Graphs of linear clique-width at most 3*

Pinar Heggernes† Daniel Meister† Charis Papadopoulos†

Abstract

A graph has linear clique-width at most k if it has a clique-width expression using at most k labels such that every disjoint union operation has an operand which is a single vertex graph. We give the first characterisation of graphs of linear clique-width at most 3, and we give the first polynomial-time recognition algorithm for graphs of linear clique-width at most 3. In addition, we present new characterisations of graphs of linear clique-width at most 2. We also give a layout characterisation of graphs of bounded linear clique-width; a similar characterisation was independently shown by Gurski and by Lozin and Rautenbach.

1 Introduction

Many generally hard problems can be solved efficiently on restricted graph classes. One graph parameter to measure the computational complexity of problems on restricted input graphs is clique-width. As an example, problems that can be expressed in a certain monadic second-order logic can be solved in linear time on graphs whose clique-width is bounded by a constant, assuming that an appropriate clique-width expression is given [5]. The clique-width of a graph is defined as the smallest number of labels that are needed for constructing the graph using the graph operations 'vertex creation', 'union', 'join' and 'relabel'. The graph parameter *linear clique-width* is closely related to clique-width and defined by a restriction of the allowed clique-width operations to only 'vertex creation', 'join' and 'relabel'. The corresponding *clique-width minimization problem* and *linear clique-width minimization problem* are NP-hard even on complements of bipartite graphs [8]. A graph class can have bounded clique-width but unbounded linear clique-width, for example cographs and trees [11].

The relationship between clique-width and linear clique-width is similar to the relationship between treewidth and pathwidth, and the two pairs of parameters are related [8, 11, 13]. Clique-width can be viewed as a more general concept than treewidth since there are graphs of bounded clique-width but unbounded treewidth (for instance complete graphs), whereas graphs of bounded treewidth have bounded clique-width [6]. While treewidth is widely studied and well-understood the knowledge about clique-width is still limited. The study of the more restricted parameter linear clique-width is a step towards a better understanding also of clique-width. For example, NP-hardness of the clique-width minimization problem is obtained by showing that the linear clique-width minimization problem is NP-hard [8].

*This work is supported by the Research Council of Norway through grant 166429/V30. An earlier version of this work was presented at TAMC 2008.

†Department of Informatics, University of Bergen, Norway. Emails: pinar.heggernes@ii.uib.no, daniel.meister@ii.uib.no, charis.papadopoulos@ii.uib.no

With this paper, we contribute to the study of linear clique-width with several results. The main result that we report is the first characterisation of graphs that have linear clique-width at most 3. We give a graph decomposition scheme that exactly characterises the graphs of linear clique-width at most 3 as the graphs that can be decomposed completely. We show that a decomposition, if possible, can be computed in $\mathcal{O}(n^2m)$ time, which gives the first polynomial-time algorithm to decide whether a graph has linear clique-width at most 3. Furthermore, we show that graphs of linear clique-width at most 3 are both cocomparability graphs and weakly-chordal graphs. Prior to this work, polynomial-time recognition algorithms for graphs of linear clique-width at most k were known only for $k \leq 2$ [9]. For bounded clique-width, polynomial-time recognition algorithms for graphs of clique-width at most k are known only for $k \leq 3$ [2, 6]. Only little is known about other characterisations of graphs of bounded clique-width. Graphs of clique-width at most 2 are characterised as the class of cographs [6], but no characterisation other than the recognition algorithm is known for graphs of clique-width at most 3.

A graph decomposition scheme is a procedure that recursively decomposes a graph into smaller graphs by partitioning the vertex set. Decompositions are often used in the design of efficient algorithms; well-known examples are linear-time recognition algorithms for comparability graphs and cocomparability graphs [17]. As a preliminary step for obtaining our decomposition scheme for graphs of linear clique-width at most 3, we consider the smaller class of graphs of linear clique-width at most 2. We give new characterisations of this graph class, in particular a characterisation by a decomposition scheme. This decomposition scheme is a special case of our decomposition scheme for graphs of linear clique-width at most 3.

Before focusing on graphs of small bounded linear clique-width, we give more general results for linear clique-width. First we present and demonstrate the use of a new layout characterisation of graphs of bounded linear clique-width. Treewidth and pathwidth have algorithmically useful characterisations through vertex layouts and embeddings into particular graph classes. No such result is known for clique-width. Recently, Gurski as well as Lozin and Rautenbach gave a layout characterisation of graphs of bounded linear clique-width [10, 15]. Our result is similar but independent and has a simpler statement and proof. Second, we give a characterisation of linear clique-width through a decomposition scheme that preserves the linear clique-width of each decomposable subgraph. Note that even the trivial decomposition into connected components does not have this property, as the linear clique-width of a disconnected graph can be larger than the linear clique-width of each of its connected components. A simple example is the $2K_2$, which has linear clique-width 3 but its two connected components have linear clique-width 2. For clique-width, several graph operations have been studied that preserve the clique-width of a graph [6, 13]. One of the most important such results shows that the clique-width of a graph is equal to the maximum clique-width of its prime induced subgraphs, making modular decomposition such a preserving operation [6]. We give such a preserving operation also for linear clique-width and show that sets of false twins can be ignored.

This paper is organised as follows. In the next section we give the necessary background and notation, in particular the definitions of clique-width and linear clique-width. Section 3 presents our layout characterisation of linear clique-width, and Section 4 presents the decomposition scheme that preserves linear clique-width. In Section 5, we consider characterisations of graphs of linear clique-width at most 2. Sections 6 and 7 are devoted to graphs of linear clique-width at most 3. In Section 6, we give the decomposition scheme characterisation and prove structural properties of graphs of linear clique-width at most 3, and in Section 7, we give the polynomial-time recognition algorithm. Final remarks and open problems are discussed in Section 8.

2 Graph preliminaries and linear clique-width

We consider simple finite undirected graphs. For a graph $G = (V, E)$, we denote its vertex set as $V(G) = V$ and its edge set as $E(G) = E$. An *empty graph* has empty vertex and edge set, and an *edgeless graph* has empty edge set. Edges of G are denoted as uv, and we say that u and v are *adjacent*. If $uv \notin E$ for $u \neq v$, we say that u and v are *non-adjacent*. The *subgraph of G induced by $S \subseteq V$* is denoted as $G[S]$. For a set $S \subseteq V$, we denote the graph $G[V \setminus S]$ by $G \setminus S$, and for a vertex x of G, we denote the graph $G[V \setminus \{x\}]$ by $G-x$. Let u be a vertex of G. The *neighbourhood* of u is $N_G(u) = \{v : uv \in E\}$. The *closed neighbourhood* of u is $N_G[u] = N_G(u) \cup \{u\}$. For a set $S \subseteq V$, $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. Two vertices x, y are called *true twins* if $N_G[x] = N_G[y]$ and they are called *false twins* if $N_G(x) = N_G(y)$. The *degree* of u is the number of its neighbours and denoted as $d_G(u)$. We call u *isolated* in G if $d_G(u) = 0$, we call u *almost universal* if $d_G(u) = |V(G)| - 2$, and we call u *universal* if $d_G(u) = |V(G)| - 1$. A *clique* of G is a set of pairwise adjacent vertices, and an *independent set* of G is a set of vertices that are pairwise non-adjacent.

In a graph $G = (V, E)$, a *path* is a sequence $(x_0, \ldots, x_r)$ of distinct vertices such that $x_i x_{i+1} \in E$ for $0 \leq i < r$. If $r \geq 2$ and additionally x_0 and x_r are adjacent then $(x_0, \ldots, x_r)$ is a *cycle*. A graph is *connected* if there is a path between every pair of vertices; otherwise the graph is *disconnected*. The maximal connected subgraphs of a graph are called *connected components*. The *complement* of G, denoted as $\overline{G}$, has vertex set V and two vertices are adjacent if and only if they are not adjacent in G. If $\overline{G}$ is connected then we say that G is *co-connected*. The *co-connected components* of a graph are the connected components of its complement.

Let G and H be two vertex-disjoint graphs. The *disjoint union* of G and H, denoted as $G \oplus H$, is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. The *join* of G and H, denoted as $G \otimes H$, is the graph obtained from $G \oplus H$ by adding all edges between the vertices of G and the vertices of H.

2.1 Special graphs and graph classes

For a set $\mathcal{H}$ of graphs, a graph is called *$\mathcal{H}$-free* if it does not contain a graph from $\mathcal{H}$ as induced subgraph. A *chord* in a path or cycle is an edge between non-consecutive vertices of the path or cycle. The chordless path on k vertices is denoted as P_k and the chordless cycle on k vertices is denoted as C_k. The complement of C_4 is denoted as $2K_2$, which is the graph on four vertices and two edges that do not share an endpoint. The complement of the disjoint union of two (vertex-disjoint) copies of P_3 is denoted as co-$(2P_3)$.

Cographs are defined inductively as follows: a single vertex is a cograph, the disjoint union of two cographs is a cograph, and the complement of a cograph is a cograph. It is well-known that cographs are exactly the P_4-free graphs [3]. A subclass of cographs are *threshold graphs*. A graph is a *threshold graph* if and only if its vertices can be partitioned into a clique and an independent set such that the independent-set vertices can be ordered by neighbourhood inclusion [16].

A *cocomparability graph* is the complement of a transitively-orientable graph. A characterisation of cocomparability graphs is by vertex orderings. A graph G is a cocomparability graph if and only if there is an ordering $\langle x_1, \ldots, x_n \rangle$ of its vertices such that for every triple $i < j < k$, $x_i x_k \in E$ implies that $x_i x_j \in E$ or $x_j x_k \in E$ [14]. Such vertex orderings are called *cocomparability orderings*.

A *weakly-chordal graph* is a graph that does not contain a C_k for $k \geq 5$ or its complement as

induced subgraph. Cographs are weakly-chordal graphs. For further properties of the mentioned graph classes we refer to [1].

2.2 Clique-width and linear clique-width

Clique-width was introduced by Courcelle, Engelfriet, Rozenberg [4]. For a natural number $k \geq 1$, a *k-labelled* graph is a graph whose vertices are assigned labels from $\{1, \ldots, k\}$. The following operations are for k-labelled graphs:

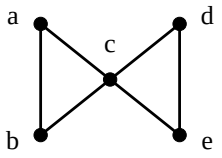
1) create a k-labelled graph on a single vertex x with label i, for $i \in \{1, \ldots, k\}$;
   denoted as $i(x)$
2) relabel every vertex with label i by label j, for $i, j \in \{1, \ldots, k\}$;
   denoted as $\rho_{i \to j}$
3) join all vertices with label i with all vertices with label j by adding edges, for $i, j \in \{1, \ldots, k\}$ and $i \neq j$;
   denoted as $\eta_{i,j}$
4) compute the disjoint union of two k-labelled graphs;
   denoted as $\oplus$.

The *clique-width* of a graph G, denoted as $\mathrm{cwd}(G)$, is the smallest number k such that G is equivalent to a k-labelled graph that is constructed using the above operations. *Clique-width expressions* are built using the defined operations. Examples are given in Figure 1. Note that operands for the create, relabel and join operations are given in round brackets after the operation whereas the two operands of the disjoint union operation are to the left and right hand side of the '$\oplus$' symbol. If a clique-width expression uses only k labels we call it a *k-expression*. We say that a k-expression t *defines* a graph G if G is equivalent to the k-labelled graph obtained from t.

The linear clique-width of a graph was introduced by Gurski and Wanke [11]. It is defined by restricting the disjoint union operation of clique-width. For a graph G, the *linear clique-width* of G, denoted as $\mathrm{lcwd}(G)$, is the smallest number k such that there is a k-expression that defines G and that is of the following form: every occurrence of a disjoint union operation has at least one operand that is a graph on a single vertex. The definition of linear clique-width is equivalent to the following: a graph has linear clique-width at most k if it is equivalent to a k-labelled graph that is constructed from an empty graph by applying the following operations: add a new vertex, relabel, join. In other words, the disjoint union operation is joined with the vertex creation operation to an operation that adds an isolated vertex with specified label to the already constructed graph. All three operations take only one operand, so that they define a linear expression. As an example, expression t_2 in Figure 1 can be made into such an expression by simply deleting the '$\oplus$' symbols. When we give explicit linear clique-width expressions, we use a different notation. We avoid the usage of round brackets and we write the operations in reversed order, ending with the operation that is applied last. As an example, the equivalent of expression t_2 in Figure 1 is the following expression:

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

Every clique-width expression defines a binary parse tree. Tree nodes with two children correspond to disjoint union operations in the expression. Parse trees for our simplified definition



$$t_1 = \eta_{1,2}\Big(\rho_{2\to 1}\Big(\eta_{1,2}\big(1(a)\oplus 2(b)\big)\oplus \eta_{1,2}\big(1(d)\oplus 2(e)\big)\Big)\oplus 2(c)\Big)$$

$$t_2 = \eta_{2,3}\Big(3(e)\oplus \rho_{3\to 2}\Big(\eta_{2,3}\big(3(d)\oplus \eta_{1,2}\big(2(c)\oplus \rho_{2\to 1}(\eta_{1,2}(1(a)\oplus 2(b)))\big)\big)\Big)\Big)$$

Figure 1: A 2-expression, t_1, and a 3-expression, t_2, that define the left hand side graph.

of linear clique-width do not contain nodes with two children, thus are paths. In this sense, the relationship between clique-width and linear clique-width can be viewed analogous to the relationship between treewidth and pathwidth. The linear clique-width of a graph is at least its clique-width. The difference between clique-width and linear clique-width of a graph can be arbitrarily large. As an example, cographs and trees have clique-width at most 3 [6] but they have unbounded linear clique-width [11]. It is easy to see that edgeless graphs are exactly the graphs of clique-width 1 and linear clique-width 1, since the creation of an edge requires at least two different labels.

3 A layout characterisation of linear clique-width

We show that linear clique-width is equivalent to a layout width parameter. This width parameter mainly measures the number of different neighbourhoods in a set of vertices with respect to the outside. Such a characterisation was already given by Gurski [10] and Lozin and Rautenbach [15], however, the width parameter that we use has a significantly simpler formulation. We use our layout characterisation to show results for linear clique-width. In particular, we apply this characterisation in this section to show that connected components of a disconnected graph can be constructed independently by a linear clique-width expression with smallest number of labels.

A *layout* for a graph G is a linear ordering of its vertices, usually defined as a bijective mapping between the number set $\{1,\ldots,|V(G)|\}$ and $V(G)$. Let $G=(V,E)$ be a graph. For $A\subseteq V$, a *group* in A is a maximal set of vertices with the same neighbourhood in $V\setminus A$. Note that two groups in A are either equal or disjoint, implying that the group relation defines a partition of A. By $\nu_G(A)$, we denote the number of groups in A. Let β be a layout for G. Let x be a vertex of G and let p be the position of x in β, i.e., $p=\beta^{-1}(x)$. The *set of vertices to the left of x with respect to β* is $\{\beta(1),\ldots,\beta(p-1)\}$ and denoted as $L_\beta(x)$, and the *set of vertices to the right of x with respect to β* is $\{\beta(p+1),\ldots,\beta(|V(G)|)\}$ and denoted as $R_\beta(x)$. We write $L_\beta[x]$ and $R_\beta[x]$ if x is included. The function ad_β is a $\{0,1\}$-valued function on the set of vertices of G with respect to β. It is defined as follows: $\mathrm{ad}_\beta(x)=1$ if one of the following conditions is satisfied:

- $\{x\}$ is a group in $L_\beta[x]$
- $\{x\}$ is not a group in $L_\beta[x]$ and all (other) vertices in the group containing x are neighbours of x
- $\{x\}$ is not a group in $L_\beta[x]$ and there are a non-neighbour y of x in the group of $L_\beta[x]$ containing x and a neighbour z of x in $L_\beta(x)$ such that y and z are non-adjacent;

$\mathrm{ad}_\beta(x)=0$ if none of the conditions is satisfied. Note that the first condition is a special case

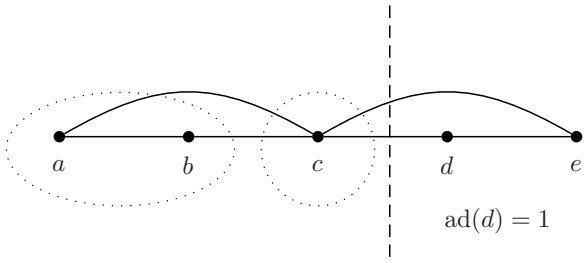


Figure 2: A specific layout for the graph in Figure 1. The set of vertices to the left of d has two groups, and $\mathrm{ad}(d) = 1$ for this layout.

of the second condition. We distinguish the two conditions for convenience reason and to keep the definition clear.

Definition 1. *The* groupwidth of a graph G with respect to a layout β for G *is the smallest number k such that $\nu_G(L_\beta(x)) + \mathrm{ad}_\beta(x) \leq k$ for all $x \in V(G)$. The groupwidth of G with respect to β is denoted as* $\mathrm{gw}(G, \beta)$.

The groupwidth *of a graph G is the smallest number k such that there is a layout β for G satisfying* $\mathrm{gw}(G, \beta) \leq k$. *The groupwidth of G is denoted as* $\mathrm{gw}(G)$.

For an example, consider Figure 2. Given is a layout β for the graph G depicted in Figure 1. We pick vertex d and see that $L_\beta(d)$ has two groups, $\{a, b\}$ and $\{c\}$. Furthermore, $L_\beta[d]$ has also two groups, $\{a, b\}$ and $\{c, d\}$, and d is adjacent to all other vertices in its group. Hence, $\mathrm{ad}_\beta(d) = 1$, and thus $\mathrm{gw}(G, \beta) \geq 3$. It is not hard to see that $\mathrm{gw}(G, \beta) = 3$.

Theorem 1. *For every graph G,* $\mathrm{lcwd}(G) = \mathrm{gw}(G)$.

Proof Let $G = (V, E)$ be a graph on n vertices. First we show $\mathrm{lcwd}(G) \geq \mathrm{gw}(G)$. Let $a = a_1 \cdots a_r$ be a linear clique-width expression for G that uses k labels. We show that $\mathrm{gw}(G) \leq k$. Let $i_1 < \cdots < i_n$ such that $a_{i_1}, \ldots, a_{i_n}$ are the vertex creation operations in a. Let $G_1, \ldots, G_n$ be the following labelled graphs: G_j is defined by expression $a_1 \cdots a_{i_j - 1}$. Note that G_1 is an empty graph. Observe that $\nu_G(V(G_j)) \leq k$, since vertices in different groups in $V(G_j)$ (with respect to G) must have different labels. We define a layout β for G as follows: the vertices of G appear in β as in $a_{i_1} \cdots a_{i_n}$. Let x be the vertex added to G_j by operation a_{i_j}. Note that $L_\beta(x) = V(G_j)$. We show that $\nu_G(V(G_j)) + \mathrm{ad}_\beta(x) \leq k$. If $\nu_G(V(G_j)) < k$ then clearly $\nu_G(V(G_j)) + \mathrm{ad}_\beta(x) \leq k$. So, let $\nu_G(V(G_j)) = k$. This means that all k labels are assigned to vertices in G_j. Hence, a_{i_j} creates x and assigns a label that is assigned to a vertex in G_j. We show that in this case $\mathrm{ad}_\beta(x) = 0$. Let A denote the set of vertices of G_j having the same label as x. By assumption, A is not empty. Then, x is not adjacent to any of the vertices from A in G, and $A \cup \{x\}$ is part of the same group in $L_\beta[x]$ (thus the first two conditions of the definition of ad are not satisfied). Let A' denote the group in $L_\beta[x]$ containing x; hence $A \cup \{x\} \subseteq A'$. Suppose there is a non-neighbour y of x in $A' \setminus (A \cup \{x\})$. Then, all vertices with the same label in G_j as y belong to the same group in $V(G_j)$ as A. This, however, means that $\nu_G(V(G_j)) < k$, which is a contradiction to the assumption. Hence, all vertices in $A' \setminus (A \cup \{x\})$ are neighbours of x, and the third condition of the definition of ad cannot be satisfied, in particular, since every neighbour of x in G is a neighbour of every vertex in A. Hence, $\mathrm{ad}_\beta(x) = 0$. By choosing a as using the smallest number of labels, we conclude this part of the proof.

Next we show $\mathrm{lcwd}(G) \leq \mathrm{gw}(G)$. Let $\beta = \langle x_1, \ldots, x_n \rangle$ be a layout for G. We show that $\mathrm{lcwd}(G) \leq \mathrm{gw}(G, \beta)$. We define linear clique-width expressions $a_1, \ldots, a_n$ that define labelled graphs $G_1, \ldots, G_n$ such that G_j is equivalent to $G[\{x_1, \ldots, x_j\}]$ and is labelled with exactly $\nu_G(L_\beta[x_j])$ labels. Let $k =_{\text{def}} \mathrm{gw}(G, \beta)$. Clearly, $a_1 =_{\text{def}} 1(x_1)$ defines G_1 properly. Now, let $j \in \{1, \ldots, n-1\}$ and assume that a_j has already been defined. Then, G_j is equivalent to $G[\{x_1, \ldots, x_j\}]$ and is labelled with exactly $\nu_G(L_\beta[x_j]) = \nu_G(L_\beta(x_{j+1}))$ labels. Without loss of generality, we can assume that the labels in G_j are from $\{1, \ldots, k\}$. We distinguish two cases. First, let $\mathrm{ad}_\beta(x_{j+1}) = 1$. Then, $\nu_G(L_\beta(x_{j+1})) < k$ and there is a label in $\{1, \ldots, k\}$ that is not used in G_j; let c be the smallest unassigned label. We define a_{j+1} by iteratively appending operations to a_j. We begin with $a_{j+1} =_{\text{def}} a_j\ c(x_{j+1})$. If x_{j+1} is not adjacent to any of the vertices in $L_\beta(x_{j+1})$, we are done. Otherwise, let c' be the label of a group in $L_\beta(x_{j+1})$ that contains neighbours of x_{j+1}. Append operation $\eta_{c,c'}$. By definition of a group, all vertices with label c' are adjacent to x. Repeat this step until x_{j+1} is adjacent to all its neighbours that are in $L_\beta(x_{j+1})$. Then, the labelled graph defined by current a_{j+1} is equivalent to $G[\{x_1, \ldots, x_{j+1}\}]$. Note that we have used exactly $\nu_G(L_\beta(x_{j+1})) + \mathrm{ad}_\beta(x_{j+1})$ labels so far. It can be that vertices from the same group in $L_\beta[x_j]$ have different labels. We add relabel operations to a_{j+1}. This can be done since every group in $L_\beta(x_{j+1})$ is entirely contained in a group in $L_\beta[x_{j+1}]$. This completes the definition of a_{j+1}. With the given arguments, it is clear that G_{j+1}, that is defined by a_{j+1}, is equivalent to $G[\{x_1, \ldots, x_{j+1}\}]$ and contains exactly $\nu_G(L_\beta[x_{j+1}])$ labels. As the second case, let $\mathrm{ad}_\beta(x_{j+1}) = 0$. We define a_{j+1} similar to the previous case with exception for the choice of c. In this case, a_{j+1} creates x_{j+1} using a label that is already assigned to a vertex in G_j. Let A be the group in $L_\beta[x_{j+1}]$ that contains x_{j+1}. According to the definition of ad, A contains at least two vertices. Furthermore, A can contain at most two groups of $L_\beta(x_{j+1})$. If all vertices in $A \setminus \{x_{j+1}\}$ are adjacent to x_{j+1} then $\mathrm{ad}_\beta(x_{j+1}) = 1$, which is a contradiction to the assumption. Hence, A contains a group A' of $L_\beta(x_{j+1})$ of vertices that are non-adjacent to x_{j+1}. We choose their label as c. It remains to show that this choice does not add wrong edges. But again, the definition of ad ensures that every neighbour of x_{j+1} that is in G_j is a neighbour of every vertex in A'. Hence, a_n is a linear clique-width k-expression that defines G_n, and G_n is equivalent to G. By choosing β of groupwidth $\mathrm{gw}(G)$, we conclude this part of the proof. ∎

The proof shows an even stronger result: there is a 1-to-1 correspondence between linear clique-width expressions using the smallest number of labels and layouts of smallest groupwidth. The proof also shows that the groupwidth of a layout is a (tight) lower bound on the number of labels every linear clique-width expression with this vertex ordering uses. Finally, the construction in the second part of the proof gives a simple algorithm for determining a linear clique-width expression using the smallest number of labels with the fixed vertex ordering. The corresponding notion is also known as *relative (linear) clique-width* and was introduced and studied by Lozin and Rautenbach [15]. Our proof provides a simple and efficient algorithm for computing the relative linear clique-width.

For the second result of this section, we consider linear clique-width expressions for disconnected graphs. It is obvious that a clique-width expression can construct the connected components of a disconnected graph separately and combine them in a last step using the disjoint union operation. For linear clique-width, this is not possible. However, it is intuitive to assume that also linear clique-width expressions construct the connected components of a disconnected graph separately. We show in the following that this is indeed the case by constructing a layout of smallest groupwidth with this property, thereby applying the characterisation of Theorem 1. Though the result is expected the proof is surprisingly non-trivial.

Lemma 2. *Let G be a graph with connected components $G_1, \ldots, G_l$. Let β be a layout for G. Then, there is a layout β' for G in which the vertices of each connected component appear consecutively and in the same order as in β and such that* $\mathrm{gw}(G, \beta) \geq \mathrm{gw}(G, \beta')$.

Proof Let $k =_{\mathrm{def}} \mathrm{gw}(G, \beta)$. We prove the statement by induction over the number of connected components. If G has only one connected component then G is connected and the statement is true using β as β'. Let G have at least two connected components. If G is edgeless, every connected component contains exactly one vertex and we conclude the statement again by using β as β'. So, let G not be edgeless. Then, $\mathrm{gw}(G, \beta) \geq 2$. Let $\beta_1, \ldots, \beta_l$ be layouts obtained from β by restricting to the vertices of $G_1, \ldots, G_l$, respectively. Suppose there is $i \in \{1, \ldots, l\}$ such that $\mathrm{gw}(G_i, \beta_i) \leq k - 1$. Let layout δ be obtained from β by deleting the vertices of G_i. Then, δ is a layout for $G \setminus V(G_i)$, and $\mathrm{gw}(G \setminus V(G_i), \delta) \leq k$. We apply the induction hypothesis to $G \setminus V(G_i)$ and δ and obtain layout δ'. Let β' be obtained from δ' and β_i by appending β_i at the end of δ'. Note that the vertices of each connected component of G appear consecutively in β' and the vertices of G_i are at the end. According to assumption and by construction, $\mathrm{gw}(G, \beta') \leq k$. As the last case, let $\mathrm{gw}(G_1, \beta_1) = \cdots = \mathrm{gw}(G_l, \beta_l) = k$. For every $i \in \{1, \ldots, l\}$, determine the leftmost vertex u_i of G_i with respect to β_i such that $\nu_{G_i}(L_{\beta_i}(u_i)) + \mathrm{ad}_{\beta_i}(u_i) = k$. Let G_j be the connected component with u_j rightmost in β among $u_1, \ldots, u_l$. Similar to the previous case, we obtain layout δ' for $G \setminus V(G_j)$. Let β' be obtained from δ' and β_j by appending β_j at the end of δ'. We show that $\mathrm{gw}(G, \beta') \leq k$. Suppose there is a vertex x such that $\nu_G(L_{\beta'}(x)) + \mathrm{ad}_{\beta'}(x) > k$. By definition of u_j and the construction of β', x is not to the left of u_j in β'. Choose x leftmost possible in β'. Then, $\nu_G(L_{\beta'}(x)) = k$ and $\mathrm{ad}_{\beta'}(x) = 1$. Otherwise, if $\nu_G(L_{\beta'}(x)) > k$, there is a vertex to the left of x that starts a new group and thus x was not chosen as leftmost. We consider the groups in $L_{\beta'}(x)$:

- exactly one group contains the vertices of $G \setminus V(G_j)$
- $k - 1$ groups contain only vertices of G_j.

Note that the one group with vertices of $G \setminus V(G_j)$ can also contain vertices of G_j, namely the vertices that do not have further neighbours. The vertices in the $k - 1$ groups all have a neighbour in $R_{\beta_j}[x]$. Then, the $k - 1$ groups of $L_{\beta'}(x)$ are also groups of $L_\beta(x)$ according to the construction of β'. Hence, all vertices in $L_\beta(x)$ of connected components different from G_j are in the same group, say A. By the choice of u_j, every connected component has a vertex in $L_\beta(u_j)$, and by assumption $k \geq 2$, every connected component of G has at least two vertices, thus every vertex has a neighbour in G. Suppose a vertex in A has a neighbour in $R_\beta(x)$. Then, no vertex of G_j that is in $L_\beta(x)$ can be in A, which means that only $k - 1$ groups in $L_\beta(x)$ contain vertices of G_j. This, however, contradicts the choice of u_j. Thus, no vertex in A has a neighbour in $R_\beta[x]$. Hence, $L_\beta(x) = L_{\beta'}(x)$. But then $\mathrm{ad}_\beta(x) = \mathrm{ad}_{\beta'}(x)$, which contradicts $\mathrm{gw}(G, \beta) = k$. We conclude that $\mathrm{gw}(G, \beta') \leq k$. ∎

4 A graph reduction that preserves linear clique-width

It is known that modular decomposition is a reduction operation that preserves clique-width in the following sense: the clique-width of a graph is equal to the maximum clique-width among its prime induced subgraphs [6]. Thus, clique-width is robust with respect to modules. In this section, we consider the analogue problem for linear clique-width. A simple observation

shows that linear clique-width is not robust with respect to arbitrary modules: cographs have unbounded linear clique-width. We restrict the notion of a module and show that linear clique-width is robust with respect to such restricted modules. The results of this section are useful both for computing the linear clique-width of a graph and for proving structural results. Examples of such applications are given in Sections 5 and 7.

For a graph G, a *module* M is a set of vertices of G that all have the same neighbours outside of M in G.

Definition 2. *Let $G = (V, E)$ be a graph and let M be a set of vertices of G. We call M a* maximal independent-set module *of G if M is both an independent set and a module of G and no proper superset of M has this property.*

As examples, an edgeless graph has exactly one maximal independent-set module, and a complete graph on n vertices has n maximal independent-set modules. Let G be a graph. Two vertices u and v of G are in relation $\sim_{\text{ft}}$, denoted as $u \sim_{\text{ft}} v$, if and only if they are false twins, i.e., they have the same open neighbourhood. The symmetric relation for true twins was considered by Roberts in connection with his study of indifference graphs [18]. Note that $\sim_{\text{ft}}$ is an equivalence relation, and the corresponding equivalence classes are exactly the maximal independent-set modules of G. In particular, different maximal independent-set modules have empty intersections. The *quotient graph* of G with respect to $\sim_{\text{ft}}$, denoted as $G/\sim_{\text{ft}}$, is obtained as follows: there is a vertex for every maximal independent-set module of G, and two vertices are adjacent if and only if the corresponding maximal independent-set modules contain vertices that are adjacent. It is clear that $G/\sim_{\text{ft}}$ is isomorphic to an induced subgraph of G. A maximal independent-set module is called *trivial* if it contains only one vertex.

Lemma 3. *For every graph G, the maximal independent-set modules of $G/\sim_{\text{ft}}$ are trivial.*

Proof Suppose $G/\sim_{\text{ft}}$ has a non-trivial maximal independent-set module M. Let a and b be vertices in M, and let M_a and M_b be the maximal independent-set modules of G corresponding to a and b, respectively. Let u and v be vertices in M_a and M_b, respectively. We show that $N_G(u) = N_G(v)$. Since a and b are non-adjacent in $G/\sim_{\text{ft}}$, u and v are non-adjacent in G. Let w be a vertex of G and different from u and v. If w is adjacent to u then w is not contained in M_a and therefore contained in a maximal independent-set module M_c. We denote the vertex of $G/\sim_{\text{ft}}$ corresponding to M_c by c. According to definition, c is adjacent to a in $G/\sim_{\text{ft}}$ and by assumption c is adjacent to b. Thus, w is adjacent to v in G. With a symmetry argument, it also follows that every neighbour of v is a neighbour of u in G. Hence, $u \sim_{\text{ft}} v$, and u and v belong to the same maximal independent-set module. This contradicts the assumption, and the lemma follows. ∎

As a direct corollary of Lemma 3, we obtain that graphs $G/\sim_{\text{ft}}$ and $(G/\sim_{\text{ft}})/\sim_{\text{ft}}$ are isomorphic. An algorithm for computing the maximal independent-set modules of a graph is given in Figure 3. The algorithm uses a partition refinement approach and has a linear-time implementation (see, for instance, [12]).

The main result of this section is given in the next lemma.

Lemma 4. *For every graph G,* $\text{lcwd}(G) = \text{lcwd}(G/\sim_{\text{ft}})$.

Proof Since $G/\sim_{\text{ft}}$ is isomorphic to an induced subgraph of G, the inequality $\text{lcwd}(G/\sim_{\text{ft}}) \leq \text{lcwd}(G)$ clearly holds. For showing the converse, let M_x for $x \in V(G/\sim_{\text{ft}})$ denote the maximal

Algorithm `Maximal independent-set modules`

begin
  let $\mathcal{M}$:= $\{V(G)\}$;
  for every vertex x of G **do**
    obtain $\mathcal{N}$ from $\mathcal{M}$ by partitioning every set in $\mathcal{M}$ into a set of neighbours and a set of non-neighbours of x;
    set $\mathcal{M}$:= $\mathcal{N} \setminus \{\emptyset\}$
  end for
end.

Figure 3: An algorithm for computing the maximal independent-set modules of a graph.

independent-set module of G corresponding to x. Let a be a linear clique-width expression for $G/\sim_{\text{ft}}$ and let ℓ_x be the label of x when adding x in the expression a. We define a linear clique-width expression a' for G in the following way. The creation operation for x in a is replaced by a sequence of creation operations for all vertices in M_x; all vertices in M_x are created with label ℓ_x. In other words, we replace the appearance of $\ell_x(x)$ in a with $\ell_x(v_1)\cdots\ell_x(v_{|M_x|})$ for $v_1,\ldots,v_{|M_x|}$ the vertices in M_x. Note that a' does not use more labels than a. For showing that the graph defined by a' is indeed equivalent to G, it suffices to consider a single module M_x. All vertices in M_x receive the same label, so are pairwise non-adjacent and have the same neighbours. The result follows with an easy induction. Choosing a as using the smallest number of labels, we conclude $\text{lcwd}(G) \leq \text{lcwd}(G/\sim_{\text{ft}})$, which completes the proof. ∎

5 Graphs of linear clique-width at most 2

The smallest non-trivial class of graphs of bounded linear clique-width is the class of graphs of linear clique-width at most 2. Gurski showed that graphs of linear clique-width at most 2 are exactly the $\{2K_2, P_4, \text{co-}(2P_3)\}$-free graphs [9]. In this section, we add several new characterisations, among others a decomposition scheme and a vertex reduction scheme. Using this we give a linear-time recognition algorithm for graphs of linear clique-width at most 2.

Definition 3. *The class of* simple cographs *is inductively defined as follows:*

(1) an empty graph is a simple cograph

(2) if A is a simple cograph and B is an edgeless graph, then $A \otimes B$ and $A \oplus B$ are simple cographs.

Theorem 5. *For a graph G, the following statements are equivalent:*

(1) G is a simple cograph

(2) $G/\sim_{\text{ft}}$ is a threshold graph

(3) G can be reduced to an empty graph by repeatedly deleting an isolated vertex, a universal vertex or a false twin vertex.

Proof We show three implications. Let $G = (V, E)$ be a graph.

(1) ⇒ (2) Let G be a simple cograph. If G is edgeless then $G/\sim_{\text{ft}}$ is a graph on a single vertex and therefore a threshold graph. If G contains an edge, there are edgeless graphs $A_1,\ldots,A_r$

and operations $\odot_i \in \{\oplus, \otimes\}$ for $2 \leq i \leq r$ such that $G = (\cdots(A_1 \odot_2 A_2)\cdots) \odot_r A_r$. We assume that r is smallest possible. This means that $\odot_2 = \otimes$ and that there is no $3 \leq i \leq r$ such that $\odot_{i-1} = \odot_i = \oplus$. Then, the maximal independent-set modules of G are exactly the sets $V(A_1), \ldots, V(A_r)$. Let $a_1, \ldots, a_r$ be vertices from $A_1, \ldots, A_r$, respectively. It holds that $G[\{a_1, \ldots, a_r\}]$ is isomorphic to $G/\sim_{\text{ft}}$. Note that $G[\{a_1, \ldots, a_r\}] = (\cdots(G[\{a_1\}] \odot_2 G[\{a_2\}])\cdots) \odot_r G[\{a_r\}]$. This shows that $G/\sim_{\text{ft}}$ is a threshold graph: $\{a_i : \odot_i = \oplus\}$ is an independent set in $G[\{a_1, \ldots, a_r\}]$, the other vertices form a clique and the independent-set vertices can be ordered by neighbourhood inclusion.

(2) ⇒ (3) Let $G/\sim_{\text{ft}}$ be a threshold graph. Let (C, D) be a partition of the vertex set of $G/\sim_{\text{ft}}$ such that C is an independent set and D is a clique of $G/\sim_{\text{ft}}$ and such that the vertices in C can be ordered by neighbourhood inclusion. Denote by M_x the maximal independent-set module of G corresponding to vertex x of $G/\sim_{\text{ft}}$. Note that the vertices in M_i are pairwise false twins. Mark a vertex from every module M_x. A reduction of the desired form is obtained by repeated deleting first all unmarked vertices (which are always false twin vertices) and then isolated and universal vertices. Note that the second part is successful since a graph is a threshold graph if and only if it can be reduced by repeatedly deleting isolated or universal vertices.

(3) ⇒ (1) Let $\sigma = \langle x_1, \ldots, x_n\rangle$ be a vertex ordering for G such that x_i is isolated, universal or false twin in $G_i =_{\text{def}} G[\{x_i, \ldots, x_n\}]$. Let σ be chosen such that false twin vertices appear leftmost possible, which means that, if possible, a false twin vertex is picked instead of a universal or isolated vertex. Suppose there is $1 \leq i < n$ such that x_{i+1} is a false twin vertex and x_i is not. Independent of whether x_i is universal or isolated in G_i, x_{i+1} is a false twin vertex also in G_i and x_i is universal or isolated in $G_i - x_{i+1}$. Hence, x_{i+1} can be deleted before x_i. We conclude that there is $1 \leq j \leq n$ such that $x_1, \ldots, x_{j-1}$ are deleted as false twin vertex and none of $x_j, \ldots, x_n$ is deleted as false twin vertex. Let $c_1, \ldots, c_{s'}$ be the vertices among $x_j, \ldots, x_n$ (and in the corresponding order) that are deleted as isolated vertices, and let $d_1, \ldots, d_{s''}$ be the vertices among $x_j, \ldots, x_n$ that are deleted as universal vertices. Then, every vertex d_i is adjacent to all vertices in a set $\{c_i, \ldots, c_{s'}\}$ for an appropriate choice of i. Hence, there is an ordering $\langle a_1, \ldots, a_r\rangle$ of the vertices in $C \cup D$ such that $G[\{x_j, \ldots, x_n\}] = (\cdots(A_1 \odot_2 A_2)\ldots) \odot_r A_r$ for A_i the subgraph of G induced by $\{a_i\}$ and appropriate choices of $\odot_i \in \{\oplus, \otimes\}$. Denote by M_i the maximal set of vertices that are false twin with a_i in G. Note that every such set M_i contains exactly one vertices from $\{x_j, \ldots, x_n\}$, and every vertex from $\{x_1, \ldots, x_{j-1}\}$ is contained in such a set. Then, $G = (\cdots(G[M_1] \odot_2 G[M_2])\cdots) \odot_r G[M_r]$, and G is a simple cograph. ∎

Note that the third part of the proof also gives a normalisation result about reduction sequences, that false twin vertices can be deleted first and then only isolated or universal vertices are deleted.

Corollary 6. *Simple cographs can be recognised in linear time.*

Proof This follows from Theorem 5 and the facts that $G/\sim_{\text{ft}}$ can be computed in linear time and threshold graphs can be recognised in linear time [16]. ∎

Theorem 7. *A graph has linear clique-width at most 2 if and only if it is a simple cograph.*

Proof Our proof distinguishes two cases; we apply the groupwidth characterisation of linear clique-width (Theorem 1). Let $G = (V, E)$ not be a simple cograph. Let $\beta = \langle x_1, \ldots, x_n\rangle$ be a layout for G. We show that $\text{gw}(G, \beta) \geq 3$. If there is $1 \leq j \leq n$ such that $\nu_G(L_\beta(x_j)) + \text{ad}_\beta(x_j) \geq 3$, we are done. So assume the contrary. Since G is not a simple cograph, there

is $1 < i < n$ such that $G[\{x_1, \ldots, x_i\}]$ is a simple cograph and $G[\{x_1, \ldots, x_{i+1}\}]$ is not. This particularly means that x_{i+1} has a neighbour and a non-neighbour in $\{x_1, \ldots, x_i\}$. Therefore, $\nu_G(L_\beta(x_{i+1})) = 2$ and $\mathrm{ad}_\beta(x_{i+1}) = 0$. Let A and A' be the two groups in $L_\beta(x_{i+1})$. One of the two groups, say A, contains only neighbours and the other group only non-neighbours of x_{i+1}. From $\mathrm{ad}_\beta(x_{i+1}) = 0$, it follows that the group in $L_\beta[x_{i+1}]$ that contains x_{i+1} contains the vertices in A', and this implies that every vertex in A is adjacent to every vertex in A'. Thus, $G\big[L_\beta[x_{i+1}]\big] = G[A] \otimes (G[A'] \oplus G[\{x_{i+1}\}])$. Let j be smallest such that $\{x_1, \ldots, x_j\}$ contains a vertex from both A and A'. Since $\{x_1, \ldots, x_{j-1}\}$ and $\{x_j\}$ belong to different groups, $\nu_G(L_\beta(x_j)) = 1$ and $\mathrm{ad}_\beta(x_j) = 1$. Since $\nu_G(L_\beta(x_{i'})) = 2$ for every $j + 1 \leq i' \leq i$, it holds that $\mathrm{ad}_\beta(x_{i'}) = 0$ for every $j+1 \leq i' \leq i$. It follows that A or A' is an independent set, depending on which group vertex x_j belongs to. If A is an independent set then $(G[A'] \oplus G[\{x_{i+1}\}]) \otimes G[A]$ is a simple cograph, if A' is an independent set then $G[A] \otimes G[A' \cup \{x_{i+1}\}]$ is a simple cograph. This, however, contradicts the assumption about $G[\{x_1, \ldots, x_{i+1}\}]$ not being a simple cograph. Hence, $\mathrm{gw}(G, \beta) \geq 3$.

For the converse, let G be a simple cograph. According to the definition, there are edgeless graphs $A_1, \ldots, A_r$ and operations $\odot_i \in \{\otimes, \oplus\}$ such that $G = (\cdots(A_1 \odot_2 A_2)\ldots) \odot_r A_r$. Let $x^i_1, \ldots, x^i_{s_i}$ be the vertices of A_i. We show that $\beta = \langle x^1_1, \ldots, x^1_{s_1}, x^2_1, \ldots, x^r_{s_r}\rangle$ is a layout for G of groupwidth at most 2: $L_\beta(x^i_1)$ has exactly one group, and since A_i is edgeless, $L_\beta(x^i_j)$ for $2 \leq j \leq s_i$ has at most two groups. So, $\mathrm{ad}_\beta(x^i_j) = 0$ for all $1 \leq i \leq r$ and $2 \leq j \leq r_i$, and we conclude $\mathrm{gw}(G) \leq 2$. ∎

Theorem 7 together with Gurski's result shows that $\{2K_2, P_4, \text{co-}(2P_3)\}$-free graphs are exactly the simple cographs. We have thus shown that this graph class can be recognised in linear time. Such a result was not known before. In fact, Theorem 5 (3) results in a linear-time *certifying* algorithm for recognising graphs of linear clique-width at most 2, outputting either a decomposition of G according to Definition 3, if yes, or an induced subgraph of G that is a $2K_2$, a P_4 or a co-$(2P_3)$, if no.

6 Graphs of linear clique-width at most 3

Graphs of linear clique-width at most 2 were shown to have a simple structure. The situation changes already for graphs of linear clique-width at most 3. In this section, we characterise the graphs of linear clique-width at most 3 by a decomposition scheme. This characterisation will lead to an efficient recognition algorithm (Section 7). The decomposition scheme can be considered as a generalisation of the decomposition scheme for simple cographs (Definition 3). At the end of this section, we apply our characterisation and show that graphs of linear clique-width at most 3 are cocomparability graphs and weakly-chordal graphs.

We can say that the decomposition scheme in Definition 3 defines simple cographs in a "1-dimensional manner". For graphs of linear clique-width at most 3, we have to add another dimension. We define a class of formal expressions, that are interpreted as graph descriptions. These expressions can be considered as 2-dimensional expressions. An *lc3-expression* is inductively defined as follows, where $d \in \{l, r\}$ and all sets A, A_1, A_2, A_3, A_4 may also be empty:

(d1) (A) is an lc3-expression, where A is a set of vertices.

(d2) Let T be an lc3-expression and let A be a set of vertices not containing a vertex appearing in T; then $(d[T], A)$ is an lc3-expression.

(d3) Let T be an lc3-expression and let A_1, A_2, A_3, A_4 be pairwise disjoint sets of vertices not containing a vertex appearing in T; then $(A_1|A_2, d[T], A_3|A_4)$ is an lc3-expression.

(d4) Let T be an lc3-expression and let A_1, A_2, A_3, A_4 be pairwise disjoint sets of vertices not containing a vertex appearing in T, and let p be one of the following number sequences: 123, 132, 312, 321, 12, 32 (not allowed are 213, 231, 21, 23, 13, 31); then $T \odot (A_1|A_2, \bullet)$, $T \odot (\bullet, A_1|A_2)$ and $T \circ (A_1, A_2|A_3, A_4, d[p])$ are lc3-expressions where $\odot \in \{\oplus, \otimes\}$.

This completes the definition of lc3-expressions. The *graph defined by an lc3-expression* is obtained according to the following inductive definition. An lc3-expression also associates a vertex partition with the defined graph. Let T be an lc3-expression. Then, $G(T)$ is the following graph, where T' always means an lc3-expression and A, A_1, A_2, A_3, A_4 are sets of vertices and $d \in \{l, r\}$:

(i1) Let $T = (A)$; then $G(T)$ is the edgeless graph on vertex set A; the vertex partition associated with $G(T)$ is $(A, \emptyset)$.

(i2) Let $T = T' \odot (A_1|A_2, \bullet)$ for $\odot \in \{\oplus, \otimes\}$, and let $G(T')$ with vertex partition (B, C) be given; then $G(T)$ is obtained from $G(T')$ by adding the vertices of A_1 and A_2 and executing two operations: (1) a union (if $\odot = \oplus$) or a join (if $\odot = \otimes$) between B and $A_1 \cup A_2$ and (2) a join between A_2 and C; that is, $B \odot (A_1 \cup A_2)$ and $A_2 \otimes C$; the vertex partition associated with $G(T)$ is $((B \cup A_1 \cup A_2), C)$.

(i3) The case $T = T' \odot (\bullet, A_1|A_2)$ is similar to the previous case, where the operations are $C \odot (A_1 \cup A_2)$ and $A_2 \otimes B$ and the vertex partition is $(B, (C \cup A_1 \cup A_2))$.

(i4) Let $T = T' \circ (A_1, A_2|A_3, A_4, d[p])$, and let $G(T')$ with vertex partition (B, C) be given; then $G(T)$ is obtained from $G(T')$ by adding the vertices in $A_1 \cup A_2 \cup A_3 \cup A_4$ and edges; the added edges are the result of a sequence of join operations, and the sequence is specified by p. We have three start sets, A_2, B, C, and three extension sets, A_3, A_1, A_4, that correspond to respectively A_2, B, C; the extension sets will be added to their corresponding start sets after performing join operations, so that we obtain the final three sets $A_2 \cup A_3$, $B \cup A_1$ and $C \cup A_4$; join operations are performed between the three sets, where each of which may be the start set or the already extended set.

We perform two or three join operations, depending on the sequence p and with the following meanings: 1 means a join operation between A_2- and B-vertices, 2 means a join operation between B- and C-vertices, 3 means a join operation between A_2- and C-vertices. It is important to notice that, for example, A_2-vertices are the vertices from A_2 or $A_2 \cup A_3$, depending on whether A_2 has been extended or not when the join operation is performed.

The operation application scheme is as follows: perform a join operation between the two start sets specified through the first number in p, and then add the extension sets to the two start sets involved in the first join operation; perform the join operation that is specified through the second number in p (note that one of the two sets is already extended and the other set is a start set), and then add the remaining extension set to its corresponding start set; if p contains three numbers, perform the join operation that is specified through the third number in p.

$G(T)$ is associated with one of the two vertex partitions:
if $d = l$ then $((B \cup A_1 \cup A_2 \cup A_3), (C \cup A_4))$, and
if $d = r$ then $((B \cup A_1), (C \cup A_4 \cup A_2 \cup A_3))$.

(i5) Let $T = (d[T'], A)$, and let $G(T')$ with vertex partition (B, C) be given; then $G(T)$ is the graph defined by $T' \otimes (\bullet, A|\emptyset)$ or $T' \otimes (A|\emptyset, \bullet)$ for $d = l$ or $d = r$, respectively; the vertex partition associated with $G(T)$ is $(B \cup C, A)$.

(i6) Let $T = (A_1|A_2, d[T'], A_3|A_4)$, and let $G(T')$ with vertex partition (B, C) be given; then $G(T)$ is the graph defined by $T' \circ (A_1, A_3|A_4, A_2, d[p'])$ where $p' =_{\text{def}} 132$ or $p' =_{\text{def}} 312$ for $d = l$ or $d = r$, respectively; the vertex partition associated with $G(T)$ is $((A_1 \cup A_2 \cup B \cup C), (A_3 \cup A_4))$.

This completes the definition of the graph defined by an lc3-expression. As a remark, we understand definitions (i5) and (i6) as "2-dimensional", since the given vertex partition is not increased in a monotone way but changed completely. Later in this section, we will see that definition (i4) is captured by what we will call an lc3-composition.

Definition 4. *A graph G is an* lc3-graph *if there is an lc3-expression T such that $G = G(T)$.*

We show that lc3-graphs are exactly the graphs of linear clique-width at most 3. We partition the proof into several smaller results. Remember that every lc3-expression defines a vertex partition, that is associated with the lc3-expression and the defined graph.

Lemma 8. *Let $G = (V, E)$ be an lc3-graph with lc3-expression T and vertex partition (B, C). Let $S \subseteq V$. Then, $G[S]$ is an lc3-graph, and there is an lc3-expression T_S for $G[S]$ with vertex partition $(B \cap S, C \cap S)$.*

Proof By definition of lc3-expressions, every vertex of G appears in exactly one lc3-expression operation of T and, thus in exactly one set of the vertex partition. It is clear that deleting a vertex x of G from the set of its appearance in T yields T' that is an lc3-expression which exactly defines $G-x$ with vertex partition $(B\backslash\{x\}, C\backslash\{x\})$. Iterated application of this operation proves the statement. ∎

For an lc3-expression T, denote by $G^C(T)$ the subgraph of $G(T)$ that is induced by the second component of the vertex partition for T. In other words, if $G(T)$ is associated with vertex partition (B, C) then $G^C(T) = G(T)[C]$.

Lemma 9. *Let T be an lc3-expression.*

(1) $G^C(T)$ is a simple cograph.

(2) Let (B, C) be the vertex partition for $G(T)$. There are no four vertices u, v, x, z in $G(T)$ such that $u, v \in B$ and $x, z \in C$ and $ux, vz \in E(G(T))$ and $uz, vx \notin E(G(T))$.

Proof We prove the two statements separately.

(1) We prove the statement by induction over the definition of lc3-expressions. If $T = (A)$ then $G^C(T)$ is an empty graph. Empty graphs are simple cographs. Let T' be an lc3-expression, for which the claim holds, and let A, A_1, A_2, A_3, A_4 be sets of vertices. If $T = (d[T'], A)$ or $T = (A_1|A_2, d[T'], A_3|A_4)$ for $d \in \{l, r\}$ then $G^C(T)$ is an edgeless graph on vertex set A or $A_3 \cup A_4$, respectively; then $G^C(T)$ is a simple cograph. If $T = T' \odot (A_1|A_2, \bullet)$ then $G^C(T) = G^C(T')$. If $T = T' \circ (A_1, A_2|A_3, A_4, l[p])$ for arbitrary value of p then $G^C(T) = G^C(T') \oplus G(T)[A_4]$. According to induction hypothesis, $G^C(T')$ is a simple cograph, and since $G(T)[A_4]$ is an edgeless graph, $G^C(T)$ is a simple cograph. Similarly, if $T = T' \odot (\bullet, A_1|A_2)$ then $G^C(T) = G_C(T') \odot G(T)[A_1 \cup A_2]$ is a simple cograph, since $G(T)[A_1 \cup A_2]$ is edgeless. Finally, let $T = T' \circ (A_1, A_2|A_3, A_4, r[p])$. Depending on p, $G^C(T)$ is one of the following graphs:

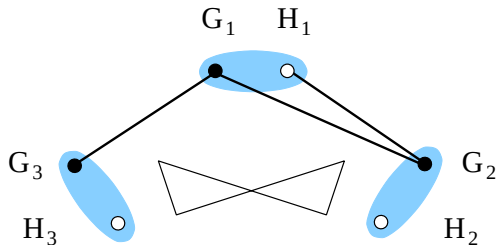


Figure 4: The result of an lc3-composition of the graphs $G_1, G_2, G_3, H_1, H_2, H_3$. The thick lines from graphs G_1 and H_1 represent joins. The bow tie between $G_2 \oplus H_2$ and $G_3 \oplus H_3$ means either a join or no edge at all.

- $(G^C(T') \otimes G(T)[A_2 \cup A_3]) \oplus G(T)[A_4]$
- $(G^C(T') \otimes G(T)[A_2]) \oplus G(T)[A_3 \cup A_4]$
- $(G^C(T') \oplus G(T)[A_4]) \otimes G(T)[A_2 \cup A_3]$.

All these graphs are simple cographs, and we conclude the first proof.

(2) Assume that G contains vertices u, v, x, z such that $u, v \in B$, $x, z \in C$ and $ux, vz \in E(G(T))$ and $uz, vx \notin E(G(T))$. Let T' be the minimal subexpression of T that contains x and z. Hence, at least one of x and z is contained in the last operation of T'. Suppose $G(T')$ does not contain u or v; without loss of generality, let u not be contained in $G(T')$. By checking all possibilities, this can only mean that u is adjacent to x and z in $G(T)$ or u is non-adjacent to x and z in $G(T)$, which contradicts the assumption. So, $G(T')$ already contains u, v, x, z. It is clear that the last operation of T' cannot be of the forms (A) or $(d[\cdot], A)$ or $\odot(A_1|A_2, \bullet)$. If the last operation of T' is of the form $\odot(\bullet, A_1|A_2)$ then x or z is contained in A_2 and must be adjacent to both u and v. If the last operation of T' is of the form $(A_1|A_2, d[\cdot], A_3|A_4)$ then $x, z \in A_3 \cup A_4$ and the neighbourhood of one vertex is contained in the neighbourhood of the other. Finally, let the last operation of T' be of the form $\circ(A_1, A_2|A_3, A_4, d[p])$. If $d = l$ then $x, z \in C \cup A_4$ and the neighbourhood of every vertex in A_4 is contained in the neighbourhood of every vertex in C. Thus $d = r$ must hold. But then $u, v \in B \cup A_1$, and we conclude a similar inclusion property for u and v. Hence, vertices u, v, x, z cannot have the described property, and we conclude the proof. ∎

Property (2) of Lemma 9 can be considered a weak form of $2K_2$-freeness of lc3-graphs. A third property of lc3-graphs is a closure property for a special composition operation. Let G_1, G_2, G_3 be an edgeless graph, a simple cograph and an lc3-graph (in arbitrary assignment) and let H_1, H_2, H_3 be edgeless graphs. Graphs may also be empty. The result of the disjoint union of these six graphs and additional edges is called *lc3-composition*. The additional edges are given in Figure 4. The bow tie means either join between $G_2 \oplus H_2$ and $G_3 \oplus H_3$ or no edges at all. The result of the join case is called *complete* lc3-composition and the result without edges for the bow tie is called *incomplete* lc3-composition.

Lemma 10. *Let G_1, G_2, G_3 be an edgeless graph, a simple cograph and an lc3-graph and let H_1, H_2, H_3 be edgeless graphs. Then, both the complete and the incomplete lc3-composition of these graphs yields an lc3-graph. Furthermore, there is an lc3-expression T for every $i \in \{1, 2, 3\}$ and every lc3-composition such that one of the two partition sets for $G(T)$ is equal to $V(G_i) \cup V(H_i)$.*

Proof We consider complete and incomplete lc3-composition separately and distinguish different assignments. Let G' and G'' be an lc3-graph and a simple cograph, respectively. Then, there are lc3-expressions T' and T'' for $G' \oplus G''$ and $G' \otimes G''$, respectively, such that $(V(G'), V(G''))$ is the vertex partition for $G(T')$ and $G(T'')$. Such lc3-expressions can be obtained from an lc3-expression T for G', starting from $(l[T], \emptyset)$ and adding operations of the forms $\oplus(\bullet, A|\emptyset)$ and $\otimes(\bullet, A|\emptyset)$ for $G' \oplus G''$ and $\oplus(\bullet, \emptyset|A)$ and $\otimes(\bullet, \emptyset|A)$ for $G' \otimes G''$. The proof of Lemma 9 describes such a construction in reverse. We distinguish the cases with respect to the edgeless graph among G_1, G_2, G_3. We first consider incomplete lc3-compositions and consider H_2 and H_3 to be empty. Let G_1 be edgeless, and let T be an lc3-expression for $G_2 \oplus G_3$ where the vertex partition for $G(T)$ groups into $V(G_2)$ and $V(G_3)$. Then, the following lc3-expressions

$$(d[T], V(H_1)) \oplus (\bullet, \emptyset|V(G_1)), \quad T \otimes (\emptyset|V(G_1), \bullet) \oplus (\emptyset|V(H_1), \bullet), \quad T \otimes (\bullet, V(H_1)|V(G_1))$$

and the complementary versions, where the $\bullet$ symbols change sides, are lc3-expressions for the incomplete lc3-composition of $G_1, \ldots, H_3$ where H_2 and H_3 are empty. If these graphs are non-empty, they are edgeless graphs and the contained vertices are isolated in the composition graphs. Appropriate lc3-expression operations can be added, which concludes this part. Now, let G_2 be edgeless. Let T be an lc3-expression for $G_1 \otimes G_3$ that groups the vertices into $V(G_1)$ and $V(G_3)$. For the following lc3-expressions, we assume that $V(G_1)$ is the right partition set. The other case is similar. Let $T' =_{\text{def}} T \oplus (\bullet, V(H_1)|\emptyset)$. Then, the following expressions show the claim where the remaining cases are obtained analogously as described above:

$$T' \oplus (\emptyset|V(G_2), \bullet), \quad (l[T'], V(G_2)), \quad T' \otimes (\bullet, V(G_2)|\emptyset)\,.$$

Finally, let G_3 be edgeless. Let T be an lc3-expression for $G_1 \otimes G_2$ grouping the vertices into $V(G_1)$ and $V(G_2)$. We assume that $V(G_1)$ is the left partition set. Then, we conclude with the following lc3-expressions:

$$T \circ (V(H_1), V(G_3)|\emptyset, \emptyset, d[12]), \quad (r[T \otimes (\bullet, V(H_1)|\emptyset)], V(G_3))\,.$$

For the case of complete lc3-expressions, we similarly list lc3-expressions. We begin with the case of G_1 being edgeless. Let T be an lc3-expression for $G_2 \otimes G_3$ that groups into $V(G_2)$ and $V(G_3)$ with $V(G_3)$ the left partition set. Then,

$$(V(H_3)|V(H_2), l[T], V(G_1)|V(H_1)), \quad (V(H_3), V(G_1)|V(H_1), V(H_2), d[132])$$

for $d \in \{l, r\}$ are lc3-expressions for the complete lc3-composition of $G_1, \ldots, H_3$. Now, let G_2 be edgeless, and let T be an lc3-expression for $G_1 \otimes G_3$ that groups into $V(G_1)$ and $V(G_3)$ where $V(G_3)$ is the left partition set. Let $T' =_{\text{def}} T \oplus (V(H_3)|\emptyset, \bullet) \oplus (\bullet, V(H_1)|\emptyset)$. Then,

$$T' \otimes (V(H_2)|V(G_2), \bullet), \quad (r[T'], V(H_2)) \oplus (\bullet, \emptyset|V(G_2)), \quad T' \otimes (\bullet, \emptyset|V(G_2)) \oplus (\bullet, \emptyset|V(H_2))$$

are lc3-expressions for the complete lc3-composition of $G_1, \ldots, H_3$. Finally, let G_3 be edgeless, and let T be an lc3-expression for $G_1 \otimes G_2$ that groups into $V(G_1)$ and $V(G_2)$ where $V(G_1)$ is the left partition set. Then,

$$T \circ (V(H_1), V(G_3)|V(H_3), V(H_2), d[123]), \quad (V(H_1)|\emptyset, l[T], V(G_3)|V(H_3)) \oplus (\emptyset|V(H_2), \bullet)$$

for $d \in \{l, r\}$ are lc3-expressions for the complete lc3-composition of $G_1, \ldots, H_3$. The remaining cases are symmetric. This then completes the proof of the lemma. ∎

A linear clique-width expression that uses at most three labels is called *linear 3-expression*. For label c and vertices $x_1, \ldots, x_r$, a clique-width expression $c(x_1) \cdots c(x_r)$ is shortly written as $c(\{x_1, \ldots, x_r\})$. Note that the ordering of the vertices is not important. We show that every lc3-graph has linear clique-width at most 3.

Lemma 11. *For every lc3-graph there exists a linear 3-expression.*

Proof Let G be an lc3-graph with lc3-expression T. We inductively define a linear 3-expression for G. After every construction step, the linear 3-expression will have at most two assigned labels and the two label classes exactly correspond to the two vertex partition classes of the lc3-expression. For the construction, we distinguish different cases. In the following, let T' be an lc3-expression with vertex partition (B, C), let a' be a linear 3-expression that defines $G(T')$ such that the constructed graph has at most two assigned labels and the two label classes exactly correspond to (B, C). Let c_1 and c_2 be the labels that correspond to the vertices in B and C, respectively. If C is empty, c_2 is one of the two non-assigned labels. Let c_3 be the third label. Finally, let A, A_1, A_2, A_3, A_4 be sets of vertices, $d \in \{l, r\}$ and p an appropriate sequence of numbers. We first consider the lc3-expression operations defined in (d1) and (d4).

- $T = (A)$: $G(T)$ is an edgeless graph on vertex set A. Then, $a =_{\text{def}} 1(A)$ is a linear 3-expression that defines G and respects the vertex partition defined by T.
- $T = T' \odot (A_1|A_2, \bullet)$: If $\odot = \otimes$, let $a =_{\text{def}} a'\ c_3(A_2)\ \eta_{c_2,c_3}\ c_3(A_1)\ \eta_{c_1,c_3}\ \rho_{c_3 \to c_1}$.

  If $\odot = \oplus$, we define a similar expression, that does not contain operation η_{c_1,c_3}. It is clear in both cases that a defines $G(T)$, that label c_3 is not assigned to any vertex in the graph defined by a and that c_1 and c_2 determine exactly the vertex partition $((B \cup A_1 \cup A_2), C)$.
- The case $T = T' \odot (\bullet, A_1|A_2)$ is purely symmetric to the previous case.
- $T = T' \circ (A_1, A_2|A_3, A_4, d[p])$: We give a sample expression for $d = l$ and $p = 123$:

  $a =_{\text{def}} a'\ c_3(A_2)\ \eta_{c_1,c_3}\ c_1(A_1)\ c_3(A_3)\ \eta_{c_1,c_2}\ c_2(A_4)\ \eta_{c_2,c_3}\ \rho_{c_3 \to c_1}$.

  The other cases are obtained analogously. In case of $d = r$, the relabel operation would be $\rho_{c_3 \to c_2}$.

For the remaining operations, we can reduce to the cases above. Only the relabel operation is replaced by $\rho_{c_2 \to c_1}$. Thus, we have shown that there is a linear 3-expression for G. ∎

Similar to linear 3-expressions, a *linear k-expression* for $k \geq 1$ is a linear clique-width expression that uses at most k labels. Before showing the counterpart of Lemma 11, which gives the main result of this section, we prove a normalisation result for linear clique-width expressions. This gives an intuitive notion of "useless operation" in linear k-expressions. For a linear clique-width expression $t = t_1 \cdots t_r$, we denote the graph that is constructed from $t_1 \cdots t_i$ as $G[t_1 \cdots t_i]$. We say that vertices "belong to the same label class in $G[t_1 \cdots t_i]$" if they have the same label in $G[t_1 \cdots t_i]$.

Lemma 12. *Let $k \geq 1$ and let G be a graph that has a linear k-expression. Then, G has a linear k-expression $a = a_1 \cdots a_r$ such that the following holds for all join and relabel (η and ρ) operations a_i in a:*

(1) $G[a_1 \cdots a_i]$ *does not contain an isolated vertex*

(2) $G[a_1 \cdots a_{i-1}]$ *contains vertices of the two label classes involved in* a_i *.*

Proof We show the two properties separately. Let $b = b_1 \cdots b_s$ be a linear k-expression for G. Let b_i be a join or relabel operation and suppose that $G[b_1 \cdots b_i]$ contains an isolated vertex, say x. Let c be the label of x in $G[b_1 \cdots b_i]$. If b_i is a join operation then c is not one of the two join labels. We obtain $b' = b'_1 \cdots b'_s$ from b by deleting the vertex creation operation for x in b and adding the operation $c(x)$ right after operation b_i. Then, $G[b'_1 \cdots b'_i] = G[b_1 \cdots b_i]$. Iterated application of this operation shows existence of a linear clique-width k-expression having the first property.

For the second property, let $d = d_1 \cdots d_t$ be a linear k-expression that has the first property. Let $d_i = \eta_{c,c'}$ be a join operation. If $G[d_1 \cdots d_i]$ contains no vertex with label c or c' then d_i adds no edge to $G[d_1 \cdots d_{i-1}]$, which means $G[d_1 \cdots d_{i-1}] = G[d_1 \cdots d_i]$. We obtain d' from d by deleting operation d_i. Now, let $d_i = \rho_{c \to c'}$ be a relabel operation, and suppose that one of the two label classes is empty in $G[d_1 \cdots d_{i-1}]$. If the label class of c is empty then we obtain d' from d by just deleting operation d_i. If the class of c is non-empty but the class of c' is empty then we obtain d' from d by first exchanging c and c' in all operations $d_{i+1}, \ldots, d_t$ and then deleting d_i. It is not difficult to show that $G[d_1 \cdots d_j]$ and $G[d'_1 \cdots d'_{j-1}]$ correspond to each other for every $i+1 \leq j \leq t$ with the exception that the label classes of c and c' are exchanged. Repeated application of the modification completes the proof. Note that the expressions after the execution of the second modification still have the first property. ▪

Theorem 13. *A graph has linear clique-width at most 3 if and only if it is an lc3-graph.*

Proof One implication follows from Lemma 11. For the converse, let G be a non-empty graph of linear clique-width at most 3. We show that G is an lc3-graph by giving an lc3-expression for G. Let $a = a_1 \cdots a_r$ be a linear 3-expression for G that has the properties of Lemma 12. In particular, a_1 is a vertex creation operation. We prove the claim by induction over the number of relabel operations. The following observations are crucial for the proof. Given a labelled graph G', execute only vertex creation and join operations on G'. Then, only the last join operation between two labels has to be considered, and no vertex that is added to G' is adjacent to another vertex assigned the same label. Let $G_i =_{\text{def}} G[a_1 \cdots a_i]$ for $1 \leq i \leq r$. If G is an edgeless graph, which means that a contains no join operation, let the lc3-expression T be defined as (V). Since $G = G(T)$, we conclude the statement. So, let a have a join operation. Without loss of generality, we can assume that a contains a relabel operation right after the last join operation that involves one of the labels of the relabel operation.

Let a_i be the first relabel operation in a. Consider G_{i-1}. Since a_i is the first relabel operation in a, every label class induces an edgeless graph in G_{i-1}. Let A_1, A_2, A_3 be the sets of vertices corresponding to the three different label classes. For defining the lc3-expression, we consider different cases. If A_3 is empty then all vertices in A_1 are adjacent to all vertices in A_2 and G_i has only one label class; we let $T_1 =_{\text{def}} (A_1) \otimes (A_2|\emptyset, \bullet)$. So, let A_3 be non-empty. Let D_1 be the set of vertices in A_1 having at least one neighbour in both A_2 and A_3. Similarly, let D_2 be the set of vertices in A_2 having at least one neighbour in both A_1 and A_3, and let D_3 be the set of vertices in A_3 having at least one neighbour in both A_1 and A_2. Note that one of the sets D_1, D_2, D_3 must be non-empty. If exactly one of these sets is non-empty, G_i is the incomplete lc3-composition of at most four edgeless graphs (involving the graphs G_1, G_2, G_3, H_1

of Figure 4). If at least two of the sets D_1, D_2, D_3 are non-empty then all three sets are non-empty. Then, G_i is the complete lc3-composition of six edgeless graphs. Let A_3 be the set of vertices that corresponds to the label that is not involved in the relabel operation a_i. According to Lemma 10, there is an lc3-expression for G_i that groups the vertices of G_i into $A_1 \cup A_2$ and A_3. This completes the proof of the base case.

Now, let $a_{j'}$ be the tth relabel operation in a for $t \geq 2$ and let a_j be the relabel operation in a preceding $a_{j'}$. By induction hypothesis, there is an lc3-expression T_{t-1} that defines G_j in such a way that vertex partition (B, C) for $G(T_{t-1})$ corresponds to the two label classes in G_j. This particularly means that if two labels are not assigned in G_j then C can be assumed empty. We show that an lc3-expression for $G_{j'}$ exists, with vertex partition that corresponds to the label classes in $G_{j'}$. The proof is done by distinguishing several cases. Let A_1, A_2, A_3 be the sets of vertices of $G_{j'-1}$ corresponding to the three labels, and let $B \subseteq A_1$ and $C \subseteq A_2$. Let $A'_i =_{\text{def}} A_i \setminus (B \cup C)$ for $i \in \{1, 2, 3\}$. The vertices in $A'_1 \cup A'_2 \cup A'_3$ are added after operation a_j in a, and $A_3 = A'_3$. Note that A'_1, A'_2, A'_3 are independent sets in $G_{j'-1}$. We consider two basic cases distinguishing whether there is a join operation involving the labels of A_1 and A_2 between a_j and $a_{j'}$ or not. First, let there be such a join operation. If this is the only type of join operations between a_j and $a_{j'}$ then A_3 is empty (otherwise the vertices in A_3 would be isolated) and $G_{j'}$ is a join of the subgraphs induced by A_1 and A_2 (since there are no isolated vertices). According to Lemma 8, $G_{j'}[B]$ is an lc3-graph, and according to Lemma 9, $G_{j'}[C]$ is a simple cograph, and by construction, A'_1 and A'_2 induce edgeless graphs. Hence, $G_{j'}$ is the complete lc3-composition of the subgraphs of $G_{j'}$ induced by B, C, A'_1, A'_2, which is an lc3-graph due to Lemma 10. Furthermore, there exists an lc3-expression for $G_{j'}$ that groups the vertices into the sets A_1 and A_2. This completes the proof of this case. Now, let there be another type of join operations between a_j and $a_{j'}$ involving vertices of A_3. If there is exactly one type of join operations involving vertices of A_3 then $G_{j'}$ is an incomplete lc3-composition. If there are two types of join operations then $G_{j'}$ is a complete lc3-composition. Similar to the first case, we conclude that $G_{j'}$ is an lc3-graph by applying Lemmata 8, 9 and 10.

Now, we consider the second basic case, where there is no join operation involving the vertices in A_1 and A_2 between a_j and $a_{j'}$. Let $a_{j'}$ involve the label assigned to the vertices of A_3. Without loss of generality assume that $a_{j'}$ changes the label of A_1 to the label of A_3 (or $a_{j'}$ changes the label of A_3 to the label of A_1); the other case follows similarly. All vertices of A'_1 and A'_2 have a neighbour in A_3, and there is a join between the vertices of A_3 and the vertices of A_1 or A_2 (otherwise a vertex in A_3 would be isolated). Let D_3 be the set of vertices of A_3 with at least one neighbour in the both A_1 and A_2. Then, one of the two lc3-expressions defines $G_{j'}$, with vertex partition $(A_1 \cup A_3, A_2)$:

- $T_t =_{\text{def}} T_{t-1} \oplus (A'_1|\emptyset, \bullet) \oplus (\bullet, A'_2|\emptyset) \otimes ((A_3 \setminus D_3)|D_3, \bullet)$

  $T_t =_{\text{def}} T_{t-1} \oplus (A'_1|\emptyset, \bullet) \oplus (\bullet, A'_2|\emptyset) \otimes (\emptyset|D_3, \bullet) \oplus (\emptyset|(A_3 \setminus D_3), \bullet)\,.$

Now, let $a_{j'}$ involve only the labels of A_1 and A_2. Then, depending on the case, the following lc3-expression defines $G_{j'}$ in the desired way:

- $T_t =_{\text{def}} (d[T_{t-1}], (A_3 \setminus D_3)) \oplus (\bullet, \emptyset|D_3)$

where $d \in \{l, r\}$. For completing the proof we have to consider vertices that are added after the last relabel operation. Only vertex creation operations can appear. We then obtain an lc3-expression for G by adding these last vertices, that are isolated in G, attaching an $\oplus(A|\emptyset, \bullet)$ operation. This completes the proof. ∎

As two last results in this section, we show that graphs of linear clique-width at most 3 belong to well-known graph classes.

Proposition 14. *Lc3-graphs are cocomparability graphs.*

Proof We show the statement by induction over the definition of lc3-expressions. Let $G = (V, E)$ be an lc3-graph with lc3-expression T. We show that there is a cocomparability ordering for G that respects the vertex partition for $G(T)$. If $T = (A)$ then $G(T)$ is an edgeless graph, and every vertex ordering for G is a cocomparability ordering and respects the vertex partition $(A, \emptyset)$. Now, let T be more complex. For the rest of the proof, let T' be an lc3-expression with vertex partition (B, C), let σ' be a cocomparability ordering for $G(T')$ that respects partition (B, C), which means that the vertices in B appear consecutively and the vertices in C appear consecutively in σ'. Without loss of generality, we can assume that the vertices in B appear to the left of the vertices in C in σ'. Let A, A_1, A_2, A_3, A_4 be sets of vertices, $d \in \{l, r\}$ and p an appropriate sequence of numbers. We distinguish different cases.

- $T = T' \odot (A_1|A_2, \bullet)$ for $\odot \in \{\oplus, \otimes\}$. We obtain σ from σ' by adding the vertices in A_1 to the left of the vertices in B and the vertices in A_2 between the vertices in B and in C. Then, σ is a cocomparability ordering for G and the vertices in $B \cup A_1 \cup A_2$ appear consecutively.
- The case $T = T' \odot (\bullet, A_1|A_2)$ is similar to the previous case where A_1 is added to the right of the vertices in C.
- $T = (l[T'], A)$. The vertices in A are adjacent to only the vertices in C. We obtain σ from σ' by adding the vertices in A to the right of the vertices in C. Then, σ is a cocomparability ordering for G that respects the partition $(B \cup C, A)$.
- The case $T = (r[T'], A)$ is similar to the previous case; we put A to the left of B.
- $T = T' \circ (A_1, A_2|A_3, A_4, l[12])$. The vertices in B and A_2 are adjacent and the vertices in $B \cup A_1$ and C are adjacent. We obtain σ by placing the vertices in the following order: A_3, A_2, B, A_1, C, A_4, and the vertices in B and C appear in order determined by σ'. Then, σ is a cocomparability ordering for G and respects the vertex partition $((B \cup A_1 \cup A_2 \cup A_3), (C \cup A_4))$.
- $T = T' \circ (A_1, A_2|A_3, A_4, r[12])$. We place the vertices in order A_4, A_3, A_2, C, B, A_1.
- The cases $T = T' \circ (A_1, A_2|A_3, A_4, d[32])$ are symmetric to the previous ones, where the roles of B and C are exchanged.

The remaining cases are $T = T' \circ (A_1, A_2|A_3, A_4, d[p])$ where $p \in \{123, 132, 312, 321\}$ and $(A_1|A_2, d[T'], A_3|A_4)$. Then, $G(T)$ is a complete lc3-composition. Let $G_1, G_2, G_3, H_1, H_2, H_3$ be graphs with their meaning as in Figure 4. Suppose cocomparability orderings are given. We define two vertex orderings. The vertices of the partition graphs appear consecutively and in order defined by the given cocomparability orderings. The order of the partition graphs is as follows:

$$H_3, G_3, H_2, G_2, G_1, H_1 \quad \text{and} \quad H_2, G_2, H_3, G_3, G_1, H_1\,.$$

It is easy to check that all three vertex orderings actually define cocomparability orderings for the complete lc3-composition. Furthermore, for every $i \in \{1, 2, 3\}$, there is a cocomparability

ordering such that the vertices of $G_i \oplus H_i$ appear consecutively at an end of the ordering. Depending on p, the pairs $(B, A_1), (C, A_4), (A_2, A_3)$ are matched to $(G_1, H_1), (G_2, H_2), (G_3, H_3)$, and depending on d and the particular case, one of the vertex orderings is chosen to achieve the correct vertex partition. This completes the proof. ▪

A chordless cycle of length at least 5 is called *hole* and the complement of a hole is called *anti-hole*. We show that graphs of linear clique-width at most 3 do not contain holes or anti-holes as induced subgraph.

Proposition 15. *Lc3-graphs are weakly-chordal graphs.*

Proof With the fact that cocomparability graphs are hole-free and Proposition 14, we already know that lc3-graphs are hole-free. It remains to show that lc3-graphs contain no anti-holes as induced subgraphs. We show that anti-holes have groupwidth at least 4. Let $k \geq 5$ and consider $\overline{C_k}$. Since C_5 and $\overline{C_5}$ are isomorphic, we can assume $k \geq 6$. Let $\beta = \langle y_1, \ldots, y_k \rangle$ be a layout for $\overline{C_k}$. We distinguish two basic cases. First, let y_{k-1} and y_k be non-adjacent in $\overline{C_k}$, i.e., y_{k-1} and y_k are adjacent in C_k. Then, $L_\beta(y_{k-1})$ induces a path in C_k and has three groups: vertices that are adjacent to both y_{k-1} and y_k, a vertex that is non-adjacent to y_{k-1} and adjacent to y_k, a vertex that is non-adjacent to y_k and adjacent to y_{k-1}. Thus, $\nu_{\overline{C_k}}(L_\beta(y_{k-1})) = 3$. For the value of $\mathrm{ad}_\beta(y_{k-1})$, observe that y_{k-1} is in a group of two vertices in $L_\beta[y_{k-1}]$ and that the other vertex is a neighbour of y_{k-1}. Hence, $\mathrm{ad}_\beta(y_{k-1}) = 1$, and $\mathrm{gw}(\overline{C_k}, \beta) \geq 4$.

Now, let y_{k-1} and y_k be adjacent in $\overline{C_k}$. Then, both y_{k-1} and y_k have exactly two non-neighbours in $L_\beta(y_{k-1})$, and at most one vertex can be a common non-neighbour. We distinguish different cases. Let y_{k-1} and y_k have a common non-neighbour. Then, $L_\beta(y_{k-1})$ has the following groups (note that this requires $k \geq 6$): vertices adjacent to y_{k-1} and y_k, vertices non-adjacent to y_{k-1} and y_k, vertices adjacent to y_{k-1} but not to y_k, vertices adjacent to y_k but not to y_{k-1}. Hence, $\nu_{\overline{C_k}}(L_\beta(y_{k-1})) = 4$. Let y_{k-1} and y_k not have a common non-neighbour in $L_\beta(y_{k-1})$. Let $k \geq 7$. Then, $L_\beta(y_{k-1})$ has three groups. Furthermore, y_{k-1} is not single vertex in its group in $L_\beta[y_{k-1}]$ and (at least) one of the two non-neighbours of y_{k-1} is non-adjacent to a neighbour of y_{k-1} in $L_\beta(y_{k-1})$. Thus, $\mathrm{ad}_\beta(y_{k-1}) = 1$, and $\mathrm{gw}(\overline{C_k}, \beta) \geq 4$. Finally, let $k = 6$. Then, $L_\beta(y_{k-1})$ induces a C_4 in $\overline{C_k}$ (since y_{k-1} and y_k have no common non-neighbour). Similar to the previous cases, $L_\beta(y_{k-2})$ has three groups and $\mathrm{ad}_\beta(y_{k-2}) = 1$ so that $\mathrm{gw}(\overline{C_k}, \beta) \geq 4$. Hence, $\mathrm{gw}(\overline{C_k}) \geq 4$ for $k \geq 6$. Applying Theorems 1 and 13 and the fact that the linear clique-width of a graph is not smaller than the linear clique-width of any of its induced subgraphs, it follows that lc3-graphs have no anti-holes as induced subgraphs, so they are weakly-chordal. ▪

Note that holes and anti-holes have linear 4-expressions. Together with the result of Proposition 15, it follows that the linear clique-width of holes and anti-holes is exactly 4. It is an interesting observation that no cycle has linear clique-width 3. C_3 and C_4 have linear clique-width 2, but their complements have linear clique-width 1 and 3, respectively.

7 Recognition of graphs of linear clique-width at most 3

This section is partitioned into two parts. We first give the recognition algorithm for lc3-graphs and then give a linear-time implementation of the algorithm. For a brief outline, the algorithm recursively constructs an lc3-expression for the input graph, if possible. Here, we distinguish the cases when additionally a vertex partition is associated with the graph or no further requirement is added. The main algorithm is given in Figure 8, that calls a slight modification of the algorithm

given in Figure 6 as subroutine. This subroutine simplifies input graphs by identifying "difficult" induced subgraphs. Properties of this algorithm are stated in Lemma 18. Before, however, we consider an algorithm for deciding a special question for lc3-compositions.

Lemma 16. *There is a linear-time algorithm that, on input a graph G and a vertex partition (B, C) for G, where B and C are non-empty, checks whether G is the complete or incomplete lc3-composition of graphs $G_1, G_2, G_3, H_1, H_2, H_3$ where at least two of the graphs are non-empty, H_1, H_2, H_3 are edgeless graphs and G_1, G_2, G_3 are an edgeless graph, a simple cograph and an arbitrary graph and there is $i \in \{1, 2, 3\}$ such that $B = V(G_i) \cup V(H_i)$ or $C = V(G_i) \cup V(H_i)$. In the positive case, the algorithm outputs an appropriate decomposition of G.*

Proof The main task is to find a partition of G into at most six induced subgraphs that can be mapped to the graphs in Figure 4. The algorithm is not difficult but has to check a number of cases. First, determine the groups in B and in C. If G is an lc3-composition of the requested kind, B or C can have at most two groups (one being $V(G_i)$ and the other being $V(H_i)$). So, if B and C have at least three groups, the algorithm rejects, which is correct. Checking the three cases in Figure 4, we conclude: if B has exactly two groups then the neighbourhood of the one group (with respect to C) is properly contained in the neighbourhood of the other group; analogously for C. If this is not possible, the algorithm rejects, which is correct. Note that, if B and C have exactly two groups, the inclusion property holds either for none or for both sets.

Now, assume that the algorithm has not yet rejected. As a first case, let one set have exactly one group and the other two groups. Without loss of generality, let C have two groups, C_1 and C_2, and let the neighbours of B be in C_1. If G is an lc3-composition where C corresponds to some $V(G_i) \cup V(H_i)$ then C_1 and C_2 correspond to respectively $V(G_i)$ and $V(H_i)$, and C_2 is an independent set in G and $G[C] = G[C_1] \oplus G[C_2]$. Checking all situations, this means that $G[B]$ or $G[C_1]$ is a simple cograph. So, accept if $G[B]$ or $G[C_1]$ is a simple cograph, and output $G[B], G[C_1], G[C_2]$ as the decomposition. Now, we assume that B corresponds to some $V(G_i) \cup V(H_i)$. If both G_i and H_i are non-empty, they have the same neighbourhood in C, so that we can restrict to the case that G_i is non-empty and H_i is empty. Remember that if G_i is a simple cograph then $G_i \oplus H_i$ is a simple cograph, too. To decide whether G is an lc3-composition, we have to check several cases, that are obtained from the complete and incomplete lc3-composition in Figure 4 by deleting an H-vertex. The obtained situations are depicted in Figure 5. The full (black) circles represent an edgeless graph, a simple cograph and an arbitrary graph, and the empty (white) circles represent edgeless graphs. The upper circle represents G_i (which is supposed to be $G[B]$), the second level represents $G[C_1]$ and the third level represents $G[C_2]$. Note that a sixth case is missing; this case (deleting H_2 for a complete lc3-composition) cannot happen since it allows only one group in C. Given the sets C_1 and C_2, each case can be checked in linear time. Note that the third level of the first case is a complete bipartite graph, for which the two colour classes are uniquely defined. The algorithm accepts if one of the five situations is suitable for G, and outputs the corresponding decomposition. If $G[B]$ is edgeless then $G[B]$ can also correspond to $V(H_i)$. We obtain the different situations by deleting full-circle vertices in Figure 4: two cases are not possible, since C_1 then must be empty, three cases turn out to be subcases of situations in Figure 5 and have already been checked (namely situations 4 and 5), and the last situation, that is not covered by the previous cases, requires $G = (G[B] \otimes G[C_1]) \oplus G[C_2]$ where $G[C_1]$ and $G[C_2]$ are a simple cograph and an arbitrary graph. This case is the only case that has to be considered and can be checked in linear time.

As a second case, let B and C have exactly one group. Then, $G = G[B] \otimes G[C]$ or $G = G[B] \oplus G[C]$. If $G[B]$ or $G[C]$ is a simple cograph, the algorithm accepts; then, G is a complete

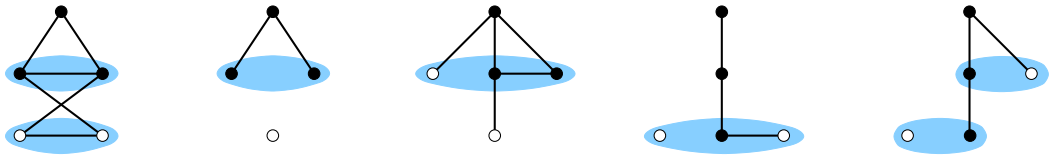


Figure 5: Different situations in a special case in the proof of Lemma 16. The first level corresponds to set B, the second and third level correspond to sets C_1 and C_2, respectively.

or incomplete lc3-composition with four empty graphs where the two non-empty graphs go into G_2 and G_3 of Figure 4. Now, let both $G[B]$ and $G[C]$ not be simple cographs. Similar to the argumentation in the previous case, we can restrict to the case that one of the two graphs corresponds to $V(G_i)$ with $V(H_i)$ empty. The situation becomes similar to the previous case where C_1 or C_2 is empty. Listing all possible situations (for instance by deleting the second or third level in Figure 5), we see that G cannot be an lc3-composition of an edgeless graph, a simple cograph, an arbitrary graph and three edgeless graphs in this case. This follows from the closure of simple cographs under join and union with edgeless graphs. Hence, if $G[B]$ and $G[C]$ are not simple cographs, the algorithm rejects.

As the third case, let B and C have exactly two groups. Let B_1 and B_2 be the groups in B and let the neighbours of B_2 in C be neighbours also of B_1. Assume that B corresponds to some $V(G_i) \cup V(H_i)$; as discussed before this means $B_1 = V(G_i)$ and $B_2 = V(H_i)$, in particular B_2 is an independent set and there is no edge between vertices in B_1 and in B_2. Similar to the first case, the algorithm checks whether C admits a partition. If this is successful the algorithm accepts, otherwise it retries with the roles of B and C exchanged. If the tests for both sets are not successful the algorithm rejects. This completes the proof. ∎

Given a graph G and a vertex partition (B, C) for G, we want to decide whether there is an lc3-expression for G with vertex partition (B, C) or (C, B). Consider Algorithm SIMPLIFY in Figure 6. This algorithm does not decide the question in all cases; however, in the cases when it cannot find a proper answer it returns an induced subgraph G' of G that is an lc3-graph if and only if G is an lc3-graph. In particular, the question about the existence of an lc3-expression for G' is not restricted by a particular vertex partition. The question of conditional number 6 of SIMPLIFY is exactly the question that is answered by Lemma 16. Besides computing an answer, SIMPLIFY also outputs lc3-expression operations. For simplicity, we do not explicitly state which operation is output in which case but combine similar cases. A new command is the "turn" command. We use it to indicate that the vertices in a partition have to change side; this becomes more clear in the proof of Lemma 18. For a graph G and a set A of vertices of G, we say that A is *false-twin-free* if A contains no pair of vertices that are false twins in G.

Lemma 17. *Let $G = (V, E)$ be a graph and let (B, C) be a vertex partition for G. Let B and C be false-twin-free. Then, at the beginning of every execution of the* **while** *loop of* SIMPLIFY*, the two current partition sets are false-twin-free. Furthermore, a return graph has no false twin vertices.*

Proof The statement is clearly true for the first execution. We consider the definitions in conditionals number (3–5). In conditional number 3, the chosen vertex u is adjacent to all (other) vertices in a partition set or to none. So, the new partition sets for $G-u$ are false-twin-free. Similarly for conditionals number 4 and 5. Now, if a graph is the return result (conditionals number 2 and 6), it is induced by vertices from only one partition set and it is a module in

Algorithm SIMPLIFY
Input a graph G and a vertex partition (B, C) for G
Result an answer ACCEPT or REJECT or a graph G'

begin
while no return **do**

1 **if** G is edgeless **then** **output** "(B, C)"; **return** ACCEPT

2 **else-if** $B = \emptyset$ or $C = \emptyset$ **then** **output** "turn+$(l[\cdot], \emptyset)$" or "$(l[\cdot], \emptyset)$"; **return** G

3 **else-if** G has a vertex u such that
$N_G(u) = \emptyset$ **or** $N_G(u) = B$ **or** $N_G(u) = C$ **or**
$N_G[u] = V(G)$ **or** $N_G[u] = B$ **or** $N_G[u] = C$ **then**
output "$\oplus(\emptyset|\{u\}, \bullet)$" or "$\oplus(\bullet, \emptyset|\{u\})$" or "$\oplus(\{u\}|\emptyset, \bullet)$" or "$\oplus(\bullet, \{u\}|\emptyset)$"
"$\otimes(\emptyset|\{u\}, \bullet)$" or "$\otimes(\bullet, \emptyset|\{u\})$" or "$\otimes(\{u\}|\emptyset, \bullet)$" or "$\otimes(\bullet, \{u\}|\emptyset)$";
set $G :=$ $G-u$ and $(B, C) :=$ $(B \setminus \{u\}, C \setminus \{u\})$

4 **else-if** there is a pair u, v of non-adjacent vertices where v is almost-universal such that
$u, v \in B$ **and** $N_G(u) = B \setminus \{u, v\}$ **or**
$u, v \in C$ **and** $N_G(u) = C \setminus \{u, v\}$ **then**
output "$\otimes(\{u\}|\{v\}, \bullet)$" or "$\otimes(\bullet, \{u\}|\{v\})$";
set $G :=$ $G \setminus \{u, v\}$ and $(B, C) :=$ $(B \setminus \{u, v\}, C \setminus \{u, v\})$

5 **else-if** $B = \{u\}$ **or** $C = \{u\}$ **and**
$|V(G) \setminus N_G[u]| \neq 2$ **or** $V(G) \setminus N_G[u] = \{x, z\}$ and the sets
$N_G(x) \setminus \{z\}$ and $N_G(z) \setminus \{x\}$ can be ordered by inclusion **then**
output "$(l[\cdot], \{u\})$" or "turn+$(l[\cdot], \{u\})$";
set $G :=$ $G-u$ and $(B, C) :=$ $(V(G) \setminus N_G[u], N_G(u))$

6 **else-if** G is the result of an lc3-composition of at least two non-empty graphs
that respects the given vertex partition **then**
output an lc3-composition scheme and, if necessary, "turn";
if all partition graphs are simple cographs **then** **return** ACCEPT
else **return** the partition graph that is not a simple cograph **end if**

7 **else** **return** REJECT **end if**

end while
end.

Figure 6: The simplification procedure.

the bigger graph. Hence, the return graph has no false twin vertices. The lemma follows by induction. ∎

Note that it is not true in general that a graph during the execution of SIMPLIFY has no false twins. Particularly conditional number 5 can create a pair of false twins, but they are in different partition sets.

Lemma 18. *Let $G = (V, E)$ be a graph and let (B, C) be a vertex partition for G. Let B and C be false-twin-free.*

(1) If SIMPLIFY *applied to G and (B, C) returns* ACCEPT *then G is an lc3-graph and there is an lc3-expression for G with vertex partition (B, C) or (C, B).*

(2) If SIMPLIFY *applied to G and (B, C) returns* REJECT *then G is not an lc3-graph or there is no lc3-expression for G with vertex partition (B, C) or (C, B).*

(3) If SIMPLIFY *applied to G and (B, C) returns a graph G' then there is an lc3-expression for G with vertex partition (B, C) or (C, B) if and only if G' is an lc3-graph. Furthermore, G' is a module of G and a subgraph of G that is induced by a subset of B or C.*

Proof We prove the lemma by induction over the number of **while** loop executions. We show the three statements simultaneously by considering the two situations: *(1)* the algorithm returns ACCEPT or a graph and *(2)* the input graph is an lc3-graph with lc3-expression T with vertex partition (B, C) or (C, B). Since we start with false-twin-free partition sets, we can assume Lemma 17 throughout the proof.

Let SIMPLIFY return ACCEPT or a graph. We show the existence of an lc3-expression (if possible). We also show that the "turn" command can be used to know whether the constructed lc3-expression corresponds to the given vertex partition or to its reverse. The "turn" command can be 'active' or 'inactive'. If G is edgeless then $(B) \oplus (\bullet, C|\emptyset)$ is an lc3-expression with vertex partition (B, C). The "turn" command is set 'inactive'. If B or C is empty and G is an lc3-graph then $(l[T], \emptyset)$ for T an (arbitrary) lc3-expression for G is an lc3-expression for G with vertex partition $(V(G), \emptyset)$. If G is not an lc3-graph then there is no lc3-expression for G. Thus, if B or C is empty, G is an lc3-graph with vertex partition (B, C) or (C, B) if and only if G is an lc3-graph. Similar for conditional number 6: if a graph is returned, G is an lc3-graph with vertex partition (B, C) or (C, B) if and only if the returned graph is an lc3-graph due to Lemmata 10 and 8. For the cases of conditionals number 3 and 4, remember that every induced subgraph of G is an lc3-graph and can be associated with a restriction of (B, C) or (C, B) due to Lemma 8. Applying the induction hypothesis, we conclude these two cases. For an lc3-expression, one of the output operations can be appended. The "turn" command remains unchanged in its state. For conditional number 5, let SIMPLIFY accept $G-u$ with vertex partition $(V(G) \setminus N_G[u], N_G(u))$ or return an lc3-graph G'. By induction hypothesis, there is an lc3-expression T' for $G-u$ with vertex partition $(V(G) \setminus N_G[u], N_G(u))$ or its reverse. Then, $(l[T], \{u\})$ or $(r[T], \{u\})$ is an lc3-expression for G with vertex partition (B, C) or (C, B). The choice of l or r depends on whether the "turn" command is 'active' or 'inactive'. If the vertex partition is (B, C) then the "turn" command becomes 'inactive', otherwise 'active'. If G' is not an lc3-graph, then G is not an lc3-graph either. This completes the first part of the proof.

For the converse, let G be an lc3-graph and let T be an lc3-expression for G with vertex partition (B, C) or (C, B). We show that the algorithm returns ACCEPT or a graph G'. Since G' is an induced subgraph of G, G' is an lc3-graph if G is an lc3-graph due to Lemma 8. If G is edgeless then SIMPLIFY returns ACCEPT (conditional number 1). If B or C is empty then SIMPLIFY returns a graph (conditional number 2). So, assume that B and C are non-empty. If the condition of conditional number 3 or 4 is positive, then Lemma 8 shows that the obtained lc3-graph can be associated with the obtained vertex partition or its reverse, and SIMPLIFY then returns ACCEPT or a graph due to induction hypothesis. Conditionals number 5 and 6 need more arguments. We assume that the execution reaches conditional number 5. Consider the last operation of T. Since the procedure execution passed the first four conditionals, the last operation cannot be of the forms (A) or $\odot(A_1|A_2, \bullet)$ or $\odot(\bullet, A_1|A_2)$, where $\odot \in \{\oplus, \otimes\}$. It is important to note that an operation $\oplus(A_1|A_2, \bullet)$ can be partitioned into two operations, $\oplus(A_1|\emptyset, \bullet) \oplus (\emptyset|A_2, \bullet)$. Let the last operation be of the form $(d[T'], A)$ for T' an lc3-expression and $d \in \{l, r\}$. Since the partition sets are false-twin-free, A contains at most one vertex, that is, according to assumption about B and C, A contains exactly one vertex, say u. Let (B', C') be the vertex partition for $G(T')$. Then, $N_G(u) = B'$ or $N_G(u) = C'$, which means that $G-u$ is an lc3-graph and has an lc3-expression with vertex partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse. By induction hypothesis, SIMPLIFY returns ACCEPT or a graph when applied to $G-u$ and partition $(N_G(u), V(G) \setminus N_G[u])$. Suppose that G fails to satisfy the condition about the vertices in $V(G) \setminus N_G[u]$. Then, $V(G) \setminus N_G[u]$ and $N_G(u)$ contain four vertices satisfying the property of the second statement of Lemma 9. This is a contradiction, and we complete this

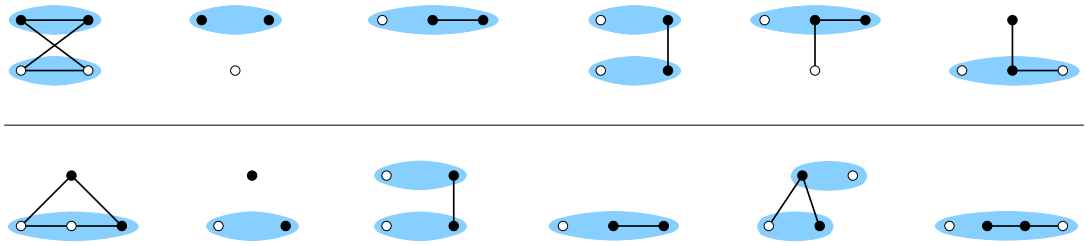


Figure 7: Subcases of the general lc3-composition, considered in the proof of Lemma 18.

case.

As a second case, let the last operation of T be of the form $\circ(A_1, A_2|A_3, A_4, d[p])$ or $(A_1|A_2, d[T'], A_3|A_4)$, which means that G is the result of an lc3-composition. If B and C contain at least two vertices then the execution continues with conditional number 6 and returns ACCEPT or a graph due to Lemma 10. So, assume that either B or C contains exactly one vertex; let this vertex be u. The algorithm execution would try conditional number 5. We know that $G-u$ is the result of an lc3-composition, but we have to show that the new vertex partition works. This means that we have to show that there is an lc3-expression with partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse for $G-u$. Let $G_1, G_2, G_3, H_1, H_2, H_3$ be the composition graphs in the sense of Figure 4, whose lc3-composition yields G. Without loss of generality, we can assume that there is $i \in \{1, 2, 3\}$ such that $\{u\} = V(G_i) \cup V(H_i)$, which means that there is $i \in \{1, 2, 3\}$ such that $\{u\} = V(G_i)$ or $\{u\} = V(H_i)$ and the other graph is empty. We consider the different cases of lc3-composition without G_i or H_i with respect to the vertex partition $(N_G(u), V(G) \setminus N_G[u])$. In Figure 7, we find all twelve situations depending on whether u is in G_i or in H_i and whether the lc3-composition is complete or incomplete. The full (black) circles represent a simple cograph and an arbitrary graph, and the empty (white) circles represent edgeless graphs. The upper level composition graphs constitute $G[N_G(u)]$ and the lower level composition graphs constitute $G \setminus N_G[u]$. By checking every situation, we finally see that every case except one can be obtained from an appropriate lc3-composition respecting the new vertex partition. The most interesting situation probably is the first case in the second row. This situation becomes a special case of the last case in the first row, since an edgeless graph and the full circle graph in the lower level can be merged into a single graph. This can be done, since their neighbourhoods with respect to the two other composition graphs are equal and simple cographs are closed under disjoint union with edgeless graphs. The situations with only one level cannot happen since u then would be a universal or an isolated vertex. In the 'good' cases we apply Lemma 10 to show that $G-u$ is an lc3-graph which can be associated with the computed vertex partition. The only 'bad' case is the first case of the first row. If all four graphs are non-empty, the algorithm would reject in the next step. However, then $V(G) \setminus N_G[u]$ contains exactly two vertices (remember that there are no false twin vertices), and the neighbourhoods of these two vertices partition $N_G(u)$ in exactly two sets. Then, $V(G) \setminus N_G[u]$ fails to satisfy the neighbourhood condition of conditional number 5, and the execution continues with conditional number 6. With previous arguments, SIMPLIFY returns ACCEPT or a graph. This completes the proof of the lemma. ∎

The recognition algorithm for lc3-graphs can be summarised and described as a sequence of iterated reductions by lc3-decomposition (the inverse of lc3-composition) and application of SIMPLIFY. To apply SIMPLIFY, a vertex partition has to be determined. Fortunately, the structure of lc3-graphs allows to restrict to only a few such vertex partitions. The complete algorithm is

Algorithm LC3-GRAPHRECOGNITION
Input a graph $G = (V, E)$
Result an answer ACCEPT and a pseudo lc3-expression for G, if G is an lc3-graph, or an answer REJECT, if G is not an lc3-graph.

begin
set G **:=** $G/\sim_{\text{ft}}$;
while no return **do**
1 **if** G is edgeless **then** **output** $(V(G))$; **return** ACCEPT
2 **else-if** G is the result of an lc3-composition of at least two non-empty graphs **then**
output an lc3-composition scheme;
if all partition graphs are simple cographs **then** **return** ACCEPT
else **set** G to the partition graph that is not a simple cograph **end if**;
3 **else-if** there is a vertex u such that
SIMPLIFYMOD on $G-u$ with partition $(N_G(u), V(G) \setminus N_G[u])$ returns a graph G' **then**
output "$\otimes(\{u\}|\emptyset, \bullet)$"; **set** G **:=** G'
4 **else-if** there is a pair u, v of non-adjacent vertices where v is almost universal such that
SIMPLIFYMOD on $G \setminus \{u, v\}$ with partition $(N_G(u), V(G) \setminus (N_G(u) \cup \{u, v\}))$
returns a graph G' **then**
output "$\otimes(\{u\}|\{v\}, \bullet)$"; **set** G **:=** G'
5 **else** **return** REJECT **end if**
end while
end.

Figure 8: The recognition algorithm for lc3-graphs.

given in Figure 8. The applied procedure SIMPLIFYMOD is a variant of SIMPLIFY, with the only difference that it returns a graph on a single vertex instead of an answer ACCEPT. This modification helps to present LC3-GRAPHRECOGNITION as short as possible (otherwise the algorithm had to distinguish more cases). Note that also LC3-GRAPHRECOGNITION provides additional output for constructing an lc3-expression for the input graph. We show in the following that LC3-GRAPHRECOGNITION is correct, i.e., that it accepts exactly the lc3-graphs and therefore the graphs of linear clique-width at most 3.

Theorem 19. *Algorithm* LC3-GRAPHRECOGNITION *is an lc3-graph recognition algorithm.*

Proof We show that Algorithm LC3-GRAPHRECOGNITION accepts an input graph if and only if it is an lc3-graph. Due to Lemma 4 and Theorem 13, a graph G is an lc3-graph if and only if $G/\sim_{\text{ft}}$ is an lc3-graph. Hence, we can restrict to consider only graphs without non-trivial maximal independent-set modules and show that the **while**-loop finally ends with answer ACCEPT if and only if the input graph is an lc3-graph. We prove the statement by induction over the number of vertices of the input graph. A graph without false twins is edgeless if and only if it has exactly one vertex. Edgeless graphs are lc3-graphs, and the algorithm accepts. Now, let the input graph have at least two vertices, which means in particular that it contains an edge. Let G be the result of an lc3-composition of an edgeless graph, a simple cograph, an arbitrary graph G' and three edgeless graphs. Since G' is a module of G, G' does not contain false twin vertices. Applying the induction hypothesis, G' is accepted by LC3-GRAPHRECOGNITION if and only if G' is an lc3-graph. If G is an lc3-graph then G' is an lc3-graph according to Lemma 8, if G is not an lc3-graph then G' cannot be an lc3-graph due to Lemma 10. Hence, LC3-GRAPHRECOGNITION accepts if and only if G is an lc3-graph. For the rest of the proof, let G not be the result of an lc3-composition.

First, let G be accepted by Lc3-graphRecognition. This means that conditional number 3 or 4 is positive for G. Let there be a vertex u such that SimplifyMod on $G-u$ and vertex partition $(N_G(u), V(G) \setminus N_G[u])$ returns a graph G'. Since u is neither isolated nor universal, $N_G(u)$ and $V(G) \setminus N_G[u]$ are non-empty. Thus, G' is a proper induced subgraph of G without false twin vertices due to assumption and by Lemma 18. By induction hypothesis, $G-u$ is an lc3-graph, and due to Lemma 18, there is an lc3-expression T for $G-u$ with vertex partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse. Then, $T \otimes (\{u\}|\emptyset, \bullet)$ or $T \otimes (\bullet, \{u\}|\emptyset)$ is an lc3-expression for G. The proof for conditional number 4 is similar. Thus, we can conclude that every accepted graph is an lc3-graph.

For the converse, we can particularly assume that G contains no isolated or universal vertex; otherwise G is the result of an lc3-composition. Let T be an lc3-expression for G. We distinguish different cases with respect to the last operation in T. By assumption and since G is not edgeless, the last operation cannot be of the forms (d1) and (d3) and the complex operation of (d4). Let the last operation in T be of the form (d2). Then, there is a vertex u such that $G-u$ is an lc3-graph with lc3-expression T' with vertex partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse. Due to Lemma 18, SimplifyMod returns a proper subgraph of G, and by induction hypothesis, this subgraph is accepted. Now, let the last operation of T be of the form $\odot(A_1|A_2, \bullet)$ or $\odot(\bullet, A_1|A_2)$. If $\odot = \oplus$ then A_1 is empty, since G contains no isolated vertex. The vertex in A_2, say u, defines vertex partition $(V(G) \setminus N_G[u], N_G(u))$ or $(N_G(u), V(G) \setminus N_G[u])$ for $G-u$, and this partition corresponds to the vertex partition for $G(T)-u$. We conclude as in the previous case that G is accepted. Let $\odot = \otimes$. If A_2 is empty, the case is similar to the previous case. If A_1 is empty, the vertex in A_2 is universal. Thus, A_1 and A_2 are non-empty. The vertex in A_2 is adjacent to all vertices but the vertex in A_1, and the vertex in A_1 defines a vertex partition for G. Similar to the previous cases, the algorithm accepts. So, we can conclude that Lc3-graphRecognition exactly accepts the lc3-graphs. ∎

It remains to consider the running time of the presented algorithms. We show that Algorithm Lc3-graphRecognition has an $\mathcal{O}(n^2m)$-time implementation. This result is partitioned into three subresults. The first subresult has already been given in Lemma 16. In the following, we consider Simplify and the problem of deciding whether a graph is the result of an lc3-composition and computing appropriate partition graphs.

Lemma 20. *Algorithm* Simplify *has a linear-time implementation.*

Proof We define a data structure that allows checking for satisfaction of the conditions of the first four conditionals in constant time. For every vertex, we store the number of neighbours in its own partition set and in the other partition set. For every number between 0 and $|V(G)|$, there are five types of buckets containing vertices: two bucket types for each partition set and a bucket type for the total vertex degree. Vertices appear in these buckets according to their degrees:

- every vertex appears in the bucket of the fifth type that corresponds to its total degree
- a vertex of the left partition set appears in a bucket of the first type, if it has neighbours only in the left partition set; if it has neighbours only in the right partition set, it appears in a bucket of the second type
- analogous to the previous case, vertices of the right partition set appear in buckets of the third and fourth type, if they have neighbours in exactly one of the two partition sets.

Finally, there are two variables for the cardinalities of the two partition sets and a variable for the number of edges in the graph.

Conditionals number 1 and 2 can be decided in constant time. For conditional number 3, at most six buckets have to be checked: buckets of the fifth type answer whether a vertex is isolated or universal. Buckets of the first and second type answer whether there is a vertex u in the left partition set such that $N_G[u] = B$ or $N_G(u) = C$. So, using these buckets, satisfaction of the condition of conditional number 3 can be checked in constant time. For the condition of conditional number 4, an almost-universal vertex can be found using the buckets of the fifth type. However, it is not easy to find the required second vertex. We additionally assign to every vertex of degree $|V(G)| - 2$ the unique non-neighbour, and vice versa, if the two vertices are in the same partition set. If the condition is satisfied, there are two non-adjacent vertices u, v in the left partition set such that v is almost-universal and u is adjacent to only vertices in the left partition set, or similarly with vertices in the right partition set. To decide the condition in constant time, we check the buckets of the first and third type for a vertex of degree $|B| - 2$ or $|C| - 2$, respectively, and with an assigned non-neighbour. For a fast implementation, we partition the buckets of the first and third type into two subbuckets, one for vertices with assigned non-neighbour and one for vertices without. Now, if conditional number 3 or 4 is positive, we have to update the data. The update affects only neighbours, whose degrees are decreased. Then, they have to be moved into other buckets, which takes constant time for each vertex. Note that vertices now may have to be added to a bucket of the first four types. Finally, a non-neighbour may have become almost-universal. Here it is to observe that a vertex becomes almost-universal at most once. So, when a vertex becomes almost-universal, its unique non-neighbour can be found in time proportional to its degree, and the link can be established.

For the condition of conditional number 5, it takes constant time to determine whether a partition set contains only one vertex, say u, and whether u has exactly two non-neighbours. In time proportional to the degree of u, the two non-neighbours can be determined. Comparing the adjacency lists of the two non-neighbour vertices shows whether the neighbourhoods can be ordered. If the test fails, the procedure stops with conditional number 6 or the **return** command. Conditional number 6 requires linear time due to Lemma 16 and Corollary 6. If the test does not fail, Simplify continues with a completely new partition, for which the vertex degrees have to be re-computed. Visiting every neighbour of u once, the array representing the current vertex partition can be modified to represent the new vertex partition. For the vertex degrees, we show that the modification can be done by considering only edges that are incident to vertices in one of the two partition sets. Let A be the left or right partition set. The numbers of neighbours in the two partition sets can be computed straightforward for every vertex in A by reading the adjacency lists. Whenever a vertex from the other partition set appears, its degree pair is modified (decrease the number of neighbours in the same partition set, increase the number of neighbours in the other partition set). When a vertex has a neighbour in the other partition set, it is removed from the bucket of one of the first four types. When this update is done the next time, one of the two partition sets will be empty, so that, when we choose the correct set A, we obtain overall linear running time. The algorithm does not know the correct partition set. However, it can compute the degree sum for the two partition sets and choose the partition set of smaller degree sum as A. Then, vertices may be considered several times, but the effort is always balanced with the deleted vertices. To finish the analysis, it is important to observe that there is at most one vertex of degree 0 in A, and this will be deleted before execution reaches conditional number 5 the next time. And for computing the degree sums, only the degree sum of the smaller partition set is computed and the degree sum of the other

partition set is determined by subtraction.

If the input graph is an lc3-graph, an appropriate lc3-expression can be obtained from the output operations as shown in the proof of Theorem 19. Care has to be taken of only the "turn" command. This completes the proof. ∎

Lemma 21. *There is a linear-time algorithm that checks whether a given graph is the complete or incomplete lc3-composition of at least two non-empty graphs $G_1, G_2, G_3, H_1, H_2, H_3$ where H_1, H_2, H_3 are edgeless graphs and G_1, G_2, G_3 are an edgeless graph, a simple cograph and an arbitrary graph. In the positive case, the algorithm outputs an appropriate decomposition.*

Proof The algorithm considers several cases. First, let G be disconnected. If one of the connected components is a simple cograph then accept; otherwise reject. In the positive case, G is an incomplete lc3-composition where G_1, H_1, H_2, H_3 of Figure 4 are empty, a simple cograph connected component goes into G_2 and all other connected components go into G_3. For the negative case, observe the following. None of the connected components is edgeless as they are not simple cographs, so they all have to go into the graphs G_1, G_2, G_3 of Figure 4. If G_1 is not empty, the resulting lc3-composition is not disconnected, so that G_1 has to be empty. But with only G_2 and G_3 non-empty, G cannot be obtained as an lc3-composition with none of the two composition graphs being a simple cograph.

Second, let G be connected and let the complement of G be disconnected. If one of the co-connected components is a simple cograph then accept; otherwise reject. Similar to the first case, G can be obtained as the complete lc3-composition of a simple cograph (in G_2) and another graph (in G_3). The negative case is more complex. We distinguish between two cases depending on whether G is an incomplete or complete lc3-composition.

- Let G be an incomplete lc3-composition. By assumption, H_2 and H_3 of Figure 4 are empty. Suppose that H_1 is non-empty. If G_3 is empty then $G = (G_1 \oplus H_1) \otimes G_2$, and every co-connected component of G is completely contained in either $G_1 \oplus H_1$ or G_2. Hence, G contains a co-connected component that is a simple cograph. If G_3 is non-empty then G_1 and G_2 are also non-empty. But then, the complement of G is not disconnected, so that this case cannot happen. Now, let H_1 be empty. Then, $G = G_1 \otimes (G_2 \oplus G_3)$, and similar to the case about, G_1 or $G_2 \oplus G_3$ is a simple cograph, and G contains a co-connected component that is a simple cograph.
- Now, let G be a complete lc3-composition. If H_1 is non-empty then G_2 is non-empty. Now consider H_2. If H_2 is non-empty then the complement of G is connected. Thus H_2 is empty. Then G results from an incomplete lc3-composition since G_1 and G_2 can be merged into G_1 and H_1 and H_2 can be merged into H_1. Thus by the previous case we conclude that H_1 is empty. Let G_1 be non-empty. By a connectivity argument, G_i is non-empty and H_i is empty for $i = 2$ or $i = 3$, and by symmetry, we can assume that H_3 is empty and G_3 is non-empty. Note then that every co-connected component of G is entirely contained either in G_3 or in $(G_1 \otimes G_2) \oplus H_2$. According to assumption, G_3 is a simple cograph or $(G_1 \otimes G_2) \oplus H_2$ is a simple cograph. Hence, G contains a co-connected component that is a simple cograph. Finally, if G_1 is non-empty then $G = (G_2 \oplus H_2) \otimes (G_3 \oplus H_3)$, and every co-connected component of G is entirely contained in either $G_2 \oplus H_2$ or $G_3 \oplus H_3$. Hence, G contains a co-connected component that is a simple cograph.

Third, let G be connected and co-connected. We first describe the different situations. If G is an incomplete lc3-composition then H_2 and H_3 of Figure 4 are empty and G_1, G_2, G_3, H_1 are

non-empty (otherwise, G or its complement would be disconnected) and maximal modules of G in a P_4-structure. Remember that the P_4 is a chordless path on four vertices. The complement graph of a P_5 is called *house*, and a *bull* is obtained from a P_4 and an additional vertex that is adjacent to the two middle vertices of the P_4. If G is a complete lc3-composition then we distinguish the following cases:

- If G_1 is empty then $H_2, (G_3 \oplus H_3), G_2, H_1$ are non-empty and maximal modules of G in a P_4-structure.
- If G_2 is empty then H_1 is empty. By assumption, G_1, G_3, H_2, H_3 are non-empty and maximal modules of G in a P_4-structure.
- If G_3 is empty then $(G_1 \oplus H_1), G_2, H_3, H_2$ are non-empty and maximal modules of G in a P_4-structure.
- If H_2 is empty then G_2 is empty, since G is co-connected. With a similar argument for the rest of the composition graphs we conclude that H_2 must be non-empty since G is connected and co-connected.
- If H_3 is empty then we have the following two cases. If G_1 is empty then H_1, G_2, G_3, H_2 are maximal modules of G in a P_4-structure. If G_1 is non-empty then G_1, G_2, G_3, H_1, H_2 are maximal modules of G in a bull-structure. In any other case, G would be neither connected nor co-connected.
- If H_1 is empty then G_1, H_2, H_3 and at least one of G_2 and G_3 must be non-empty. If G_2 is empty then G_1, G_3, H_2, H_3 are maximal modules of G in a P_4-structure. Similarly, if G_3 is empty then G_1, G_2, H_3, H_2 are maximal modules of G in a P_4-structure. If both G_2 and G_3 are non-empty then G_1, G_2, G_3, H_2, H_3 are maximal modules of G in a house-structure.
- If $G_1, G_2, G_3, H_1, H_2, H_3$ are non-empty then all these graphs are maximal modules of G in a situation exactly described by Figure 4.

Hence for the decision algorithm, compute the maximal modules of G and the corresponding prime graph. Check for the prime graph whether it is one of the above-mentioned (P_4, house, bull, the graph shown in Figure 4), check for edgeless graphs and simple cographs and test whether the graphs that are not edgeless are in the correct positions. If all conditions are satisfied then accept; otherwise reject. Correctness immediately follows from the study above.

For the running time, we observe the following: (1) edgeless graphs and simple cographs can be recognised in linear time due to Corollary 6, (2) connected components and co-connected components can be computed in linear time, (3) maximal modules and corresponding prime graphs can be computed in linear time [7]. Then, only a finite number of configurations have to be checked (in the third case), so that all this sums up to total linear running time. The output is obtained as described, and we conclude the proof. ∎

Theorem 22. *There is an algorithm that decides in $\mathcal{O}(n^2m)$ time whether a given graph has linear clique-width at most 3. If so, a linear 3-expression is output.*

Proof By Theorems 13 and 19, it suffices to analyse the running time of Algorithm Lc3-graphRecognition. For a given graph G, $G/\sim_{\text{ft}}$ can be computed in linear time. Every **while** loop execution is done with a smaller graph, so that there are at most n **while** loop executions.

A single **while** loop execution takes time $\mathcal{O}(nm)$: linear time for checking for an lc3-composition (Lemma 21) and at most $2n$ applications of SIMPLIFY (SIMPLIFYMOD, more precisely, which also has a linear-time implementation) that require linear time each due to Lemma 20. This shows the total $\mathcal{O}(n^2m)$ running time. The linear 3-expression is obtained by first constructing an lc3-expression and then converting it into a linear clique-width expression according to the rules established in the proof of Theorem 13. Note that the length of both expressions is linear in the number of vertices. ∎

8 Conclusions

We have given characterisations for graphs of linear clique-width at most 2 and at most 3 of different types. One type of characterisation is via graph decomposition schemes. For graphs of linear clique-width at most 3, we have presented an efficient recognition algorithm using the decomposition approach. We have also seen that our decomposition for graphs of linear clique-width at most 3 generalises the decomposition for graphs of linear clique-width at most 2. Can this approach be extended to graphs of linear clique-width at most 4, or in general to graphs of arbitrarily bounded linear clique-width?

For graphs of linear clique-width at most 2, there exists a characterisation by a small set of forbidden induced subgraphs. Does there exist an easily describable set of forbidden induced subgraphs that characterises graphs of linear clique-width at most 3?

References

[1] A. Brandstädt, V. B. Le, J. P. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications, 1999.

[2] D. G. Corneil, M. Habib, J.-M. Lanlignel, B. A. Reed, U. Rotics. Polynomial time recognition of clique-width ≤ 3 graphs. *LATIN 2000*, Springer LNCS, 1776:126–134, 2000.

[3] D. G. Corneil, Y. Perl, L. K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14:926–934, 1985.

[4] B. Courcelle, J. Engelfriet, G. Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46:218–270, 1993.

[5] B. Courcelle, J. A. Makowsky, U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Computing Systems*, 33:125–150, 2000.

[6] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.

[7] E. Dahlhaus, J. Gustedt, R. M. McConnell. Efficient and practical algorithms for sequential modular decomposition. *Journal of Algorithms*, 41:360–387, 2001.

[8] M. R. Fellows, F. A. Rosamond, U. Rotics, S. Szeider. Clique-Width is NP-Complete. *SIAM Journal on Discrete Mathematics*, 23:909–939, 2009.

[9] F. Gurski. Characterizations for co-graphs defined by restricted NLC-width or clique-width operations. *Discrete Mathematics*, 306:271–277, 2006.

[10] F. Gurski. Linear layouts measuring neighbourhoods in graphs. *Discrete Mathematics*, 306:1637–1650, 2006.

[11] F. Gurski and E. Wanke. On the relationship between NLC-width and linear NLC-width. *Theoretical Computer Science*, 347:76–89, 2005.

[12] M. Habib, R. M. McConnell, C. Paul, L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234:59–84, 2000.

[13] P. Hliněný, S. Oum, D. Seese, G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal*, 51:326–362, 2008.

[14] D. Kratsch and L. Stewart. Domination on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 6:400–417, 1993.

[15] V. Lozin and D. Rautenbach. The relative clique-width of a graph. *Journal of Combinatorial Theory, Series B*, 97:846–858, 2007.

[16] N. Mahadev and U. Peled. Threshold graphs and related topics. *Annals of Discrete Mathematics 56*, North Holland, 1995.

[17] R. M. McConnell and J. P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201:189–241, 1999.

[18] F. S. Roberts. Indifference graphs. in: F. Harary (Ed.), *Proof techniques in graph theory*, pp. 139–146, Academic Press, New York, 1969.