorem 6, we have taken all splits of the graph into account, we have just shown the following lemma.

**Lemma 9.** If $G$ admits a trilabeling $G'$ of clique-width $\leq 3$ such that $H$ has no trilabeled connected components, then it is captured by B1, B3, B4, or T1.

**COMPLEXITY ANALYSIS:**

**Claim 10.** Function $\text{LABEL}$ (see Algorithm 2) can be implemented in time $O(n(G)m(G))$ for given graph $G \in \mathcal{G}$. Furthermore, for any such $G$, the length of $\text{LabG}$ is bounded by $O(n(G))$.

**Proof.** In the first for all loop, it is clear that the number of bilabelings defined above is at most $2n(G)$, and each of them can be generated in $O(m(G))$ time, which yields a complexity of $O(n(G)m(G))$ for this step.

As mentioned previously, we will use the slower ($O(n(G)^3)$), but easier skeleton-finding algorithm by McConnell and Spinrad [32], since its running time is not a bottleneck for the complexity analysis. We have seen in Section 3 that a skeleton may contain $O(n(G))$ special edges and clique and star components. For each of these objects, we can search the skeleton in $O(m(G))$ time to determine the associated labeling.

We can thus execute Function $\text{LABEL}$ in $O(n(G)m(G))$ time, and $|\text{LabG}|$ is bounded by $O(n(G))$. $\square$

5. **Function $\text{DECOMPOSE}$**

Function $\text{LABEL}$ (Algorithm 2) has returned $\text{LabG}$, a set of bilabeled and trilabeled graphs such that the given graph $\tilde{G}$ is of $\text{cwd} \leq 3$ if and only if, for each prime graph $G \in \mathcal{G}$, at least one of the labeled graphs in $\text{LabG}$ has $\text{cwd} \leq 3$. Function $\text{DECOMPOSE}$ (Algorithm 3) takes each of the bilabeled or trilabeled graphs $H$ in $\text{LabG}$ and tries to find a $\text{cwd} \leq 3$ parse tree for it.

Before describing Function $\text{DECOMPOSE}$, we raise an issue concerning the input graphs to $\text{DECOMPOSE}$ as well as the various subfunctions it employs. Following the same justification in Algorithm 1 of restricting our attention to $\mathcal{G}$, the set of prime connected subgraphs of the given graph $G$, we do the same for each of the labeled graphs that are stored in lists $\text{LabG}$ and $\text{Leaves}$ (to be introduced in Function $\text{DECOMPOSE-1leaf}$), but now we assume that each such graph $H$ is $\ell$-prime and connected. In particular, we do not want to reduce a module in a bilabeled or trilabeled graph $H$ which contains vertices bearing different labels, since different labels would indicate that these vertices could gain different new neighbors later, which would destroy the module. For example, suppose that $H$ has a module $M$ that is bilabeled $1–2$ and the parse tree of $H'$ extends the parse tree of $H$ by additional operations above the root of $H$'s parse tree. In the construction of $H$'s parse tree, we cannot factor out the bilabeled module $M$ of $H$ the way we can a unlabeled module (i.e., $\ell$-module), since, if there is an $\eta_{1,2}$ above the root of $H$'s parse tree, and no $\eta_{1,3}$, then $M$ is no longer a module of $H'$, since in $H'$ the $M$-vertices labeled 1 have a different neighbor set outside $M$ than do the $M$-vertices labeled 2. But, if a subset $M$ is an $\ell$-module, then no higher operations can destroy $M$. Since we want a module to remain a module during the construction process, we thus assume that $H$ is $\ell$-prime. We also note that, if a labeled graph is prime in the unlabeled sense, then it is also $\ell$-prime in the labeled sense, since any $\ell$-module in the labeled sense is a module in the unlabeled sense. Thus all graphs in $\text{LabG}$ are connected, bilabeled or trilabeled, and $\ell$-prime.

The $\ell$-modular decomposition tree is easy to find from the modular decomposition tree of the underlying graph, because we are mainly interested in identifying the maximal modules (which have a normal modular decomposition, since they are unlabeled), it can be obtained by a simple bottom-up sweep, checking that the modules we meet are unlabeled.

Function $\text{DECOMPOSE}$ recursively calls Function $\text{DECOMPOSE-1leaf}$ to determine whether $H$, taken from $\text{LabG}$, is of clique-width $\leq 3$; if so, such a parse tree is stored in $\text{parse-tree}$. This recursion is achieved through the list $\text{Leaves}$ that is initialized to $H \in \text{LabG}$. Function $\text{DECOMPOSE-1leaf}(\Gamma')$ will return either $\text{false}$ (indicating that $\Gamma'$ is not of clique-width $\leq 3$), or $\text{true}$. A true return has two possible meanings. The first is that $\Gamma'$ has clique-width $\leq 3$ and $\text{parse-tree}$ is augmented by replacing the leaf $\Gamma'$ with a clique-width $\leq 3$ parse tree of $\Gamma'$. (Note that this is the base case of the recursion.) The second meaning of $\text{DECOMPOSE-1leaf}(\Gamma')$ being true is that there is an appropriate decomposition that can be applied to $\Gamma'$. Such a decomposition determines the top few steps that must be present in any clique-width $\leq 3$ parse tree of $\Gamma'$. These steps replace leaf $\Gamma'$ in $\text{parse-tree}$. Since $\Gamma'$ is not a base case, the decomposition will also add new leaves to $\text{Leaves}$. These new leaves are induced subgraphs of $\Gamma'$ that are connected, bilabeled or trilabeled, and $\ell$-prime.

Function $\text{DECOMPOSE-1leaf}$, when applied to bilabeled graphs, is quite complicated, and involves a somewhat tedious case by case development. To help the reader understand that part of the algorithm, we have postponed the details of the proof of correctness until Section 6. The rest of this section explains how Function $\text{DECOMPOSE-1leaf}$ can begin the decomposition of a bilabeled or trilabeled $\ell$-prime connected graph $\Gamma'$, i.e., disconnect it without falling out of the class of graphs of clique-width $\leq 3$. Recursive deployment of $\text{DECOMPOSE-1leaf}$ allows us to determine whether a given bilabeled or trilabeled graph $H \in \text{LabG}$ has a $\text{cwd} \leq 3$ parse tree.

We now deal with the two cases of whether $\Gamma'$ is trilabeled or bilabeled.
that other cases, we are correct in concluding that there is no way to continue a clique-width \( \leq \eta \) is handled as a base case. Thus we have shown the restricted conditions where an a therest of the graph (i.e., we have a trivial split). (See Fig. 10, where \( a, b \in \{1, 2\} \)).

If, however, \( \Gamma' \) is connected, the first node of a parse tree of \( \Gamma' \) cannot be a \( \oplus \) (disconnected subgraphs are needed), or a \( \rho_{i \rightarrow j} \) (a fourth label is needed). Could there be an \( \eta_{i,j} \) where \( \{a, b\} \neq \{i, j\} \)? If so, we may assume without loss of generality that \( i = a \) and \( j = c \). But now, \( \Gamma' \) induced on \( V_a \) is an \( \ell \)-module of \( \Gamma' \) unless \( V_a \) consists of a single vertex \( x \) that is universal to the rest of the graph (i.e., we have a trivial split). (See Fig. 10, where \( a = 1 \)). Finally, we could have all three pairs of complete edge sets. Now, to avoid \( \ell \)-modules in \( \Gamma' \), all three label sets must be a single vertex, and thus \( \Gamma' \) is a trilabeled triangle that is handled as a base case. Thus we have shown the restricted conditions where an \( \eta_{i,j} \) could exist with \( \{a, b\} \neq \{i, j\} \); in the other cases, we are correct in concluding that there is no way to continue a clique-width \( \leq 3 \) decomposition of \( \Gamma' \).

There is, however, one other point we must address. In Fig. 9, we have assumed that we can add all the \( ab \) edges in \( \Gamma \) to the disconnected components that make up \( \Gamma' \). In particular, it is safe for us to form \( \Gamma' \) by removing all the \( ab \) edges in \( \Gamma \) (i.e., is it possible that \( \Gamma' \) has clique-width \( \leq 3 \) but \( \Gamma' \) does not)? The following lemma, which is true for any clique-width, answers this question.

**Lemma 12.** Let \( H \) be a trilabeled graph of clique-width \( \leq 3 \), such that \( E_{12} \) is complete. Let \( H' = H \setminus E_{12} \). Then \( H' \) is of clique-width \( \leq 3 \).

**Proof.** Let \( t \) be a parse tree of \( H \). By **Lemma 2**, we may assume that no \( \eta \) operation recreates edges that are already present. In \( t \), suppose there are some \( \eta \) operations that create edges that are not in \( H' \). Let \( v \) be such a node in \( t \) where \( v \) is an \( \eta_{ab} \) operation that creates edge \( xy \) that is in \( H \), but not in \( H' \). Without loss of generality, assume that, immediately before the execution of this \( \eta \) operation, the label of \( x \) is \( a \) and the label of \( y \) is \( b \), and that, in the path from \( v \) to the root of \( t \), there is a
Fig. 9. Decomposing a split in a trilabeled $\Gamma$, where $\Gamma'_1 \cdots \Gamma'_k$ are the connected components (of $\Gamma$ with the $E_{12}$ edges removed) that contain vertices of $V_1$ and $\Gamma'_{k+1} \cdots \Gamma'_j$ are the components that contain vertices of $V_2$.

Fig. 10. Decomposing a universal vertex in a trilabeled $\Gamma$, where $\Gamma'_1 \cdots \Gamma'_k$ are the connected components of $V_2 \cup V_3$ once the $E_{12} \cup E_{13}$ edges have been removed from $\Gamma$.

sequence of $\rho$ operations that collectively map $a$ onto 1 and $b$ onto 2. Thus all edges created at $v$ must be in $E_{12}$, and hence in $H \setminus H'$. Let $t'$ be the parse tree obtained from $t$ by removing all such nodes $v$. Since no new labels have been used, $t'$ creates $H'$, using at most three labels. □

Thus Function Decompose-leaf correctly handles the trilabeled case.

Algorithm 4 Function Decompose-leaf

**Input:** A bi- or trilabeled, $\ell$-prime, connected graph $\Gamma$

**Output:** true if a decomposition of $\Gamma$ can be started, or false if it was detected that $\Gamma$ is not of clique-width $\leq 3$

**Begin**

if $\Gamma$ has no more than 3 vertices then
  Decompose $\Gamma$;
  return (true);
else Compute the sizes of $V_1$, $V_2$, and $V_3$;
Compute the partial degrees of the vertices;
if $\Gamma$ is trilabeled then
  if $\Gamma$ has a label that consists of a single universal vertex $x$ then proceed with the decomposition shown in Fig. 10 with $x$ of label 1 and return (true);
  if there is $a \neq b \in \{1, 2, 3\}$ such that $E_{ab}$ is complete and $\Gamma' = \Gamma \setminus E_{ab}$ is disconnected then proceed with the decomposition shown in Fig. 9 with $(a, b) = \{1, 2\}$ and return (true);
  return (false)
else return (Decompose-leaf-BI($\Gamma$))

**End**
COMPLEXITY ANALYSIS:

**Claim 13.** Function $\text{Decompose-leaf}$ (see Algorithm 4) can be implemented to run in time $O(m(\Gamma'))$.

**Proof.** Note that we can sweep the graph and compute the partial degrees of the vertices in terms of the labels of their neighbors, and then check in $O(n(\Gamma'))$ whether a given $E_{ij}$ is complete. From Claim 15 in the next subsection, Function $\text{Decompose-leaf-BI}$ can be implemented in time $O(m(\Gamma'))$. $\square$

5.2. Decomposing a bilabeled $\Gamma'$

When $\Gamma'$ is bilabeled, the decomposition can be more difficult. We assume that $\Gamma'$ is labeled with 1s and 2s, and will use $\ell$ as an arbitrary element of $\{1, 2\}; \bar{\ell} = \{1, 2\} \setminus \ell$. We partition $V_1$ as follows (see Fig. 11).

- $V_1^a$: the vertices that see all of $V_2$;
- $V_1^s$: the vertices that see some $V_2$s, but not all;
- $V_1^n$: the vertices that see none of $V_2$.

We define symmetrically the sets $V_2^a, V_2^s, V_2^n$.

Although the general case is difficult, there are three simple cases that can be handled immediately (see Figs. 12 and 13(a); in each case, we extend the parse tree as shown in the figures).

PC1: $\Gamma'$ has a universal vertex $x$ of label $\ell$ (see Fig. 12(a));
PC2: $\Gamma'$ has a vertex $x$ of label $\ell$ that is universal to all vertices of label $\ell' \in \{1, 2\}$ but is non-adjacent to all vertices of label $\bar{\ell'}$ (see Fig. 12(b));
PC3: $\Gamma'$ has two vertices $x$ and $y$ of label $\ell$, where $y$ is universal to everything other than $x$, and $x$ is universal to all vertices of label $\ell$ other than $y$, and is non-adjacent of all vertices of label $\bar{\ell}$ (see Fig. 13(a)).

The procedures for the particular cases are correct, since, as shown in Figs. 12 and 13(a), in these cases we know how to build $\Gamma'$ from a subgraph $\Gamma''$, which must be of clique-width $\leq 3$ if $\Gamma$ is. It may seem strange that we should not introduce the symmetric case of PC3, with $x$ universal to all vertices of label $\bar{\ell}$. The reason is that, in this case, PC2 applies with $x$ as the special vertex yielding graph $\Gamma_1$, to which PC1 applies, since $y$ is a universal vertex of $\Gamma_1$ (see Fig. 13(b)).

If we do not encounter one of the simple cases, we have to enter the complex part of the algorithm.

![Fig. 11. The sets $V_1^a, V_1^s,$ and $V_1^n$, denoted 1a, 1s, and 1n, respectively.](image)

![Fig. 12. Cases PC1 and PC2, where $\Gamma''$ is $\Gamma \setminus x$.](image)
Let \( \mathcal{E} \) be the set of graphs that are of clique-width \( \leq 3 \), bilabeled 1–2, connected, \( \ell \)-prime, and not simple (i.e., with more than three vertices, and not one of the particular cases PC1, PC2, or PC3); see the examples in Fig. 14, and more generally in Fig. 15. Note that in this and subsequent figures a crossed bold line indicates the biclique edges to be removed. Thus either \( \Gamma \in \mathcal{E} \), or \( G \) is not of clique-width \( \leq 3 \). We define the following subsets of \( \mathcal{E} \) for \( \ell \in \{1, 2\} \).

- \( \mathcal{U}_\ell \): \( V^a_\ell \neq \emptyset \), and removing the edges between \( V^a_\ell \) and \( V^c_\ell \) disconnects \( \Gamma \) (for an example of a \( \mathcal{U}_1 \) graph, see Fig. 14(a), where \( V^a_1 = \{b\} \), and the removal of edges \( bc \) and \( bd \) disconnects the graph);
- \( \overline{\mathcal{D}}_\ell \): \( V^c_\ell \) is not connected, and removing edges between the coconnected components of \( \overline{V}^c_\ell \) disconnects \( \Gamma \) (for an example of a \( \overline{\mathcal{D}}_1 \) graph, see Fig. 14(b), where \( V_1 = \{b, c, d\} \); the coconnected components of \( V_1 \) are \( \{b\} \), \( \{c\} \) and \( \{d\} \); and the removal of any pair of edges in \( \{bc, bd, cd\} \) disconnects the graph).

The notation \( \mathcal{U}_\ell \) recalls that there are universal vertices (with respect to \( V^c_\ell \)) in \( V^a_\ell \) and \( \overline{\mathcal{D}}_\ell \) recalls that the complement of \( V^c_\ell \) is disconnected. It is important that the graph should become disconnected if the relevant edges are removed; some kind of decomposition of \( \Gamma \) can then be considered. Indeed, to each of these two sets can be associated a way of introducing the third label and undoing some \( \eta_{3,i} \) operation. For instance, if \( \Gamma' \in \mathcal{U}_1 \), we can relabel with 3 (some of) the \( V^a_1 \) vertices, and undo an \( \eta_{3,2} \), as Fig. 14(a) suggests (in particular, we would create the subgraph on \( \{a, b\} \), where vertex \( b \) has label 3, as well as the subgraph on \( \{c, d, e\} \) with their present labels and then perform an \( \eta_{3,2} \) operation); and it is possible to disconnect \( \Gamma' \) in this way. If \( \Gamma' \in \overline{\mathcal{D}}_1 \), we can relabel with 3 a set of coconnected components of \( V_1 \) (for example, \( \{b\} \) in the
Thedefinitionsofthesetwosetsarisequitenaturallyfromtheconsiderationofthebasicwaysofbuildingagraph.
The following proposition (proved in Section 6.1) is the key to the DECOMPOSE function; it shows that no possibility has
been forgotten, and, most importantly, that there is no choice.

**Proposition 14.** \( V = U_1 \cup U_2 \cup D_1 \cup D_2 \), and this union is disjoint.

This is the reason why the PCs were dealt with beforehand; they would correspond to two overlapping sets. We thus have
a reciprocal to the preceding constructions; once we know to which set a \( G \) belongs, we know what kind of decomposition
must be used. We do not know yet exactly what vertices are to be relabeled with 3, but we have shown that we do not need
to follow a trial-and-error approach.

Thus (see Algorithm 5) we test the membership of \( G \) in these sets. \( G \) is not of clique-width \( \leq 3 \) if it does not match
exactly one of the definitions. If it does, we are left with two cases, up to symmetry: \( G \in U_i \) or \( G \in D_i \), which are handled
in the next two subsubsections, respectively.
Algorithm 5 Function Decompose-leaf-BI

Input: A graph \( \Gamma \), bilabeled, \( \ell \)-prime and connected
Output: true if a decomposition of \( \Gamma \) can be started, or false if it was detected that \( \Gamma \) is not of clique-width \( \leq 3 \)

Begin
  for all vertex \( v \)
    if \( v \) is in PC1 then apply the decomposition shown in Fig. 12(a) and return(true);
    if \( v \) is in PC2 then apply the decomposition shown in Fig. 12(b) and return(true);
    if \( v \) is in PC3 then apply the decomposition shown in Fig. 13(a) and return(true);

  Compute the coconnected components of \( V_1 \) and \( V_2 \);
  Test membership of \( \Gamma \) in \( U_1 \), \( U_2 \), \( D_1 \) and \( D_2 \);
  if \( \Gamma \) does not belong to exactly one of these sets then return(false);
  if \( \Gamma \in U_1 \) then return(Decompose-\( U_1 \)(\( \Gamma \)));
  if \( \Gamma \in U_2 \) then return(Decompose-\( U_2 \)(\( \Gamma \)));
  if \( \Gamma \in D_1 \) then return(Decompose-\( D_1 \)(\( \Gamma \)));
  if \( \Gamma \in D_2 \) then return(Decompose-\( D_2 \)(\( \Gamma \)));

End

COMPLEXITY ANALYSIS:

Claim 15. Function Decompose-leaf-BI (see Algorithm 5) can be implemented to run in time \( O(m(\Gamma)) \).

Proof. First, we must be able to detect a PCI in linear time, which is easy. Knowing the sizes of \( V_1 \) and \( V_2 \), and, for each vertex, its partial degrees, and its non-neighbor, if it has only one, we can then check in constant time if a given vertex is a particular vertex of a PCI.

When we used the partial degrees of the vertices for detecting the PCi’s, we had in fact identified \( V_1^D \) and \( V_2^D \); we can now compute the connected components of \( \Gamma \), ignoring the edges between \( V_1^D \) and \( V_2^D \). If there are several such components, we decide that \( \Gamma \in U_1 \). The same principle can be used to test whether \( \Gamma \in U_2 \). We can also use this principle for the other two tests, except that we must be able to compute the coconnected components of a graph \( (V_1 \cup V_2) \) in linear time. One way to do this is to employ the partition refinement technique described in [27]. In particular, by placing non-adjacent vertices before adjacent vertices in a search of the graph, one can easily compute the coconnected components in linear time.

Now, using the results of Claims 18 and 22 (in Sections 5.2.1 and 5.2.2, respectively), which show that Functions Decompose-\( U_1 \) and Decompose-\( D_1 \), respectively, can be implemented in \( O(m(\Gamma)) \), we complete the proof of the claim. □

5.2.1. Case \( \Gamma \in U_1 \) (and \( \Gamma \in U_2 \))

We now deal with the first difficult case, namely, when \( \Gamma \in U_1 \). Referring to Fig. 11, recall that \( \Gamma \in U_1 \) if \( V_1^D \neq \emptyset \) and removing edges between \( V_1^D \) and \( V_2 \) disconnects \( \Gamma \). We have to decide which vertices in \( V_1^D \) will be relabeled; as shown in Fig. 16, it does not always work to relabel them all. This graph is of clique-width \( \leq 3 \), as shown by the parse tree, and the only \( V_1^D \) vertex that is relabeled is \( D \). The other \( V_1^D \) vertices, namely \{\( b, y \)\}, retain their initial 1 label. Now, suppose that we had relabeled all vertices in \( V_1^D \). In this case, we would have to show that the bilabeled square on the vertices \{\( b, x, c, y \)\} is of clique-width \( \leq 3 \) (see Fig. 17); note that it is easy to show that this is impossible. Thus, when relabeling vertices in \( V_1^D \), we were forced to ignore the \( V_1^D \) vertices \( b \) and \( y \). Now, the question is: how do we choose the \( V_1^D \) vertices to relabel?

We first divide the connected components of \( V_1 \) into three categories. A connected component of \( V_1 \) that contains at least one vertex of \( V_1^D \) is called partial. (Note that, when we used the partial degrees of the vertices to detect the PCi’s, we could also determine which connected components of \( V_1 \) are partial.) For the non-partial connected components, each vertex either sees all of \( V_2 \), or misses it. We say that a non-partial connected component \( C \) of \( V_1 \) is good (respectively, bad), if \( \Gamma \) is of clique-width \( \leq 3 \) implies that the bilabeled graph obtained from \( C \) by relabeling all the vertices of \( V_1^D \cap C \) with 3 is of clique-width \( \leq 3 \) (respectively, clique-width \( \geq 3 \)). Note that, in the above definition, we assume that \( \Gamma \) is of clique-width \( \leq 3 \), since, if \( \Gamma \) is not of clique-width \( \leq 3 \), then, no matter how we decompose \( \Gamma \), the decomposition will eventually fail.

To help us decide whether \( \Gamma \) is of clique-width \( > 3 \) or what decomposition should be applied, we examine each non-partial connected component \( C \) of \( V_1 \) and determine whether it is good (as defined above), indicating that by relabeling all its \( V_1^D \) vertices it is possible for \( \Gamma \) to be of clique-width \( \leq 3 \) or bad (as defined above), indicating that such a relabeling would not allow \( \Gamma \) to be of clique-width \( \leq 3 \).

Loosely speaking, partial connected components tend to create trilabeled subproblems; good non-partial connected components allow the easy decomposition of Fig. 18 (where \( D \) is the result of relabeling the \( V_1^D \) vertices with 3); bad non-partial connected components can only occur in graphs of clique-width \( \leq 3 \) in very special circumstances, such as illustrated in Fig. 16. Note that, in this example, the bilabeled graph induced on \{\( a, b, x, c, y \)\} is added to Leaves. When Function DECOMPOSE examines this graph, it calls Decompose-leaf, which in turn calls Decompose-leaf-BI; at this
Fig. 16. Example showing that not all $V^a_1$ vertices can be relabeled.

Fig. 17. The square.

Fig. 18. Separating a good connected component $C$, where $\Gamma'$ is the set of components of $\Gamma \setminus C$ (with the $(C \cap V^a_1) \ast V_2$ edges removed).

point, the graph is recognized to be a PC3. The problem, of course, is how to decide whether a given non-partial connected component is good or bad. This is done by Function Test-component, which appears in Algorithm 6.
In the proof of correctness of this algorithm, we will show that, if \( \Gamma \) is of clique-width \( \leq 3 \), then a connected component of \( V_1 \) is good if and only if it does not contain the square (see Fig. 17) after \( V_1^u \) is relabeled.

### Algorithm 6 Function Test-component

**Input:** \( C \), a non-partial connected component of \( V_1 \) of \( \Gamma \), a labeled, connected, \( \ell \)-prime and not simple graph in \( \mathcal{U}_1 \)

**Output:** whether \( C \) is good or bad

**Begin**

- Compute \( T_C \), the modular decomposition tree of \( C \);
- \( C' := \) the graph obtained from \( C \) by recursively removing vertices that are universal in their connected component;
- \( C \).type := \text{good};
- \( \text{for each connected component } c \text{ of } C' \text{ do} \)
  - if \( c \) contains at least two vertices of \( V_a^u \) and at least two vertices of \( V_n^u \) and two vertices that are the only non-neighbors of each other, one in \( V_a^u \), one in \( V_n^u \) then \( C \).type := bad; /* See Fig. 19. */
- return (\( C \).type);

**End**

### COMPLEXITY ANALYSIS:

**Claim 16.** Function Test-component (see Algorithm 6) can be implemented to run in time \( O(m(C)) \) for given non-partial connected component \( C \).

**Proof.** As previously mentioned, the modular decomposition tree of \( C \) can be calculated in time \( O(m(C)) \). In computing \( C' \) from \( C \), the difficulty is that removing a universal vertex might break its connected component into several connected subcomponents. The correctness of the following linear-time implementation should be clear.

We search \( C' \)'s modular decomposition tree from the root, in such a way that we will take exactly the vertices of \( C' \). For each possible node in the tree, we take the following action.

**Series node:** We ignore each child that is only one vertex (which would be a universal one). If only one child remains, we continue the search on this child. Otherwise (i.e., the graph is connected without a universal vertex), we take every descendant vertex of the remaining children.

**Parallel node:** We continue the search on each child separately (this amounts to separating the connected components).

**Prime node:** We take all its descendant vertices. (Note that a prime node represents a connected graph without universal vertices.)

Now that we know \( C' \), we must check, for every connected component of \( C' \) that has at least two vertices of each label (where we assume that the vertices of \( V_a^u \) have been assigned label 3), whether it is of the shape illustrated in Fig. 19. This can be done in linear time with the technique of partial degrees that we have already used. \( \square \)

**Lemma 17.** If \( \Gamma \) is of clique-width \( \leq 3 \), Function Test-component correctly decides whether a non-partial connected component is good or bad.

This lemma is proved in Section 6.2.

Having seen how to test the non-partial connected components of \( V_1 \), we use the algorithm in Function Decompose-\( \mathcal{U}_1 \) to determine whether it is possible for \( \Gamma' \) to be of clique-width \( \leq 3 \), and, if so, which vertices are to be relabeled. The first three if statements in Decompose-\( \mathcal{U}_1 \) identify cases where \( \Gamma' \) is not of clique-width \( \leq 3 \). If these conditions are not met, it is possible that there is one bad component, in which case there are no partial connected components and there is at least one good connected component. In this case, we relabel with 3 the \( V_1^u \) vertices of the good connected component(s) and proceed with the decomposition shown in Fig. 18.

Otherwise (i.e., all the connected components are partial), we relabel with 3 all the vertices of \( V_1^u \), and undo an \( \eta_{3,2} \), which disconnects the graph by the definition of \( \mathcal{U}_1 \) (see Fig. 20).
Algorithm 7 Function Decompose-$\mathcal{U}_1$

Input: a graph $\Gamma \in \mathcal{U}_1$
Output: true if the decomposition of $\Gamma$ could be started, or false if we have detected that $\Gamma$ is not of clique-width $\leq 3$

Begin
Compute the connected components of $V_1$;
Using the partial degrees, identify the partial and non-partial components;
for each non-partial connected component $C$ do
  type($C$):= Test-component($C$)
if there are at least two bad components then return (false);
if there is at least one partial, and one bad component then return (false);
if there is only one component and this component is bad then return (false);
if there are good components then separate the good components with the decomposition shown in Fig. 18 and return (true);
/* Or else there are only partial components */
Relabel with 3 all the vertices of $V_1^1$, proceed with the decomposition shown in Fig. 20 and return (true);
End

COMPLEXITY ANALYSIS:

Claim 18. Function Decompose-$\mathcal{U}_1$ (see Algorithm 7) can be implemented to run in time $O(m(\Gamma))$.

Proof. By Claim 16, Function Test-component can be implemented in time $O(m(C))$ for connected component $C$. Thus the for each loop can be implemented in time $O(m(\Gamma))$. The rest of the linear-time implementation is straightforward. $\square$

The following lemma, which is proved in Section 6.2, establishes the correctness of Function Decompose-$\mathcal{U}_1$.

Lemma 19. Function Decompose-$\mathcal{U}_1$ presented in Algorithm 7 is correct.

This finishes the case $\Gamma \in \mathcal{U}_1$; obvious changes to Algorithms 6 and 7 yield Function Decompose-$\mathcal{U}_2$.

5.2.2. Case $\Gamma \in \overline{\mathcal{D}}_1$ (and $\Gamma \in \overline{\mathcal{D}}_2$)

Our final case is when $\Gamma \in \overline{\mathcal{D}}_1$. Recall that $\Gamma \in \overline{\mathcal{D}}_1$ if $V_1^1$ is not connected, and removing the edges between the coconnected components of $V_1^1$ disconnects $\Gamma$. We must decide how we will partition the coconnected components of $V_1^1$ into two parts $A$ and $B$, one of which will be relabeled with 3. We are about to remove all the edges between some $A$ and some $B$. If there is no way to disconnect the graph in this way, then the graph cannot be of clique-width $\leq 3$.

First, there are two simple cases.

• If there are only two coconnected components (as in Fig. 21, where the circles containing a 1 represent the connected subcomponents of $V_1^1$, and the larger circles represent the former coconnected components), there is no real choice; we relabel one of them at random. We know, by the definition of $\overline{\mathcal{D}}_1$, that undoing an $\eta_{1,1}$ will then disconnect the graph.

• Suppose that it is possible to partition the coconnected components of $V_1^1$ into two sides $A$ and $B$ such that the vertices of $V_2^1$ also partition into the $A$-side and the $B$-side but no connected component of $V_2^1$ has vertices both in $A$ and in $B$, thereby providing a bridge between the two sides (see Fig. 22; also see a counterexample in Fig. 23). If we have such a partition, called a proper partition, then undoing the $\eta_{1,3}$ disconnects the graph, leaving only bilabeled components. These components are subgraphs of $\Gamma$, so it is correct to proceed with the decomposition shown in Fig. 22, if such a proper partition can be found.

Thus, if there are more than two coconnected components, we first check whether there exists a proper partition.

If there is no proper partition, it is still possible that $\Gamma$ is of clique-width $\leq 3$; but then it must have a very special structure.

For $\Gamma \in \overline{\mathcal{D}}_1$, with at least three coconnected components and no proper partition, we let $\Gamma_d$ be the disconnected graph obtained by removing from $\Gamma$ the edges between the coconnected components of $V_1^1$ (see Fig. 23, where the circles containing a 1 represent the connected subcomponents of $V_1^1$, and the larger circles and ellipses represent the former coconnected components). Such a $\Gamma$ is said to be eligible if $\Gamma_d$ satisfies the following properties (see Fig. 23).

• There exists a unique connected component (called $c$) of $\Gamma_d$ that has vertices in several coconnected components of $V_1^1$.
• $d = V_2^1 \setminus c$ has neighbors in exactly one coconnected component (called $ccc_d$) of $V_1^1$.
• $c \cap ccc_d$ is a connected subcomponent ($y$), and $y$ is universal to $V_2^1 \cap c$.
• $ccc_d \setminus \{y\} \neq \emptyset$.
• All coconnected components of $V_1^1$ intersect $c$. 
Fig. 20. All the components are partial, where $\Gamma_1 \ldots \Gamma_k$ are the components of $\Gamma'$ when edges between $V_1^d$ and $V_2$ are removed.

Fig. 21. Example of a graph with two coconnected components CCC1 and CCC2, where $\Gamma_1 \ldots \Gamma_k$ are the connected components of $\Gamma$ after the edges between CCC1 and CCC2 are removed and $\Gamma_i^*=\Gamma_i$ after the relabeling of the vertices of CCC2 with 3.

Fig. 22. A graph $\Gamma' \in D_1$ that can be properly partitioned, where $\Gamma_1$ (respectively, $\Gamma_2$) is the set of connected components of the $A$-side (respectively, $B$-side) after the $A*B$ edges have been removed.

If $\Gamma'$ is eligible, then we relabel ccc_d with 3 and we let $\Gamma_1$, $\Gamma_2$, $\ldots$, $\Gamma_k$ be the connected components of $\Gamma' \setminus E_{1,3}$, where $\Gamma_1$ contains c (see Fig. 24), and perform the decomposition shown in the figure.

The next lemma (which is proved in Section 6.3) sums up these claims on the structure of $\Gamma'$.
Fig. 23. A graph $\Gamma \in \mathcal{D}_1$ that cannot be properly partitioned, yet is eligible: structure of $\Gamma_d$.

Fig. 24. An eligible $\Gamma \in \mathcal{D}_1$ and its decomposition, where $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ are the coconnected components of $\Gamma$ (with the edges of $\mathrm{CCC}_d \ast (V_1 \setminus \mathrm{CCC}_d)$ removed), and $\Gamma_1$ contains $c$.

Lemma 20. If $\Gamma \in \mathcal{D}_1$, with at least three coconnected components and no proper partition, is of clique-width $\leq 3$, then $\Gamma$ is eligible, and the decomposition shown in Fig. 24 is correct.

This finishes the case $\Gamma \in \mathcal{D}_1$, and the whole algorithm; Function Decompose-$\mathcal{D}_1$ is summed up in Algorithm 8. The correctness of Function Decompose-$\mathcal{D}_1$ is established in the following lemma, which follows immediately from Lemma 20.

Lemma 21. Function Decompose-$\mathcal{D}_1$ (see Algorithm 8) is correct.

**Complexity Analysis:**

Claim 22. Function Decompose-$\mathcal{D}_1$ (see Algorithm 8) can be implemented to run in time $O(m(\Gamma'))$.

Proof. To see that there exists a simple linear algorithm that finds a proper partition (if one exists), first we note that we have already computed the coconnected components of $V_1$ in linear time when we checked whether $\Gamma \in \mathcal{D}_1$. Now we can make minor modifications to an algorithm that computes the connected components of a graph, a breadth-first search, for instance.

- We ignore the edges between the coconnected components of $V_1$.
- Each connected component of $V_2$ may cause various coconnected components to be merged into master components (i.e., coconnected components that must be on the same side of a proper partition, if one exists).
Algorithm 8 Function Decompose-$D_1$

Input: a graph $\Gamma \in D_1$
Output: true if the decomposition of $\Gamma$ could be started, or false if we have detected that $\Gamma$ is not of clique-width $\leq 3$

Begin
if $\Gamma$ has only two coconnected components then
Relabel a random one with 3;
Proceed with the decomposition shown in Fig. 21 and
return(true);

Determine whether $\Gamma$ has a proper partition;
if so then
Relabel a random side with 3;
Proceed with the decomposition shown in Fig. 22 and
return(true);

Check whether $\Gamma$ is eligible;
if so then
Proceed with the decomposition shown in Fig. 24 and
return(true);

/* Or else $\Gamma$ is not of clique-width $\leq 3 */
return(false);
End

If there is only one master component, then there is no proper partition. If there are two master components, then we have a proper partition. If there are more than two master components, then any bipartition of the sets of master components yields a proper partition.

The algorithm will now compute the proper components of the graph (note that we can develop the decomposition shown in Fig. 22 to break $\Gamma$ in one step into more than two subproblems). Clearly this can be accomplished in linear time.

It is not difficult to test in linear time whether $\Gamma$ has the structure shown in Fig. 24.

6. Proofs concerning Function Decompose

We now provide the proofs of various propositions and lemmas stated in Section 5. The next three subsections present the proofs associated with Sections 5.2, 5.2.1 and 5.2.2, respectively.

6.1. Decomposing a bilabeled $\Gamma$ (the disjoint cases proposition)

We now prove Proposition 14: if $\Gamma$ is of clique-width $\leq 3$ and not simple, then it belongs to exactly one of the sets $U_1$, $U_2$, $D_1$, or $D_2$. Recall that, for $\ell \in \{1, 2\}$, the following hold:

- $U_\ell: V^\ell_1 \neq \emptyset$ and removing the edges between $V^\ell_1$ and $V^\ell_2$ disconnects $\Gamma$ (for an example of a $U_1$ graph, see Fig. 14(a), where $V^1_1 = \{b\}$ and the removal of edges $bc$ and $bd$ disconnects the graph);
- $D_\ell: V^\ell_1$ is not connected, and removing edges between the coconnected components of $V^\ell_1$ disconnects $\Gamma$ (for an example of a $D_1$ graph, see Fig. 14(b), where $V^1_1 = \{b, c, d\}$; the coconnected components of $V^1_1$ are $\{b\}$, $\{c\}$ and $\{d\}$; and the removal of any pair of edges in $\{bc, bd, cd\}$ disconnects the graph).

We first show that we have not forgotten some case: i.e., $\mathcal{E} = U_1 \cup U_2 \cup D_1 \cup D_2$, where $\mathcal{E}$ is the set of graphs $H$ that are of clique-width $\leq 3$, bilabeled 1–2, connected, $\ell$-prime, and not simple (i.e., with more than three vertices, and not one of the particular cases PC1, PC2, or PC3, illustrated in Figs. 12 and 13(a)). Then, in Lemma 25 to Proposition 29, we will show that the unions are disjoint.

Lemma 23. Let $H \in \mathcal{E}$. Then:

- $H = \rho_{3 \to 1} \circ \eta_{3,2}(H_1 \oplus H_2) \implies H \in U_1$
- $H = \rho_{3 \to 2} \circ \eta_{3,1}(H_1 \oplus H_2) \implies H \in U_2$
- $H = \rho_{3 \to 1} \circ \eta_{3,1}(H_1 \oplus H_2) \implies H \in D_1$
- $H = \rho_{3 \to 2} \circ \eta_{3,2}(H_1 \oplus H_2) \implies H \in D_2$.

where $H_1$ and $H_2$ denote clique-width $\leq 3$ graphs.

Moreover, there exists a decomposition of $H$ corresponding to one of the left-hand-side terms.
Proof. First, we give some insight into what the various decompositions achieve. For example, the first decomposition consists of splitting the 1s into two parts by introducing the 3s in such a way that these 3s see all the 2s; we can thus remove the $E_{32}$ edges, and this disconnects the graph. This graph thus belongs to $\mathcal{U}_1$ (there are some 1s that see all the 2s, and removing these edges disconnects the graph).

Now, let $H$ be a clique-width $\leq 3$ graph, bilabeled 1–2, connected, $\ell$-prime, with more than 3 vertices, and without PC1 (PC2 and PC3 are not needed for this lemma).

The proof begins like that of Lemma 7. We look at a parse tree of $H$. The top operation cannot be an $\eta_{1,2}$, or else, as $H$ is $\ell$-prime, it would be the graph with two adjacent vertices, which is impossible. Thus the top operation is a $\rho_{3 \rightarrow x}$, $x \in \{1, 2\}$; $H = \rho_{3 \rightarrow x}(H')$, where $H'$ is $\ell$-prime and trilabeled. It is connected, and the only possibility is $H' = \eta_{a,b}(H'')$.

The next operation before that cannot be some $\eta_{i,j}$, or else we would have, for instance, $H'' = \eta_{1,2}(\eta_{1,3}(H''''))$, and $V_1$, an $\ell$-module of $H''$, would be a single universal vertex; thus $H$ would be a PC1, which contradicts $H \in \mathcal{E}$. So $H''$ is not connected, and we have (see Fig. 25, where the dashed heavy lines indicate biclique edges that are not present in $H_1 \cup H_2$, but will be added by the $\eta_{a,b}$ operation)

$$H = \rho_{3 \rightarrow x} \circ \eta_{a,b}(H_1 \oplus H_2), \quad \{a, b\} \subset \{1, 2, 3\}, x \in \{1, 2\},$$

where $H_1$ and $H_2$ are clique-width $\leq 3$ graphs.

Depending on the values of $x$ and $\{a, b\}$, we have six possible decompositions. But we observe that the case $\rho_{3 \rightarrow x} \circ \eta_{1,2}$ (we introduce the 3s among the 1s, and then remove the edges that link the remaining 1s to the 2s) is the same as the case $\rho_{3 \rightarrow 1} \circ \eta_{1,2}$ (we introduce the 3s among the 1s, and then remove the edges that link these 3s to the 2s), up to swapping labels 1 and 3 (i.e., relabeling the other side of $V_1$). The same observation applies to cases $\rho_{3 \rightarrow 2} \circ \eta_{1,2}$ and $\rho_{3 \rightarrow 2} \circ \eta_{1,3}$.

In conclusion, every such graph $H$ can be decomposed according to one of the four left-hand-side terms. □

We must now prove that this union is disjoint. Most of the following proofs will be based on the following principle: the structure of a graph allegedly belonging to an intersection is analysed, using the fact that some small graphs are not of clique-width $\leq 3$. We will show that such a graph has more and more properties, until we show that it cannot exist.

First, we show that a number of small graphs are forbidden.

Lemma 24. The bilabeled graphs$^1$ shown in Figs. 26(c), 27(b), 27(c), 29(d) and 17 are not of clique-width $\leq 3$.

Proof. All graphs in the statement of the lemma are bilabeled, $\ell$-prime, connected, with more than three vertices, and without PC1’s. Let us prove this lemma for the graphs in Fig. 27(c) (possibly the most difficult ones). Since $V_1$ and $V_2$ are symmetrical, there are only two cases of the possible construction to consider (following Lemma 23): either we split $V_1$ according to its cconnected components, or we relabel some vertices of $V_2$. As $V_2 = \emptyset$, and $V_1$ has only two cconnected components, $\{z_1\}$ and $\{x_1, y_1\}$, in fact there is no choice; we relabel, for instance, $z_1$ with 3, and we remove the 1 $\ast$ 3 edges. Vertex $x_1$ is then disconnected, but a trilabeled connected graph remains in which no $E_{ij}$ is complete; thus no operation is possible, and this graph is not of clique-width $\leq 3$. (Note that the dashed lines play no role in this proof.)

The same type of proof, with systematic trial of all possibilities, can just as easily be conducted for the other graphs of this lemma; we leave the details to the reader. □

---

$^1$ In the figures of these graphs, the label of a vertex is either the label itself or appears as a subscript of the name of the vertex. Note also that a figure may represent several graphs, because a dashed line stands for an edge or a non-edge [i.e., we have no information on the existence of that particular edge]; we must then prove that all these graphs are not of clique-width $\leq 3$. 

---
Fig. 26. Configuration of an $H \in U_1 \cap U_2$.

Fig. 27. Configuration of an $H \in D_1 \cap D_2$.

PC2 (Fig. 12(b)) and PC3 (Fig. 13(a)) will now show their usefulness, namely in taking care of graphs that would otherwise have been decomposable in several different fashions.

Lemma 25. $U_1 \cap U_2 = \emptyset$.

Proof. We proceed by letting $H \in U_1 \cap U_2$ (i.e., $V_1^a \neq \emptyset$, $V_2^a \neq \emptyset$, and thus $V_1^a = V_2^a = \emptyset$). Note that $V_1^i$ is not empty, since otherwise $V_1$ would be a single vertex that would see all of $V_2$, and thus $H$ would be a PC1.

Let $H'$ denote the disconnected graph obtained by removing from $H$ the edges between $V_1^a$ and $V_2$ (see Fig. 26(a)).

Let $C$ be a connected component of $H'$ that contains a vertex of $V_1^i$. As all the edges between $V_1^i$ and $V_2^a$ are present, $C$ contains these two sets. We then remark that a connected component of $V_2^a$ must have a neighbor in $V_1^i$, or else it would be an $\ell$-module, and thus a single vertex that would see all of $V_2$; this would result in a PC2. So $C$ must also contain $V_2^i$.

Let $C'$ be $H' \setminus C$; thus $C' \subseteq V_2^i$. Since $H'$ is not connected, $C' \neq \emptyset$. The neighborhood of $C'$ in $H$ is exactly $V_1^i$; now $C'$ is an $\ell$-module and thus a vertex, which we denote $x_2$ (note that $x_2$ has $V_1^a$ as its neighborhood in $H$). Applying the same ideas symmetrically, we find $x_1$, a vertex of $V_2^a$ that has $V_1^a$ as its neighborhood in $H$ (see Fig. 26(b)).

Let $V_2'^i = V_2^i \setminus \{x_2\}$. We claim that $V_2'^i \neq \emptyset$, and there exists an edge $y_2 - y_2'$ between a vertex of $V_2'^i$ and a vertex of $V_2'^a$. Otherwise, $V_2'^a$ would be an $\ell$-module, and thus a vertex with neighborhood exactly $V_1^i$, and we would have a PC2. We define $V_1'^i = V_1^i \setminus \{x_1\}$ and we find $y_1 - y_1'$, an edge between a vertex of $V_1'^i$ and a vertex of $V_2'^i$. Now, considering $x_1, y_1, y_2, y_1', y_2'$, we find one of the two graphs of Fig. 26(c). But (by Lemma 24), these graphs are not of clique-width $\leq 3$, which is a contradiction.

\[\square\]

Lemma 26. $U_1 \cap D_2 = \emptyset$; $U_2 \cap D_1 = \emptyset$.

Proof. Let $H \in U_1$. A vertex of $V_1$ sees all of $V_2$, so removing edges in $V_2$ does not affect the connectedness of $H$. In particular, $H$ remains connected when we remove the edges between the possible coconnected components of $V_2$, so $H \not\in D_2$.

By reasons of symmetry, the other intersection is empty as well. \[\square\]

Lemma 27. $D_1 \cap D_2 = \emptyset$. 
Proof. Let $H \in \overline{D}_1 \cap \overline{D}_2$. Removing the edges between the coconnected components of $V_1$ must disconnect $H$. But $V_2$ remains connected (since $H \in \overline{D}_2$); there is a connected subcomponent of $V_1$ that sees no vertex of $V_2$. It is an $\ell$-module, and thus a vertex denoted $x_1$, which belongs to some coconnected component denoted $ccc_1$. Moreover, every coconnected component of $V_1$ sees a vertex of $V_2$, or else it would be an $\ell$-module, and thus a vertex that would see all of $V_1$, and we would have a PC2. In particular, we can find a $y_1 \in ccc_1$ that has 2-neighbors (see Fig. 27(a)). With the same reasoning, we define $x_2 \in ccc_2$ of $V_2$, and we remark that every coconnected component of $V_2$ has 1-neighbors.

If we could find a $y_1 \in ccc_1$ that has 2-neighbors outside $ccc_2$, and a $y_2 \in ccc_2$ that has 1-neighbors outside $ccc_1$, we would find one of the graphs shown in Fig. 27(b), which is impossible, since they are not of clique-width $\leq 3$ by Lemma 24.

Thus, for instance, the 2-neighbors of $ccc_1$ are all in $ccc_2$. Let $y_1 \in ccc_1$ and $y_2 \in ccc_2$ be two such neighbors. Looking at another coconnected component of $V_2$, we find a $z_2 \in V_2 \setminus ccc_2$ that has a 1-neighbor denoted $z_1$. As $z_1 \notin ccc_1$, we find the graphs in Fig. 27(c), which is a contradiction, since they are not of clique-width $\leq 3$. □

Unfortunately, we have not been able to find an elegant proof that the last intersection should be empty as well; instead, we consider the different cases for the decomposition of the graph.

Lemma 28. $U_1 \cap \overline{D}_1 = \emptyset$; $U_2 \cap \overline{D}_2 = \emptyset$.

Proof. Let $H \in U_1 \cap \overline{D}_1$ (the other intersection is symmetrical). As $H \in U_1$, $H \notin U_2$ (Lemma 25) and $H \notin \overline{D}_2$ (Lemma 26), only two cases remain from the four possible constructions of $H$ of Lemma 23: $H = \rho_{3\to1} \circ \eta_{3,1}(H_1 \oplus H_2)$ or $H = \rho_{3\to1} \circ \eta_{3,1}(H_1 \oplus H_2)$.

First case: $H = \rho_{3\to1} \circ \eta_{3,1}(H_1 \oplus H_2)$ (we remove the edges between two groups of coconnected components of $V_1$).

Let $H' = H_1 \oplus H_2$ (and $V_1$ and $V_2$ still denote the label classes of $H$). Some vertices of $V_1$ are labeled with 3 in $H'$, and all $1 \ast 3$ edges exist in $H$. The coconnected components of $V_1$ are unlabeled in $H'$, or else they would not be coconnected. In particular, the connected subcomponents of $V_1$ are also connected in $H'$.

As there is a vertex of $V_1$ that sees all of $V_2$, $V_2$ is entirely contained in a connected component of $H'$, for instance, in $H_2$. Then $H_2$ must also contain $V_1^0$ and $V_1^0$. We may assume that $H_2$ is connected (up to transferring other connected components into $H_1$; see Fig. 28). $H_1$ consists of connected subcomponents of $V_1$ that contain only vertices of $V_1^0$. As there are no $\ell$-modules, these connected subcomponents contain at most one vertex, and, in any coconnected component of $V_1$, there is at most one such vertex. When present, this vertex is denoted $y_1$, or else $ccc_1$ contains only vertices of $H_2$.

But $ccc_1$, a coconnected component of $V_1$ that has a $y_1$, must have other vertices, or else $y_1 \in V_1^0$ would be universal to $V_1$, and $H$ is a PC2. Thus $H_2$ has a vertex in every coconnected component of $V_1$. Since some coconnected components of $V_1$ are unlabeled 1 and the others are unlabeled 3, it follows that $H_2$ is trilabeled. It is also connected, of clique-width $\leq 3$, and without edges $1 \ast 3$, so all the edges of type $1 \ast 2$ must be present (or, symmetrically, all the edges of type $3 \ast 2$).

Let $ccc_1$ be any coconnected component of $V_1$ unlabeled 1 in $H'$. We have just shown that all the vertices of $ccc_1 \setminus \{y_1\}$ belong to $V_1^0$; they form an $\ell$-module, and so consist of one vertex only. But this is impossible, since $H$ is either a PC3 if $y_1$ exists, or is a PC1 if $y_1$ does not exist.

Second case: $H = \rho_{3\to1} \circ \eta_{3,1}(H_1 \oplus H_2)$ (we remove the edges between $V_2$ and a subset of $V_1^0$).

The proof begins in a similar way to the first case. Let $K$ be the graph obtained from $H$ by removing the edges between the coconnected components of $V_1$; $K$ is not connected since $H \in \overline{D}_1$. Since there is a vertex of $V_1$ that sees all of $V_2$, $V_2$ belongs to a connected component of $K$. Looking at another connected component of $K$, we find a coconnected subcomponent of $V_1$ that has no 2-neighbor. It is an $\ell$-module, and thus a vertex $x$, in a coconnected component of $V_1$ denoted $ccc_1$ (see Fig. 29(a)).

If the $\rho_{3\to1}$ did not relabel all of $V_1^0$, the graph would remain connected after the $\eta_{3,2}$, since $V_1$ is connected and would still contain a vertex that sees all of $V_2$. Thus $H' = H_1 \oplus H_2$ is the graph obtained from $H$ by relabeling $V_1^0$ with 3, and removing the edges between $V_1^0$ and $V_2$ (Fig. 29(b)). We now speak of $V_1' = V_1^0$ and of $V_2' = V_1^0 \cup V_1^0$.

We observe that $V_1' \cup V_2' \subseteq V_1^0$ is connected; we may assume that $H_1$ is the connected component of $H'$ that contains $V_1' \cup V_2'$. Then $H_2$ consists of connected components of $V_2$ whose vertices have exactly $V_1^0$ as 1-neighbors in $H$. As there are no $\ell$-modules, $H_2$ is a single vertex, denoted $z$.

We first show that in fact $H_1$ does not contain vertices of $V_2$. Indeed, if it did, we would have a trilabeled connected graph of clique-width $\leq 3$, where neither $E_{12}$ (because of $x$) nor $E_{22}$ would be complete; so all the edges of type $1 \ast 3$ would be present. Thus the coconnected components of $V_1 = V_1^0 \cup V_1^0$ would be unlabeled (so that they are coconnected). We could

\[\text{Fig. 28. Configuration of an } H \in U_1 \cap \overline{D}_1: \text{first case.}\]
then find in $H$ a coconnected component that would be included in $V_3' = V_1'$; it would be an $\ell$-module, and thus a universal vertex, which is impossible.

So $H_1$ does not contain vertices labeled with 2, which means that, in $H, V_2 = \{z\} = H_2$ (and $V_1' = \emptyset$). As every coconnected component of $V_1$ must have a 2-neighbor so as to avoid $H$ being a PC2, they all must contain at least one vertex of $V_1'$. They must all have a vertex of $V_1'$ as well, since otherwise $H$ is a PC1. In other words, they all have vertices of $V_1'$, and of $V_2'$. Moreover, $H_1$ does not have $\ell$-modules, since $z$ distinguishes neither $V_1'$ nor $V_2'$; an $\ell$-module of $H_1$ would be an $\ell$-module of $H$. If $\text{ccc}_1$ does not have edges of type $3 \ast 1$, it would have only two vertices, one in $V_1'$ and one in $V_3'$; but this is impossible, since $H$ would be a PC3. So we can find the graph shown in Fig. 29(c) in $\text{ccc}_1$.

But the graph in Fig. 29(d) is not of clique-width $\leq 3$. This means that, in the other coconnected components, all the edges between $V_1'$ and $V_3'$ are present. But then they are not coconnected, which is impossible. \)

This completes the proof of Proposition 14. We can now state it in a more precise form, since, knowing to which set a graph $H$ belongs, we know how it must be decomposed.

**Proposition 29.** The union $\mathcal{E} = \mathcal{U}_1 \cup \mathcal{U}_2 \cup \overline{\mathcal{D}}_1 \cup \overline{\mathcal{D}}_2$ is disjoint.

Moreover,

- $H \in \mathcal{U}_1 \iff (\exists H_1, H_2 \in \text{cw}(3)) H = \rho_{1 \rightarrow 1} \circ \eta_{3, 2}(H_1 \oplus H_2)$
- $H \in \mathcal{U}_2 \iff (\exists H_1, H_2 \in \text{cw}(3)) H = \rho_{1 \rightarrow 2} \circ \eta_{3, 1}(H_1 \oplus H_2)$
- $H \in \overline{\mathcal{D}}_1 \iff (\exists H_1, H_2 \in \text{cw}(3)) H = \rho_{3 \rightarrow 1} \circ \eta_{3, 3}(H_1 \oplus H_2)$
- $H \in \overline{\mathcal{D}}_2 \iff (\exists H_1, H_2 \in \text{cw}(3)) H = \rho_{3 \rightarrow 2} \circ \eta_{3, 2}(H_1 \oplus H_2)$.

6.2. Case $\Gamma' \in \mathcal{U}_1$ (and $\Gamma' \in \mathcal{U}_2$)

We now present the proofs of the lemmas in Section 5.2.1. Assume that $\Gamma' \in \mathcal{U}_1$; we know that we have to relabel some vertices in $V_1'$ with 3 so as to disconnect the graph.

We first prove Lemma 17 (i.e., whether we correctly decide whether a non-partial connected component of $V_1$ is good or bad). Recall that a non-partial connected component $C$ of $V_1$ is called good (respectively, bad) if $\Gamma'$ is of clique-width $\leq 3$ implies that the bilabeled graph obtained from $C$ by relabeling all the vertices of $V_1' \cap C$ with 3 is of clique-width $\leq 3$ (respectively, of clique-width $> 3$).
We prove a more precise result: if \( X \) is of clique-width \( \leq 3 \) then the good connected components are characterized by the absence of a certain subgraph, namely, the square (see Fig. 17). First we examine Function Test-component (see Algorithm 6) and show that, under the assumption that \( \Gamma \) is of clique-width \( \leq 3 \), we correctly conclude whether a non-partial connected component of \( V_1 \) is good or bad.

**Lemma 30.** Let \( C \) be a non-partial connected component of \( V_1 \), let \( D \) be the bilabeled graph obtained from \( C \) by relabeling all the vertices of \( V_1^c \cap C \) with 3, and let \( C' \) be as defined in Algorithm 6.

If \( \Gamma \) is of clique-width \( \leq 3 \), the following properties are equivalent.

- All connected components of \( C' \) fulfill one of the following conditions: it has at most one vertex in \( V_1^c \); or it has at most one vertex in \( V_1^o \); or it has at least four vertices, and it is not of the shape shown in Fig. 19, where the vertices in \( V_1^o \) have label 3 and the vertices in \( V_1^c \) retain label 1;
- \( C \) is good (\( D \) is of clique-width \( \leq 3 \));
- \( D \) does not contain the square (see Fig. 17).

**Proof.** To simplify the terminology, we say that a connected component of \( C' \) is \( c \)-good if it satisfies one of the three conditions listed in the first bullet of the statement of the lemma.

We now show that a connected component of \( C' \) is \( c \)-good if and only if it is of clique-width \( \leq 3 \), which justifies the terminology. It is sufficient to show, for every connected component of \( C' \), that if it is \( c \)-good \( \iff \) it is of clique-width \( \leq 3 \) \( \iff \) it does not contain the square. (And if \( C' = \emptyset \), it is correct to decide that \( C \) is good.) These equivalences indeed lift up to \( D \).

- Given parse trees for the connected components of \( C' \), it is easy to find a parse tree for \( D \): we only have to add universal vertices or regroup connected components. As \( C' \) is a subgraph of \( D \), all the connected components of \( C' \) are of clique-width \( \leq 3 \) \iff \( D \) is of clique-width \( \leq 3 \);
- Adding or removing a universal vertex in a connected component cannot add or remove a square. Thus no connected component of \( C' \) contains the square \( \iff \) \( D \) does not contain the square.

(and if \( C' = \emptyset \), it is correct to decide that \( C \) is good).

Let \( c \) be a connected component of \( C' \) that has the labeling present in \( D \); then \( c \) is connected without a universal vertex.

If \( c \) is unilabeled, then it is \( c \)-good, of clique-width \( \leq 3 \) (as a subgraph of \( \Gamma \)), and without the square.

Two cases remain. First case: one label class of \( c \) contains exactly one vertex; thus \( c \) is \( c \)-good, without the square, and we must show that it is of clique-width \( \leq 3 \). Note that it has at least four vertices, or else it would have a universal vertex (since it is connected). Second case: \( c \) has at least two vertices in each label class, and we must show that it is \( c \)-good (not of the shape shown in Fig. 19) \iff it is of clique-width \( \leq 3 \) \iff it does not contain the square. We thus assume that \( c \) falls into one of these two cases.

Before dealing with these two cases, we first remark that \( D \) is \( \ell \)-prime, because an \( \ell \)-module of \( D \) would be an \( \ell \)-module of \( \Gamma \). Removing universal vertices cannot create \( \ell \)-modules, so \( c \) is also \( \ell \)-prime (but it may have bilabeled modules!).

To show that \( c \) is of clique-width \( \leq 3 \), we introduce graph \( K \), which is a supergraph of \( c \) and a subgraph of \( \Gamma \). We will show that \( K \in \mathcal{E} \) (in particular, \( \mathcal{U}_1 \)) by showing that \( K \) is not a PCI. Then, by Proposition 29, we know how to disconnect \( K \) to conclude that \( c \) is of clique-width \( \leq 3 \).

Let \( K \) be the subgraph of \( \Gamma \) induced by the vertices of \( c \) and a vertex of \( V_2 \) (recall that, in \( \Gamma \), and thus in \( K \) as well, all vertices of \( c \) have label 1). Since \( K \) is a subgraph of \( \Gamma \), it is of clique-width \( \leq 3 \). Furthermore, \( K \) is bilabeled, connected (since \( c \) is not unilabeled, it has at least one vertex of \( V_1^c \), which is linked to the 2 vertex), \( \ell \)-prime (because an \( \ell \)-module of \( K \) should be unilabeled 1, and should not be distinguished by the 2 vertex, and thus would be an \( \ell \)-module of \( c \)), and it has more than three vertices. In \( K \), there is no universal or isolated vertex in \( V_1 \), and the 2 vertex is not universal (it has at least one non-neighbor in the first case, or two in the second case); thus \( K \) is neither a PCI nor a PC2.

In fact, if \( K \) is not a PC3, then \( c \) is of clique-width \( \leq 3 \). Indeed, in this case \( K \in \mathcal{E} \), and furthermore \( K \in \mathcal{U}_1 \); Proposition 29 states that we must disconnect \( K \) by relabeling a part of \( V_1^c \) with 3 and removing the edges of type 3 \(*\ 2 \). Since \( c \) is connected, we must relabel in \( K \) all of \( V_1^c \), and disconnect the 2 vertex. The \( c \) vertices in \( K \) thus get the labels they have in \( c \); this shows that we must build \( K \) from \( c \), and thus \( c \) is indeed of clique-width \( \leq 3 \).

To conclude the first case, it is thus sufficient to show that \( K \) is not a PC3. Fig. 30 shows a \( K \) that is a PC3. Now, one of the sets \( V_1^o \) or \( V_1^n \) (defined to be \( V_1^o \setminus y \) and \( V_1^n \setminus x \), respectively) would be empty. The other would not be empty (since \( c \) is connected), and would be an \( \ell \)-module, and thus a single vertex; but this is impossible, since this vertex would be universal in \( c \).

To conclude the second case, we observe that \( c \) is bad (of the shape shown in Fig. 19) \iff \( K \) is of the shape shown in Fig. 30, (i.e., \( K \) is a PC3). In other words, \( c \) is good \iff \( K \) is not a PC3. We have already shown that \( K \) is not a PC3 \iff \( c \) is of clique-width \( \leq 3 \) \iff \( c \) does not contain the square (the square is not of clique-width \( \leq 3 \)). Thus, to have all the equivalences, it only remains to show that \( c \) does not contain the square \iff \( K \) is not a PC3.

We suppose that \( K \) is a PC3 to show the contrapositive (i.e., that \( c \) contains a square). As we are in the second case, the sets \( V_1^o \) and \( V_1^n \) shown in Fig. 30 each contain at least one vertex. If all the edges between these two sets were present, they would be \( \ell \)-modules, and thus would simply be two neighbors; but this is impossible, since we would have a universal vertex in \( c \). Thus we can find a non-edge \( a \ b \), thereby forming a square on \( \{a, b, x, y\} \). □
The types of the components are now correctly determined, and we must show that Function Decompose-$\mathcal{U}_1$ (see Algorithm 7), which decomposes the graphs $\Gamma \in \mathcal{U}_1$, is correct (Lemma 19).

Lemma 31. The following algorithm correctly determines whether it is possible for $\Gamma$ to be of clique-width $\leq 3$, and, if so, which vertices are to be relabeled.

1. If there are at least two bad connected components, or one bad and one partial, or if $V_1$ has only one connected component, which turns out to be bad, then $\Gamma$ is not of clique-width $\leq 3$.
2. Otherwise, if there are good connected components, we relabel with 3 their vertices of $V_1^u$ and proceed with the decomposition shown in Fig. 18.
3. Otherwise (i.e., all the connected components are partial), we relabel with 3 all the vertices of $V_1^u$, and undo an $\eta_{3,2}$, which disconnects the graph by the definition of $\mathcal{U}_1$ (see Fig. 20).

Proof. In fact the second case (eliminating good components with the decomposition shown in Fig. 18) is obviously correct, and the third case (when all the components are partial) is easy as well; since every component contains a vertex of $V_1$, we cannot leave a vertex that sees all of $V_2$ if we want to disconnect the graph; so we have to relabel all of $V_1^u$.

Thus only the first case requires a proof. We begin with a preliminary case, where there are no good connected components, and there are bad ones (and maybe partial ones). Since, by definition, we cannot relabel all the vertices of $V_1^u$ of a bad component, each one must keep at least one vertex that sees all of $V_2$. Since the partial components must also see $V_2$, it is impossible to disconnect the graph; thus $\Gamma$ is not of clique-width $\leq 3$.

In particular, we have just shown the third part of the first case; if $\Gamma$ has only one component, it cannot be of clique-width $\leq 3$ if this component is bad.

Let us assume that $\Gamma$ has a bad component, plus a bad or partial component. Let $H$ be the induced subgraph of $\Gamma$ obtained by removing the good components of $V_1$; $H$ is bilabeled, connected, $\ell$-prime (the good components did not distinguish $V_2$), and has more than three vertices.

$H$ is not a PC1, since the universal vertex could not be in $V_1$ (which contains at least two connected components) nor in $V_2$ (which does not see $V_1^u$, and every bad component contains at least one vertex of $V_1^u$).

$H$ is not a PC2, since no vertex of $V_2$ sees all or none of $V_1$, no vertex of $V_1$ that sees all of $V_2$ is isolated in $V_1$ (or $\Gamma$ would be a PC2), and no vertex of $V_1$ sees the rest of $V_1$.

Finally, $H$ is not a PC3, since no vertex of $V_2$ sees all the 1s; if a vertex has only one non-neighbor, which has the same label, then this label must be 1, and if this non-neighbor sees all the other 1s, then $V_1$ is connected.

Thus $H \in \mathcal{U}_1$, has no good components, but has a bad one; thus, as seen in the preliminary case, $H$ is not of clique-width $\leq 3$, so its supergraph $\Gamma$ is not either.  

6.3. Case $\Gamma \in \overline{\mathcal{D}}_1$ (and $\Gamma \in \overline{\mathcal{D}}_2$)

Finally, we deal with the case $\Gamma \in \overline{\mathcal{D}}_1$. We know that $\Gamma' = \rho_{3\rightarrow 1} \circ \eta_{3,1}(\Gamma'')$ for $\Gamma''$ a disconnected trilabeled graph, and we must decide which coconnected components of $V_1$ should be relabeled with 3. We have already solved the cases when there are only two coconnected components, or when there is a proper partition; the only unresolved case to be proved is Lemma 20.

To deal with the remaining case, recall that such a $\Gamma$ is said to be eligible if $\Gamma_d$ satisfies the following properties (see Fig. 23).

- There exists a unique connected component (called $c$) of $\Gamma_d$ that has vertices in several coconnected components of $V_1$.
- $d = V_2 \setminus c$ has neighbors in exactly one coconnected component (called $ccc_d$) of $V_1$.
- $c \cap ccc_d$ is connected subcomponent $\{y\}$, and $y$ is universal to $V_2 \cap c$.
- $ccc_d \setminus \{y\} = \emptyset$.
- All coconnected components of $V_1$ intersect $c$. 

Fig. 30. Case $K$ being a PC3.
Lemma 32. Let $\Gamma$ be a clique-width \( \leq 3 \) graph in $\overline{D}_1$ that has more than two coconnected components and has no proper partition. Then $\Gamma$ is eligible, and is correctly identified in Function Decompose-$\overline{D}_1$ (see Algorithm 8). Furthermore, the decomposition shown in Fig. 24 is correct.

Proof. For such a $\Gamma$, we examine $\Gamma'$, the disconnected trilabeled graph resulting from $\Gamma' = \rho^1 \circ \eta_{3,1}(\Gamma')$. We let $V'_1$, $V'_2$, and $V'_3$ denote the label classes of $\Gamma'$, i.e., $V_1 = V'_1 \cup V'_2$, and we have relabeled the vertices of $V'_3$. Recall that $\Gamma_d$ denotes the graph $\Gamma$ without the edges between the coconnected components of $V_1$ (see Fig. 23).

Either $V'_1$ or $V'_2$ (or both) is connected in $\Gamma'$ (we assume $V'_1$), since at least one of them consists of several coconnected components. Let $C$ be the connected component of $\Gamma'$ that contains all of $V'_1$ ($\Gamma_1$ in Fig. 24). $C$ contains the union of all the connected components of $\Gamma_d$ that have a vertex of $V'_1$. $C$ is trilabeled, or else it would be a proper partition. So $V'_2$ cannot be connected, or else $\Gamma'$ would be connected. It follows that $V'_2$ consists of a single coconnected component of $V_1$ (or else $V'_3$ would be connected). We now examine the structure of $C$ in more detail. $C$ is of clique-width $\leq 3$, connected, and without edges of type $1 * 3$ (which will be added by the $\eta_{3,1}$ operation). Thus all the edges of type $1 * 2$ or $3 * 2$ are present in $C$. We know that $E_{12}$ cannot be complete, or else $V'_1$ would be an $\ell$-module of $\Gamma'$, and would be a vertex; but a vertex cannot contain several coconnected components. Thus $E_{32}$ is complete in $C$, and $V'_2 \cap C$ is an $\ell$-module, and thus a vertex, denoted $y$; and $y$ sees all the rest of $C$ in $\Gamma$. At any rate, there is a vertex in $C$ that sees all its vertices with label 2.

This means that, in terms of union of connected components of $\Gamma_d$, $C$ contains a component $c$ of $\Gamma_d$ that has vertices of $V_2$ (the 2-neighbors of $y$), plus possibly some components that are just connected subcomponents of $V_1$. Moreover, $c$ has a vertex in every coconnected component of $V'_1$, or else one of them would not have 2-neighbors (a 2-neighbor of a vertex of $V'_1$ must be in $c$), and $\Gamma'$ would be a PC2. Let $c'$ be another connected component of $\Gamma_d$ that contains vertices of $V_2$ (if such a component exists). By the definition of $C$, the 1-neighbors of these vertices cannot be in $V'_1$, so they are in $V'_2$, and thus are in only one coconnected component. Thus $\Gamma_d$ has only one connected component that has vertices in several coconnected components, and this component is $c$.

If $V_2 \subseteq c$, the other connected components of $\Gamma_d$ are just connected subcomponents, which are $\ell$-modules, and thus vertices. In this case, there is furthermore only one vertex outside $C$ (to avoid $\ell$-modules), denoted $z$, where $z \in V'_2$. But, coming back to $\Gamma$, we see that vertex $y$ has only one non-neighbor, namely $z$, and $z$ sees only the other 1s; thus $\Gamma'$ is a PC3, which is impossible.

So there are other connected components of $\Gamma_d$ that contain vertices of $V_2$, and they can have a non-empty intersection with only one coconnected component, namely $V'_2$.

So $V'_2$ is found by Function Decompose-$\overline{D}_1$, and we have already seen that $c$ has only one vertex $y$ in $V'_2$, and that $y$ sees all of $c \cap V'_2$. Thus $\Gamma'$ is eligible.

The correctness of the decomposition presented in Fig. 24 is immediate. □

Thus, if $\Gamma'$ is not eligible, it is correct to determine that it is not of clique-width $\leq 3$; this completes the proof of the algorithm.

7. Concluding remarks

This paper contains the first step in the process of resolving the complexity status of clique-width recognition for fixed $k$. The most important open question is whether the recognition of graphs of clique-width $\leq k$, for fixed constant $k \geq 4$ is in P. Can the techniques developed in this paper generalize to larger values of $k$? To have some perspective on this, it is interesting to re-examine our algorithm. The general algorithm does not need to backtrack, thanks to a preprocessing step that computes the modular decomposition. Thus we avoid having to backtrack on unlabeled graphs. To generalize this to larger fixed values of $k$, we would need a generalization of modular decomposition, together with an established relationship with the clique-width question. This would require a significantly deeper understanding of the structure of graphs of restricted clique-width. The fact that our algorithm for clique-width $\leq 3$ uses split decomposition (which is a generalization of modular decomposition), to determine the initial candidate labelings might be a hint that the key to solving the clique-width problem for graphs of clique-width $\leq k$ for $k \geq 4$ lies in the understanding of a generalized modular decomposition. Continuing with the examination of our algorithm, we note that Function DECOMPOSE avoids backtracking by identifying particular independent cases. Can this be generalized to larger values of $k$? We note that handling the clique-width $\leq 3$ case is already quite tedious, and such an approach might be humanly intractable for $k \geq 4$. Again, it seems as though a deeper insight into the structure of graphs of bounded clique-width is required.

As a first step in determining the complexity of recognition of graphs of a fixed constant clique-width, it would be of interest to see if such algorithms can be developed for restricted families of graphs where the clique-width can be unbounded, such as unit-interval graphs and permutation graphs. In [20], Espelage et al. announced a linear-time algorithm for deciding “clique-width at most $k$” for graphs of bounded treewidth [20].

Acknowledgments

The authors wish to thank Janos Makowsky for many helpful conversations as well as an anonymous referee of a previous version of the paper for many helpful suggestions. Financial assistance for the first, fourth, and last authors was gratefully received from the Natural Science and Engineering Research Council of Canada. This research was carried out while the last
author was a Post-doctoral Fellow at the Department of Computer Science, University of Toronto; he wishes to thank the members of the department for their hospitality.

References


