REPORTS IN INFORMATICS

ISSN 0333-3590

Graphs of small bounded linear clique-width

P. Heggernes, D. Meister, Ch. Papadopoulos

REPORT NO 362 **October 2007**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL http://www.ii.uib.no/publikasjoner/texrap/pdf/2007-362.pdf

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at http://www.ii.uib.no/publikasjoner/texrap/.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Graphs of small bounded linear clique-width*

Pinar Heggernes† Daniel Meister† Charis Papadopoulos†

October 9, 2007

Abstract

In this paper we give the first characterisation of graphs with linear clique-width at most 3, and we give a polynomial-time recognition algorithm for such graphs. In addition, we give a new characterisation of graphs with linear clique-width at most 2 and a new layout characterisation of linear clique-width in general. Among our results is also a decomposition scheme that preserves the linear clique-width of the decomposable subgraph.

1 Introduction

Clique-width is an important graph parameter that is useful for measuring the computational complexity of generally NP-hard problems on specific graph classes. In particular, all problems that can be expressed in a certain kind of monadic second order logic can be solved in linear time on graphs whose clique-width is bounded by a constant [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 related graph parameter *linear clique-width* is obtained by restricting the allowed clique-width operations to only 'vertex creation', 'join' and 'relabel'. Both parameters are NP-hard to compute, even on complements of bipartite graphs [8]. A graph class can have bounded clique-width but unbounded linear clique-width. Examples of such graph classes are 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, 12]. However, clique-width can be viewed as a more general concept than treewidth since there are graphs of bounded clique-width but unbounded treewidth, whereas graphs of bounded treewidth have bounded clique-width. While treewidth is widely studied and well understood the knowledge on clique-width is still limited. The study of the more restricted parameter linear clique-width is a step towards a better understanding of clique-width. For example, NP-hardness is obtained by showing that linear clique-width is NP-hard to compute [8].

In 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 and the first polynomial time algorithm ($\mathcal{O}(n^2m)$) to decide whether a graph has linear

*This work is supported by the Research Council of Norway through grant 166429/V30.

†Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: pinar@ii.uib.no, danielm@ii.uib.no, charis@ii.uib.no

clique-width at most 3. We also show that such graphs are both cocomparability and weakly chordal graphs, and we present a decomposition scheme for them. For bounded linear clique-width, until now only graphs of linear clique-width at most 2 could be recognised in polynomial time [9, 3]. For bounded clique-width, graphs of clique-width at most 2 [6] and at most 3 [2] can be recognized in polynomial time ($\mathcal{O}(n^2m)$). Whereas clique-width 2 graphs are characterised as the class of cographs [6], no characterisation is known for clique-width 3 graphs. Furthermore, even from the proposed algorithm in [2] there is no straightforward way of deciding whether a graph of clique-width at most 3 has linear clique-width at most 3.

Our main motivation for characterising graphs of a given linear clique-width is to obtain decomposition results for these graphs that are useful for designing algorithms on them. Decomposition means a recursive scheme that partitions the vertex set into smaller sets such that edges between vertices in different sets are determined immediately and not influenced by earlier or later partitions. For this purpose, we also study graphs of linear clique-width at most 2, although a forbidden subgraph characterisation for them is already known. Gurski [9] showed that a graph has linear clique-width at most 2 if and only if it contains no induced copy of a $2K_2$, P_4, or co-$(2P_3)$. As an additional result and independently of the results of [9], we give an alternative characterisation of graphs of linear clique-width at most 2, which is needed for the understanding of our first mentioned result, and which we find interesting on its own because it gives a decomposition scheme for these graphs.

Before giving the above mentioned results, we start with more general results on linear clique-width. First we present and demonstrate the use of a new layout characterisation of linear clique-width. Treewidth and pathwidth have algorithmically useful characterisations through vertex layouts and embeddings into particular graph classes, but no such result is known about clique-width. Recently Gurski [10] gave a layout characterisation of linear clique-width. In this paper, we give a similar but independent layout characterisation of linear clique-width, which has a simpler statement and proof. Second, we give a characterisation of the linear clique-width of a graph 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 the given graph can be larger than the linear clique-width of any of its connected components. Going back to clique-width, several graph operations have been proposed that maintain the clique-width of a graph [6, 12]. For that purpose, one of the most famous and applicable operations is obtained through the prime induced graphs of a given graph with respect to its modular decomposition. Motivated by this operation for clique-width, we give a similar decomposition scheme for the linear clique-width of a graph by simply disregarding false twins.

Our paper is organised as follows. In the next section we give the necessary background and notation. Section 3 presents the alternative layout characterisation of linear clique-width, and Section 4 presents the decomposition scheme that preserves linear clique-width. Section 5 presents the additional characterisations of graphs of linear clique-width at most 2. Sections 6 and 7, which present the main results of the paper, contain the characterisation and some properties of graphs with linear clique-width at most 3, and how to recognise them in polynomial time. The paper is concluded in Section 8.

2 Graph preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by $V(G) = V$ and $E(G) = E$, respectively, with $n = |V|$ and $m = |E|$. For a vertex subset $S \subseteq V$, the *subgraph of G induced by S* is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V \setminus S]$ and by $G - v$ the graph $G[V \setminus \{v\}]$.

The *neighbourhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighbourhood* of x is $N_G[x] = N_G(x) \cup \{x\}$. Vertex x is *isolated* in G if $N_G(x) = \emptyset$ and *universal* if $N_G[x] = V(G)$. The *degree* of a vertex x in a graph G is $d_G(x) = |N_G(x)|$. A vertex x is called *almost-universal* if $d_G(x) = |V(G)| - 2$. For $S \subseteq V$, $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. Two vertices x, y of G are called *true twins* if $N_G[x] = N_G[y]$ and they are called *false twins* if $N_G(x) = N_G(y)$.

For a pair of vertices x, y of G we call xy a *non-edge* of G if $xy \notin E$. The *complement* $\overline{G}$ of a graph G consists of all vertices and all non-edges of G. A *chord* of a path or a cycle is an edge between two non-consecutive vertices of the path or the cycle. A chordless cycle on k vertices is denoted by C_k and a chordless path on k vertices is denoted by P_k. The graph consisting of only two disjoint edges is denoted by $2K_2$ while the complement of the graph consisting of two disjoint P_3's is denoted by co-$(2P_3)$. The complement of a P_5 is called *house*. The graph obtained from a P_4 and an additional vertex adjacent only to the two vertices of degree 2 in the P_4 is called *bull*.

A graph is *connected* if there is a path between every pair of vertices. If the complement of a graph is connected then we say that the graph is *co-connected*. A *connected component* of a disconnected graph is a maximal connected subgraph of it. A *co-connected component* of a graph G is a connected component of $\overline{G}$. A *clique* is a set of pairwise adjacent vertices, while an *independent set* is a set of pairwise non-adjacent vertices.

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

2.1 Known classes of graphs

Here we give an overview of well-known graph classes that will be needed in this paper. For a set $\mathcal{H}$ of graphs, a graph is called *$\mathcal{H}$-free* if it does not contain a graph from $\mathcal{F}$ as induced subgraph.

The class of *cographs*, also known as *complement reducible graphs*, is defined recursively as follows: a single vertex is a cograph, the disjoint union of two cographs is also a cograph, and the complement of a cograph is a cograph. It is known that the class of cographs coincide with the class of P_4-free graphs [3].

A *treshold graph* is a graph that can be de-constructed by repreatedly deleting an isolated or a universal vertex [1]. Equivalently, threshold graphs are the graphs whose vertices can be assigned real numbers such that two vertices are adjacent if and only if the sum of their assigned numbers is not smaller than a given real-valued threshold. In addition, threshold graphs are exactly the class of $\{2K_2, P_4, C_4\}$-free graphs [14]. Note also that threshold graphs are cographs [1].

An undirected graph is called *comparability* if directions can be assigned to its edges so that the resulting directed graph is (i) acyclic and (ii) whenever there are directed edges from a vertex a to another vertex b and from b to another vertex c, then there is a directed edge from a to c. A graph is called *cocomparability* if it is the complement of a comparability graph. A useful characterisation of cocomparability graphs is given by the following vertex ordering. For a graph G, a vertex ordering σ is called a *cocomparability ordering* if for every triple u, v, w of vertices such that $u \prec_\sigma v \prec_\sigma w$, $uw \in E$ implies $uv \in E$ or $vw \in E$. A graph G is cocomparability if and only if G admits a cocomparability ordering [13].

A graph is called *weakly-chordal* if it does not contain a C_k or $\overline{C}_k$ as an induced subgraph, with $k \geq 5$. Note that cographs are cocomparability graphs and weakly-chordal graphs but there is no relationship between a cocomparability graph and a weakly-chordal graph. For further properties on the mentioned graph classes we refer to [1].

2.2 Clique-width and linear clique-width

The notion of *clique-width* of graphs was first introduced by Courcelle, Engelfriet, and Rozenberg in [4]. The *clique-width* of a graph G, denoted by $cwd(G)$, is defined as the minimum number of labels needed to construct G, using the following operations:

(i) Creation of a new vertex v with label i, denoted by $i(v)$;

(ii) Disjoint union, denoted by $\oplus$;

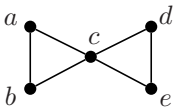
(iii) Changing all labels i to j, denoted by $\rho_{i\to j}$;

(iv) Adding edges between all vertices with label i and all vertices with label j, $i \neq j$, denoted by $\eta_{i,j} = \eta_{j,i}$.

An expression built by using the above four operations is called a *clique-width expression*. If k labels are used in a clique-width expression then it is called a *k-expression*. We say that a k-expression t *defines* a graph G if G is equal to the graph obtained by using the operations in t in the order given by t.

The *linear clique-width* of a graph, denoted by $\mathrm{lcwd}(G)$, is introduced in [11] and defined by restricting the disjoint union operation (ii) of clique-width. In a linear clique-width expression all clique-width operations are allowed, but whenever the $\oplus$ operation is used, at least one of the two operands must be an expression defining a single vertex. The restricted version of operation (ii) becomes redundant if we allow operation (i) to automatically add the vertex to the graph as an isolated vertex when it is created. For simplicity, we adopt this notation in this paper. Hence whenever a vertex v is created with operation $i(v)$, it is added to the graph as an isolated vertex v with label i, which means that we never use operation $\oplus$ in our linear clique-width expressions. With this convention, the linearity of a linear clique-width expression becomes clearly visible.

Traditionally in a clique-width expression every operation except the disjoint union $\oplus$ is followed by their operands, whereas for the disjoint union operation its two operands are placed side by side. However, as we explained before, in a linear clique-width expression we disregard the disjoint union operation and for that reason the relabeling and join operations, ρ and η, follow their operands.



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

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

Figure 1: Two expressions defining the graph on the left side.

In Figure 1 we show a graph G and two expressions t_1 and t_2 defining G. Note that t_1 is a 2-expression and t_2 is a 3-expression since there are 2 and 3 labels used in each expression, respectively. Moreover both t_1 and t_2 are clique-width expressions. However only t_2 is a linear clique-width expression, as in t_1 there is a disjoint union operation between two non-trivial graphs.

For every k-expression that defines a graph G there is a parse tree that encodes the composition of G starting with single vertices followed by interleaved operations of relabeling, edge insertion, and disjoint union. If t is a linear clique-width expression then its parse tree is path-like. Thus one can view the relationship between clique-width and linear clique-width analogous to the relationship between treewidth and pathwidth. Note that the difference between clique-width and linear clique-width of a graph can be arbitrarily large. For example cographs and trees have bounded clique-width [6] but unbounded linear clique-width [11]. It is easy to see that the edgeless graphs are exactly the graphs of clique-width 1 and linear clique-width 1. To create an edge, the join operation requires the existence of vertices with different labels.

3 A layout characterisation of linear clique-width

In this section we present a new layout characterisation of linear clique-width. This is independent of but coincidentally similar to the layout characterisation of linear clique-width given by Gurski [10]. However, our layout characterisation has a simpler formulation.

Let $G = (V, E)$ be a graph, and let β be a layout for G. Let x be a vertex of G, where $p =_{\text{def}} \beta^{-1}(x)$. The *vertices to the left of x with respect to β* are $\beta(1), \ldots, \beta(p-1)$ and denoted as $L_\beta(x)$, and the *vertices to the right of x with respect to β* are $\beta(p+1), \ldots, \beta(n)$ and denoted as $R_\beta(x)$. We write $L_\beta[x]$ and $R_\beta[x]$, if x is included. For a partition (A, B) of the vertex set of G, a *group* in A is a maximal set of vertices with the same neighbourhood in B. By $\nu_G(A)$, we denote the number of groups in A with respect to $V(G) \setminus A$. The goal is to define a width measure on layouts that corresponds to the linear clique-width. To achieve this, we define a function, that assigns number 0 or 1 to each vertex. We call this function ad, and it is parameterised by β. For a vertex x of G, $\text{ad}_\beta(x) = 1$ in exactly the following cases ($\text{ad}_\beta(x) = 0$ otherwise):

- x is single vertex in its group in $L_\beta[x]$
- x is not single vertex in its group in $L_\beta[x]$ and all (other) vertices in the group are neighbours of x
- x is not single vertex in its group in $L_\beta[x]$ and there are a non-neighbour y in its group and a neighbour z in $L_\beta(x)$ such that z is not a neighbour of y.

Note that the first case can be considered a special case of the second. We distinguish the two cases for convenience and to make the definition more clear.

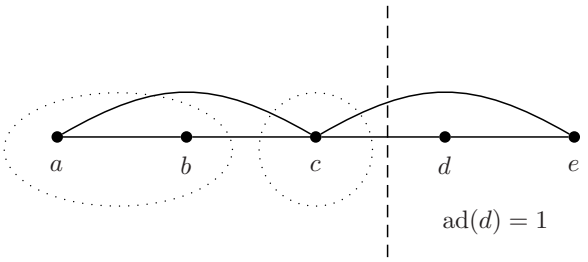


Figure 2: Illustrating the groupwidth of a given layout for the graph shown in Figure 1.

Definition 1. *Let $G = (V, E)$ be a graph. The* groupwidth *of a layout β for G, denoted as* $\mathrm{gw}(G, \beta)$*, is defined as the maximum over $\nu_G(L_\beta(x)) + \mathrm{ad}_\beta(x)$ for all $x \in V$. The* groupwidth *of G, denoted as* $\mathrm{gw}(G)$*, is the minimum group width taken over all layouts for G.*

In Figure 2 we give a specific layout β of the graph shown in Figure 1. If we consider vertex d then there are two groups in $L_\beta(d)$. Moreover $\mathrm{ad}_\beta(d) = 1$ since the group that d belongs to in $L_\beta[d]$ is $\{c, d\}$ and, thus, the second rule applies to d. It is not difficult to see that for the graph G shown in the figure, $\mathrm{gw}(G) = 3$.

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

Proof First we show that lcwd $\geq$ gw. Let $G = (V, E)$ be a graph on n vertices. Let a be a linear clique-width k-expression for G, where $k \geq 1$. We show that $\mathrm{gw}(G) \leq k$, which gives the result choosing an expression for $k = \mathrm{lcwd}(G)$. Let β be the vertex layout corresponding to the order in which the vertices are created by a. Let $G_1, \ldots, G_n$ be the labelled subgraphs of G defined by a where G_i is the graph right before the ith vertex is created. We first show that $\nu_G(V(G_i)) \leq k$ for all $i \in \{1, \ldots, n\}$. But this directly follows from the definition of a group, since vertices from different groups must have different labels. Now, we show that $\nu_G(V(G_i)) + \mathrm{ad}_\beta(\beta^{-1}(i)) \leq k$ for all $i \in \{1, \ldots, n\}$. Let $i \in \{1, \ldots, n\}$. If $\nu_G(V(G_i)) < k$ then $\nu_G(V(G_i)) + \mathrm{ad}_\beta(\beta^{-1}(i)) \leq k$, following from $\mathrm{ad}_\beta(\beta^{-1}(i)) \leq 1$. So, let $\nu_G(V(G_i)) = k$. This means that a creates the next vertex, say x, assigning a label that is assigned also in G_i. We have to show that $\mathrm{ad}_\beta(x) = 0$. Let A denote the set of vertices of G_i having the same label as x. By assumption, $A \neq \emptyset$. Then, x is not adjacent to any of the vertices in A, and A and x are contained in the same group in $L_\beta[x]$. Let A' denote the group in $L_\beta[x]$ containing x. Since vertices in A' that are not in A can differ from the vertices in A only with respect to x, all vertices in $A' \setminus (A \cup \{x\})$ are neighbours of x. It follows that every neighbour of x in G is a neighbour of every vertex in A. Thus, $\mathrm{ad}_\beta(x) = 0$, and we conclude this part of the proof.

Next we show that lcwd $\leq$ gw. Let $G = (V, E)$ be a graph on n vertices. Let β be a layout for G, where $\mathrm{gw}(G, \beta) = k$. We show that $\mathrm{lcwd}(G) \leq k$, which gives the result choosing a layout of groupwidth $\mathrm{gw}(G)$. We define linear clique-width k-expressions $a_1, \ldots, a_n$ for G, that create the labelled subgraphs $G_1, \ldots, G_n$ of G where G_i corresponds to $G[L_\beta[\beta^{-1}(i)]]$ and is labelled with exactly $\nu_G(L_\beta[\beta^{-1}(i)])$ many labels from the labels set $\{1, \ldots, k\}$. Let $\beta = \langle x_1, \ldots, x_n \rangle$. Clearly, $a_1 = 1(x_1)$ defines G_1 properly. Now, let $i \in \{1, \ldots, n-1\}$ and assume that a_i has already been defined. Then, G_i corresponds to $G[\{x_1, \ldots, x_i\}]$ and is labelled with exactly $\nu_G(L_\beta[x_i]) = \nu_G(L_\beta(x_{i+1}))$ many labels. Let $\mathrm{ad}_\beta(x_{i+1}) = 1$. Then, $\nu_G(L_\beta(x_{i+1})) < k$, and there is a label c in $\{1, \ldots, k\}$ not used in G_i. We define a_{i+1} iteratively by appending more and more

operations to a_i. We begin with $a_{i+1} =_{\text{def}} a_i c(x_{i+1})$. If x_{i+1} is not adjacent to any of the vertices in $L_\beta(x_{i+1})$, we are done. Otherwise, let c' be the label of a group in $L_\beta(x_{i+1})$ that contains neighbours of x_{i+1}. Append operation $\eta_{c,c'}$. By definition of group, all vertices with label c' are neighbours of x. Repeat this step until x_{i+1} is adjacent to all its neighbours in $L_\beta(x_{i+1})$. Then, the labelled graph defined by current a_{i+1} corresponds to G_{i+1}. Note that there were exactly $\nu_G(L_\beta(x_{i+1})) + \text{ad}_\beta(x_{i+1})$ many labels involved to go from G_i to the labelled graph defined by current a_{i+1}. It might now be that vertices from the same group in $L_\beta[x_{i+1}]$ have different labels. So, we add relabel operations to a_{i+1}. This can be done, since a group in $L_\beta(x_{i+1})$ is completely contained in a group in $L_\beta[x_{i+1}]$. We then have finished the definition of a_{i+1}, and it is clear that the labelled graph defined by a_{i+1}, G_{i+1}, corresponds to $G[\{x_1, \ldots, x_{i+1}\}]$ and contains exactly $\nu_G(L_\beta[x_{i+1}])$ many labels from $\{1, \ldots, k\}$. For the remaining case of $\text{ad}_\beta(x_{i+1}) = 0$, we define a_{i+1} similarly with the only exception for the choice of c. In this case, a_{i+1} creates x_{i+1} using a label already assigned in G_i. According to the definition of ad, the group A in $L_\beta[x_{i+1}]$ containing x_{i+1} contains at least two vertices, and this group can contain at most two groups in $L_\beta(x_{i+1})$. If the vertices in $A \setminus \{x_{i+1}\}$ are all neighbours of x_{i+1} then $\text{ad}_\beta(x_{i+1}) = 1$ according to definition. Hence, A contains a group A' in $L_\beta(x_{i+1})$ of non-neighbours of x_{i+1}. We choose their label as c. It remains to show that this choice does not add false edges. But again, the definition of ad ensures that every neighbour of x_{i+1} in G_i is a neighbour of every vertex in A'. This completes the proof. ∎

Note that the proof has shown an even stronger result: there is a 1-to-1 correspondence between linear clique-width expressions using the minimum number of labels and layouts of smallest groupwidth. Our first result about optimal groupwidth layouts concerns disconnected graphs. Intuitively, vertices of the same connected component should appear consecutively. We show that this is indeed true.

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 such that* $\text{gw}(G, \beta) \geq \text{gw}(G, \beta')$.

Proof Let $k =_{\text{def}} \text{gw}(G, \beta)$. We prove the statement by induction over the number of connected components. If G has only one connected component, the statement is true using β as β'. So, 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. Thus, $\text{gw}(G, \beta) \geq 2$. Let $\beta_1, \ldots, \beta_l$ be layouts obtained from β by restricting the vertices of $G_1, \ldots, G_l$, respectively. Suppose there is $i \in \{1, \ldots, l\}$ such that $\text{gw}(G_i, \beta_i) \leq k - 1$. 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 $\text{gw}(G \setminus V(G_i), \delta) \leq k$. 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, $\text{gw}(G, \beta') \leq k$. Finally, let $\text{gw}(G_i, \beta_i) = k$ for every $i \in \{1, \ldots, l\}$. For every $i \in \{1, \ldots, l\}$, determine the leftmost vertex u_i of G_i with respect to β_i such that $\nu_{G_i}(L_{\beta_i}(u_i)) + \text{ad}_{\beta_i}(u_i) = k$. 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 $\text{gw}(G, \beta') \leq k$. Suppose there is a vertex x such that $\nu_G(L_{\beta'}(x)) + \text{ad}_{\beta'}(x) > k$. 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 $\nu_G(L_{\beta'}(x)) > k$, x would not be leftmost. We consider the groups in $L_{\beta'}(x)$:

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

According to the construction of β', the groups of $L_\beta(x)$ and $L_{\beta'}(x)$ correspond on the $k-1$ many groups containing only vertices of G_j. 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, every vertex has a neighbour in G. Suppose the vertices in A have a neighbour in $R_\beta(x)$. Then, no vertex of G_i in $L_\beta(x)$ can be in A, which means that only $k-1$ many groups in $L_\beta(x)$ contain vertices of G_i. 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 operation that preserves linear clique-width

Some graph operations preserve clique-width. *Substituting* a vertex by a graph is the replacement of a vertex v of G with a graph H such that every vertex of H is adjacent to its neighbours in H and the neighbours of v in G. The *modular decomposition* is the reverse operation of obtaining a graph recursively by substitution. A *module* in a graph is a set of vertices that have the same neighbourhood outside of the module. A *prime* graph with respect to modular decomposition is a graph that cannot be obtained by nontrivial substitution. Note that the class of cographs are completely decomposable with respect to modular decomposition and they have no prime graph [1]. It is known that the clique-width of a graph G is equal to the maximum of the clique-width of all prime induced subgraphs of G [6]. Thus for the clique-width it is enough to consider the prime graphs appearing in the modular decomposition. For the linear clique-width of graphs this observation is not true, since cographs have unbounded linear clique-width.

In other words modules do not affect the clique-width of a graph and the scheme provided by modular decomposition gives an efficient way of considering only the clique-width of the decomposed subgraphs. Motivated by the above property on clique-width, in this section we give an analogous result for linear clique-width. In particular we are able to show that for certain types of modules this nice property holds when restricted to linear clique-width of graphs.

Definition 2. *Let G be a graph and M be a set of vertices of G. We call M a* maximal independent-set module *of G if M is an inclusion maximal set of vertices that is both an independent set in G and a module of G.*

For a graph G, we define a binary relation for false twins u and v, denoted by $u \sim_{\mathrm{ft}} v$. Since $\sim_{\mathrm{ft}}$ is an equivalence relation, the corresponding equivalence classes partition the vertex set. It is easy to see that the equivalence classes are exactly the maximal independent-set modules of G. The *quotient graph* of G with respect to $\sim_{\mathrm{ft}}$, denoted as $G/\sim_{\mathrm{ft}}$, 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 adjacent vertices. It is clear that $G/\sim_{\mathrm{ft}}$ is isomorphic to an induced subgraph of G. An independent-set module is called *trivial* if it contains a single vertex.

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

Proof Suppose $G/\sim_{\text{ft}}$ has a non-trivial 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. By c, we denote the vertex of $G/\sim_{\text{ft}}$ corresponding to M_c. According to definition, c then is adjacent to a in $G/\sim_{\text{ft}}$ and by assumption to b. Thus, w is adjacent to v in G. With a symmetry argument, every neighbour of v in G is also 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. ∎

Lemma 4. *For any 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 $\text{lcwd}(G/\sim_{\text{ft}}) \leq \text{lcwd}(G)$ is immediate. For showing $\text{lcwd}(G) \leq \text{lcwd}(G/\sim_{\text{ft}})$, let M_x for $x \in V(G/\sim_{\text{ft}})$ be the maximal independent-set module of G corresponding to x. Let a be a linear clique-width k-expression for $G/\sim_{\text{ft}}$ where $k = \text{lcwd}(G/\sim_{\text{ft}})$ and let ℓ_x be the label of x when adding x in the expression a. We define a linear clique-width k-expression a' for G in the following way. For every vertex x of $G/\sim_{\text{ft}}$ we place the vertices of M_x at the occurrence of the addition of x and we give them the same label of x. That is, we replace the appearance of $\ell_x(x)$ in a with $\ell_x(v_1)\cdots\ell_x(v_{|M_x|})$, where $v_i \in M_x$. In this way we construct a k-expression a'. M_x is a module in G and every vertex of M_x will obtain the same neighbourhood as x in the graph defined by a'. Moreover as every vertex of M_x receives the same label in a' it follows that M_x is an independent set in the graph defined by a'. Hence $\text{lcwd}(G) \leq \text{lcwd}(G/\sim_{\text{ft}})$. ∎

Lemma 3 defines a reduction of a graph G with respect to its false twins, and by Lemma 4 one is able to compute $\text{lcwd}(G)$ using the reduced graph $G/\sim_{\text{ft}}$, rather than G itself. Hence the operation of substituting a vertex by a graph H preserves the linear clique-width of a given graph, provided that H is an edgeless graph. Then the restricted operation of substituting a vertex by an edgeless graph defines an efficient reduction operation in a close relationship to the substitution operation and modular decomposition.

5 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 give further characterisations of these graphs, for instance by a decomposition scheme. We also give an alternative proof of the result by Gurski using the layout characterisation of linear clique-width.

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.

Note that cographs are exactly the graphs obtained when both A and B can be arbitrary simple cographs. We will show that simple cographs are exactly the graphs of linear clique-width at most 2. First, we show a connection to threshold graphs, a known subclass of cographs, that also illustrates the linearity of simple cographs.

Theorem 5. *For any graph G, G is a simple cograph if and only if $G/\sim_{\text{ft}}$ is a threshold graph.*

Proof Let G be a simple cograph. If G is an edgeless graph then $G/\sim_{\text{ft}}$ is a graph on a single vertex and therefore a threshold graph. Otherwise, if G contains an edge, there are edgeless graphs $A_1, \ldots, A_r$ and operations $\odot_i \in \{\oplus, \otimes\}$ for $i \in \{2, \ldots, r\}$ such that $G = (\cdots(A_1 \odot_2 A_2)\cdots) \odot_r A_r$. We assume that r is smallest possible. This particularly means that there is no $i \in \{3, \ldots, r\}$ such that $\odot_{i-1} = \odot_i = \oplus$. Note then that 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. Then, $G/\sim_{\text{ft}}$ is isomorphic to $G[\{a_1, \ldots, a_r\}]$. Note that $G[\{a_1, \ldots, a_r\}] = (\cdots(G[\{a_1\}] \odot_2 G[\{a_2\}])\cdots) \odot_r G[\{a_r\}]$, which shows that $G[\{a_1, \ldots, a_r\}]$ and therefore $G/\sim_{\text{ft}}$ is a threshold graph. For the converse, let $G/\sim_{\text{ft}}$ be a threshold graph, that is, the vertices of $G/\sim_{\text{ft}}$ can be enumerated as $a_1, \ldots, a_r$ such that $G/\sim_{\text{ft}} = (\cdots(A_1 \odot_2 A_2)\ldots) \odot_r A_r$ for appropriate $\odot_i \in \{\oplus, \otimes\}$, where A_i is the graph on vertex set $\{a_i\}$. Let M_i be the maximal independent-set module of G corresponding to a_i, $i \in \{1, \ldots, r\}$. Then, $G = (\cdots(G[M_1] \odot_2 G[M_2])\cdots) \odot_r G[M_r]$, thus a simple cograph. ∎

By Theorem 5 and the fact that threshold graphs can be recognised in linear time [14] we obtain the following result.

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

The following theorem, which is the main theorem of this section, gives further characterisations of simple cographs, one through a special elimination scheme and one by forbidden induced subgraphs.

Theorem 7. *For a graph G, the following statements are equivalent:*

(1) G is a simple cograph

(2) G is $\{2K_2, P_4, \text{co-}(2P_3)\}$-free

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

Proof We show the theorem in two steps. We first show equivalence of statements (2) and (3) and then show equivalence of statements (1) and (3).

(2) ⇔ (3) Let $\sigma = \langle x_1, \ldots, x_n\rangle$ be a vertex ordering for G such that x_i is an isolated vertex, a universal vertex or a false twin vertex in $G_i =_{\text{def}} G[\{x_i, \ldots, x_n\}]$, $i \in \{1, \ldots, n\}$. Suppose that G is not $\{2K_2, P_4, \text{co-}(2P_3)\}$-free. Let $j \in \{1, \ldots, n\}$ be largest such that G_j contains $2K_2$, P_4 or co-$(2P_3)$ as induced subgraph. Then, x_j is a vertex in a $2K_2$, P_4 or co-$(2P_3)$ in G_j. However, $2K_2$, P_4 and co-$(2P_3)$ do not contain a false twin or isolated or universal vertex, which contradicts the assumtion. Hence, G is $\{2K_2, P_4, \text{co-}(2P_3)\}$-free. For the converse, we show that every graph that does not have an elimination sequence of the kind of statement (3) contains a

$2K_2$, P_4 or co-$(2P_3)$ as induced subgraph. Let $G = (V, E)$ be a graph. Since every graph on at most three vertices admits such an elimination, G contains an induced subgraph G' on at least four vertices that does not have an isolated vertex, a universal vertex or a false twin vertex, but every proper induced subgraph of G' has such a vertex. Note that G' is connected, since otherwise it would contain an isolated vertex or a $2K_2$ as induced subgraph. Let x be a vertex of G'. We distinguish three cases.

(a) Let $G'-x$ have an isolated vertex y. Since G' is connected, x and y are adjacent in G'. There is a vertex z that is non-adjacent to x; otherwise, x would be universal in G' contradicting the definition of G'. Let P be a shortest x, z-path in G'. This path has length at least 2. It can be extended by starting at y to a shortest y, z-path, which is of length at least 3, and shows that G' contains a P_4 as induced subgraph.

(b) Let $G'-x$ have a universal vertex y. Since y is not universal in G', x and y are non-adjacent in G'. Since G' does not contain false twin vertices, the neighbourhood of x in G' is properly contained in the neighbourhood of y. Suppose there are vertices $a \in N_{G'}(x)$ and $d \in N_{G'}(y) \setminus N_{G'}(x)$ that are non-adjacent in G'. Then, (x, a, y, d) is a chordless path of length 3 in G', that is, G' contains the P_4 as induced subgraph. Now, assume that every pair a, d of vertices where $a \in N_{G'}(x)$ and $d \in N_{G'}(y) \setminus N_{G'}(x)$ is adjacent in G'. Remember that $N_{G'}(y) \setminus N_{G'}(x)$ is non-empty. Consider $N_{G'}(x)$. Since G' does not have a universal vertex or a false twin vertex, $G'[N_{G'}(x)]$ does not have a universal vertex or a false twin vertex. By construction, $G'[N_{G'}(x)]$ admits an elimination scheme repeatedly choosing only isolated, universal or false twin vertices. According to assumption, $G'[N_{G'}(x)]$ contains an isolated vertex a. Since a is not universal in G', $N_{G'}(x)$ contains a second vertex b. Since a and b are not false twins in G', b has a neighbour c in G' that is not a neighbour of a, and c can be only in $N_{G'}(x)$. Now consider $G'[\{x, y, a, b, c, d\}]$, where $d \in N_{G'}(y) \setminus N_{G'}(x)$. The non-neighbours of x are y and d, and the non-neighbours of a are b and c. Since all other pairs of vertices are adjacent, $\{x, y, a, b, c, d\}$ induces a co-$(2P_3)$ in G'.

(c) Let $G'-x$ have a false twin vertex pair y, y'. Since G' does not contain false twin vertices, we can assume without loss of generality that x and y are adjacent and x and y' are non-adjacent in G'. If y' has a neighbour a that is not a neighbour of x then (x, y, a, y') is a chordless path of length 4 in G', that is, $\{x, y, y', a\}$ induces a P_4 in G'. Let every neighbour of y' be a neighbour of x. If there is no vertex in $V \setminus (N_{G'}(x) \cup \{x, y'\})$ then y is a universal vertex in $G'-y'$, and we can apply case (2). So, let there be a vertex $b \in V \setminus (N_{G'}(x) \cup \{x, y'\})$. Since G' is connected, we can assume that b is a neighbour of a neighbour c of x. If c is not a neighbour of y then (y, x, c, b) is a chordless path of length 4 in G'. Thus every neighbour of b that is a neighbour of x must be a neighbour of y. If b has a neighbour d that is not a neighbour of x then $\{x, y, b, d\}$ induces a $2K_2$ in G'. So, if this is not the case, every vertex in $V \setminus (N_{G'}(x) \cup \{x, y'\})$ has neighbours only in $N_{G'}(y')$. If there are two vertices $b, b' \in V \setminus (N_{G'}(x) \cup \{x, y'\})$ such that neither $N_{G'}(b) \subseteq N_{G'}(b')$ nor $N_{G'}(b') \subseteq N_{G'}(b)$ then G' contains a $2K_2$ or a P_4 as induced subgraph. Assume that this is not the case, which means that the neighbourhoods of every pair of vertices in $V \setminus (N_{G'}(x) \cup \{x, y'\})$ is comparable with respect to $\subseteq$, and this means that there is a vertex z in $N_{G'}(y')$ which is adjacent to every vertex in $V \setminus (N_{G'}(x) \cup \{x, y'\})$. Since all vertices in $N_{G'}(x) \setminus (N_{G'}(y') \cup \{y\})$ are adjacent to z (otherwise, there was an induced P_4

containing x, z, y'), z is a universal vertex in G', contradicting the assumption about G'.

We have shown that every graph that does not admit an elimination scheme that repeatedly deletes isolated, universal or false twin vertices contains a $2K_2$, P_4 or co-$(2P_3)$ as induced subgraph. Hence, $\{2K_2, P_4, \text{co-}(2P_3)\}$-free graphs admit such elimination schemes.

(1) $\Leftrightarrow$ (3) Let $G = (V, E)$ be a graph, and let $\sigma = \langle x_1, \ldots, x_n \rangle$ be a vertex ordering for G such that x_i is an isolated vertex, a universal vertex or a false twin vertex in $G_i =_{\text{def}} G[\{x_i, \ldots, x_n\}]$, $i \in \{1, \ldots, n\}$. Let σ be chosen such that false twin vertices are deleted earliest possible, which means that, if possible, a false twin vertex is picked instead of a universal or isolated vertex. Suppose there is $i \in \{1, \ldots, n-1\}$ such that x_{i+1} is a false twin vertex and x_i is not. Independent of whether x_i is a universal or isolated vertex in G_i, x_{i+1} is a false twin vertex also in G_i and x_i is a universal or isolated vertex in $G_i - x_{i+1}$. Hence, x_{i+1} can be deleted before x_i. We conclude that there is $j \in \{1, \ldots, n\}$ such that none of $x_j, \ldots, x_n$ is deleted as false twin vertex. Thus, G_j is isomorphic to $G/\sim_{\text{ft}}$ and therefore a threshold graph. Applying Theorem 5, we conclude that G is a simple cograph. The converse similarly follows from Theorem 5 and observing that the graph $G/\sim_{\text{ft}}$ is obtained from G by repeatedly deleting only false twin vertices. ■

The characterisation of simple cographs by forbidden induced subgraphs shows that the class of simple cographs is between the class of threshold graphs ($\{2K_2, P_4, C_4\}$-free) and the class of co-trivially perfect graphs ($\{2K_2, P_4\}$-free). Moreover by the result of [9] and Theorem 7 we conclude that simple cographs are exactly the graphs of linear clique-width at most 2. However in [9] no immediate proof is given as it is stated that it follows from arguments similar to another proof. For completeness we provide the proof of the next theorem.

Theorem 8. *A graph has linear clique-width at most 2 if and only if it is a simple cograph.*

Proof Our proof shows two implications using the groupwidth characterisation of linear clique-width. Let $G = (V, E)$ be an arbitrary graph. First, let G 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 $j \in \{1, \ldots, n\}$ such that $\nu_G(L_\beta(x_j)) + \text{ad}_\beta(x_j) \geq 3$, we are done. So, assume the contrary for the proof. Since G is not a simple cograph, there is $i \in \{2, \ldots, n-1\}$ 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 $\text{ad}_\beta(x_{i+1}) = 0$. Let A and A' be the two groups in $L_\beta(x_{i+1})$. Then, one of the two sets, say A, contains only neighbours and the other set only non-neighbours of x_{i+1}. From $\text{ad}_\beta(x_{i+1}) = 0$, it follows that the group of x_{i+1} in $L_\beta[x_{i+1}]$ contains the vertices of A' and that every vertex in A is adjacent to every vertex in A'. Thus, $G[L_\beta(x_{i+1})] = G[A] \otimes G[A']$. Let j be smallest such that $\{x_1, \ldots, x_j\}$ contains a vertex from A and A'. Since $x_1, \ldots, x_{j-1}$ and x_j are in different groups, $\nu_G(L_\beta(x_j)) = 1$ and $\text{ad}_\beta(x_j) = 1$. Since $\nu_G(L_\beta(x_{i'})) = 2$, $\text{ad}_\beta(x_{i'}) = 0$ for every $i' \in \{j+1, \ldots, i\}$. It follows that A or A' is an independent set. If A is an independent set, $G[\{x_1, \ldots, x_{i+1}\}] = (G[A'] \oplus G[\{x_{i+1}\}]) \otimes G[A]$ is a simple cograph, if A' is an independent set, $G[\{x_1, \ldots, x_{i+1}\}] = G[A] \otimes G[A' \cup \{x_{i+1}\}]$ is a simple cograph. This, however, contradicts the assumption about $G[\{x_1, \ldots, x_{i+1}\}]$ not being a simple cograph. Hence, $\text{gw}(G, \beta) \geq 3$.

For the converse, let G be a simple cograph. According to the definition, there are vertex-disjoint edgeless graphs $A_1, \ldots, A_s$ and operations $\odot_i \in \{\otimes, \oplus\}$, $i \in \{2, \ldots, s\}$, such that $G_1 =_{\text{def}}$

A_1 and $G_i =_{\text{def}} G_{i-1} \odot_i A_i$ for all $i \in \{2, \ldots, s\}$ and $G = G_s$. Let $x_1^i, \ldots, x_{r_i}^i$ be the vertices of A_i, $i \in \{1, \ldots, s\}$. We show that $\beta = \langle x_1^1, \ldots, x_{r_1}^1, x_1^2, \ldots, x_{r_s}^s \rangle$ is a layout for G of groupwidth at most 2. For every $i \in \{1, \ldots, s\}$, $L_\beta(x_1^i)$ has exactly one group, and since A_i is edgeless, $L_\beta(x_j^i)$ has at most two groups ($\{x_1^1, \ldots, x_{r_{i-1}}^{i-1}\}$ and $\{x_1^i, \ldots, x_{j-1}^i\}$), $j \in \{2, \ldots, r_i\}$. So, $\text{ad}_\beta(x_j^i) = 0$ for all $i \in \{1, \ldots, s\}$ and $j \in \{2, \ldots, r_i\}$, and we conclude $\text{lcwd}(G) \leq 2$ applying Theorem 1. ∎

6 Graphs of linear clique-width at most 3

Graphs of linear clique-width at most 2 were shown to have a quite simple structure, admitting also a number of different characterisations (Theorems 7 and 8). The situation changes already for graphs of linear clique-width at most 3. We give a characterisation of these graphs by a decomposition scheme, which will also lead into an efficient recognition algorithm. The decomposition scheme can be considered a generalisation of the decomposition scheme of simple cographs. We will apply this characterisation result to show that graphs of linear clique-width at most 3 are cocomparability graphs and weakly-chordal graphs.

Considering the definition of simple cographs as graphs admitting a certain decomposition using the join and union operation, we can say that simple cographs are defined in a "1-dimensional manner". For graphs of linear clique-width at most 3, we have to add another dimension. We first 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 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 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 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 $\odot \in \{\otimes, \oplus\}$.

This completes the definition of lc3-expressions. According to our intuition, parts (d2) and (d3) are "2-dimensional". As mentioned, lc3-expressions are defined for describing graphs. 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 $\odot \in \{\otimes, \oplus\}$, 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 join or a union (depending on $\odot$) 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 one, 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 of $A_1 \cup A_2 \cup A_3 \cup A_4$; the set of edges is dependent on p: we have three start sets, A_2, B, C, and three additional sets, A_3, A_1, A_4, that correspond to A_2, B, C, respectively; the first join is executed between the two specified start sets; then the two additional sets are added to the sets involved in the first join; the second join is executed between a now enlarged set and a start set; then the third additional set is added to its corresponding start set and the last join operation (if there is one) is executed; so the result depends on the order of the operations; the numbers stand for: 1 means $A_2 \otimes B$, 2 means $B \otimes C$, 3 means $A_2 \otimes C$; note that the start sets become bigger after each join operation; the vertex partition associated with $G(T)$ is $((B \cup A_1 \cup A_2 \cup A_3), (C \cup A_4))$ or $((B \cup A_1), (C \cup A_4 \cup A_2 \cup A_3))$ for $d = l$ or $d = r$, respectively.

(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.

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

We want to show that lc3-graphs are exactly the graphs of linear clique-width at most 3. The "difficult" part of this result show that every graph with a linear clique-width 3-expression is an lc3-graph. We partition the proof of this implication into several lemmata, that are interesting also on their own.

Lemma 9. *Let $G = (V, E)$ be an lc3-graph. Let T be an lc3-expression for G and let (B, C) be the vertex partition associated with $G(T)$. Let $S \subseteq V$. Then, $G[S]$ is an lc3-graph, and there is an lc3-expression T_S for $G[S]$ that associates $G(T_S)$ with vertex partition $(B \cap S, C \cap S)$ or $(C \cap S, B \cap S)$.*

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 obvious 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 $G-x$ and associates $G(T')$ with vertex partition $(B \setminus \{x\}, C \setminus \{x\})$. Iterated application of this operation proves the statement. ∎

Lemma 10. *Let T be an lc3-expression, and let (B, C) be the vertex partition associated with $G(T)$. The following two statements hold.*

(1) The subgraph of $G(T)$ induced by C is a simple cograph.

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

Proof We prove the two statements separately. Both proofs work on a given lc3-expression. We begin with statement (1). Let G_C denote the subgraph of $G(T)$ induced by C. If $T = (A)$ for some set A of vertices then G_C is an empty graph. Empty graphs are simple cographs. Let T' be an lc3-expression 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 is an edgeless graph on vertex set A or $A_3 \cup A_4$, respectively. If $T = T' \odot (A_1|A_2, \bullet)$ then C induces G_C already in $G(T')$, and we obtain the result by application of the induction hypothesis. Now, let $T = T' \circ (A_1, A_2|A_3, A_4, l[p])$. Let (B', C') be the vertex partition associated with $G(T')$. Then, G_C is equal to $G_{C'} \oplus G(T)[A_4]$, where $G_{C'}$ denotes the subgraph of $G(T)$ induced by C'. According to induction hypothesis, $G_{C'}$ is a simple cograph, and since $G(T)[A_4]$ is an edgeless graph, G_C is a simple cograph, too. Using the same definitions for the case $T = T' \odot (\bullet, A_1|A_2)$, G_C is equal to $G_{C'} \odot G(T)[A_1 \cup A_2]$, which is a simple cograph. Finally, let $T = T' \circ (A_1, A_2|A_3, A_4, r[p])$. Depending on p, G_C is one of the following graphs:

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

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

For statement (2), assume the contrary, 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 the two vertices 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 $G(T)$. 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. ∎

The 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)

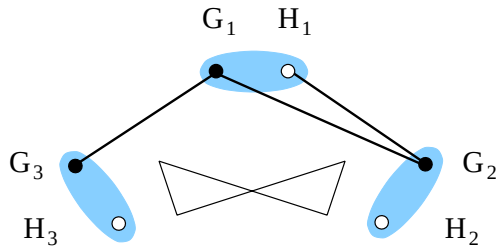


Figure 3: The result of an lc3-composition of the graphs $G_1, G_2, G_3, H_1, H_2, H_3$. The thick lines from graphs G_1 and H_1 represent joins. The bow tie between $G_2 \oplus H_2$ and $G_3 \oplus H_3$ means either a join or no edge at all.

and let H_1, H_2, H_3 be edgeless graphs. Graphs may be empty but at least two of them are non-empty. Then, the graph that is obtained from these six graphs and the additional edges as in Figure 3 is an lc3-graph. The bow tie in Figure 3 means either a join between $G_2 \oplus H_2$ and $G_3 \oplus H_3$ or no edge at all. We call this operation *complete* or *incomplete lc3-composition* depending on whether the bow tie represents a join operation ('complete') or a union operation ('incomplete').

Lemma 11. *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, where at least two of the six graphs are non-empty. Then, both the complete and the incomplete lc3-compositions of these graphs yield lc3-graphs. 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 associated with $G(T)$ is equal to $V(G_i) \cup V(H_i)$.*

Proof We consider complete and incomplete lc3-compositions 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 $G' \otimes G''$ and $G' \oplus G''$, respectively, such that $G(T')$ and $G(T'')$ are associated with the vertex partition $(V(G'), V(G''))$. Such lc3-expressions can be obtained from an lc3-expression T for G', starting from $(l[T], \emptyset)$ and adding operations of the forms $\otimes(\bullet, \emptyset|A)$ and $\oplus(\bullet, \emptyset|A)$ for $G' \otimes G''$ and $\otimes(\bullet, A|\emptyset)$ and $\oplus(\bullet, A|\emptyset)$ for $G' \oplus G''$. The proof of Lemma 10 describes such a construction in reverse. We distinguish the cases according to which graph is edgeless. 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 associated with $G(T)$ 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-compositions 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 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. ∎

Next we show that every lc3-graph admits a linear clique-width expression that uses at most 3 labels.

Lemma 12. *For an lc3-graph there exists a linear clique-width 3-expression.*

Proof Let G be an lc3-graph and let T be an lc3-expression such that $G(T) = G$. We inductively define a linear clique-width 3-expression for G. After every construction step, the linear clique-width 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 defined vertex partition (B, C), let a' be a linear clique-width 3-expression that defines $G(T')$ and has only two assigned labels such that the two label classes exactly correspond to (B, C). Let c_1 and c_2 be the labels in $G(T')$ corresponding 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. In the following, we will shorten an expression $c(x_1) \cdots c(x_n)$ to $c(X)$ where $X = \{x_1, \ldots, x_n\}$. Note that the order in which the vertices appear is not important. We 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 clique-width 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 an expression similar to the previous one, without the operation η_{c_1,c_3}. It is clear in both cases that a defines $G(T)$, that c_3 is not assigned in $G(T)$ and that c_1 and c_2 determine exactly the vertex partition $((B \cup A_1 \cup A_2), C)$, which is defined by T.

– 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 $p = 123$. The other cases are obtained in the same way. Let

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

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 clique-width 3-expression for G, which means that G has linear clique-width at most 3. ■

Before showing that lc3-graphs are exactly the graphs of linear clique-width at most 3, we first show a normalisation result for clique-width expressions. We use the following notation: Given a linear clique-width k-expression t that defines a graph G, we write $t = t_1 \cdots t_r$ if there are r clique-width operations involved in t. In other words we partition t into r subexpressions appearing consecutive in t. Note that the number of operations r must be at least as the number of labels k used in t. The graph $G[t_1, \ldots, t_i]$ is the subgraph of G induced by the vertices appearing in $t_1 \cdots t_i$, $1 \leq i \leq r$; note that $G[t] = G$. We say that vertices belong to the same label class of $G[t_1, \ldots, t_i]$ if they have the same label in $t_1 \cdots t_i$.

Lemma 13. *Let $k \geq 1$ and let G be a graph that has a linear clique-width k-expression. Then, G has a linear clique-width 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 clique-width 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 clique-width k-expression that has the first property. Let $d_i = \eta_{c,c'}$ be a join operation. If $G[d_1 \cdots d_i]$ does not contain a vertex with label c or c' then d_i does not add an edge to $G[d_1 \cdots, d_{i-1}]$, that is, $G[d_1 \cdots d_{i-1}] = G[d_1 \cdots d_i]$. 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 corresponding to c is empty then we obtain d' from d by just deleting operation d_i. If the class

corresponding to c is non-empty but the class corresponding to 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 clear that $G[d_1 \cdots d_j]$ and $G[d'_1 \cdots d'_{j-1}]$ correspond to each other for every $j \in \{i+1, \ldots, t\}$ with the exception that the classes corresponding to 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 14. *A graph has linear clique-width at most 3 if and only if it is an lc3-graph.*

Proof One direction follows from Lemma 12. For the converse, let G be a graph with 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 clique-width 3-expression for G that has the properties of Lemma 13. Without loss of generality, we can assume that a_1 adds a vertex. We will 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 $G_i =_{\text{def}} G[a_1 \cdots a_i]$ denote the labelled graph defined by $a_1 \cdots a_i$. If G is an edgeless graph, which means that a does not contain a 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 involving the same labels.

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 $T_1 =_{\text{def}} (A_1) \otimes (A_2 | \emptyset, \bullet)$. So, let A_3 be non-empty. Let D_1 be the set of vertices of A_1 having at least one neighbour in A_2 and A_3. Similarly, let D_2 be the set of vertices of A_2 having at least one neighbour in A_1 and A_3, and let D_3 be the set of vertices of A_3 having at least one neighbour in 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 3). 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 11, there is an lc3-expression for G_i that groups the vertices of G_i into $A_1 \cup A_2$ and A_3.

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) associated with $G(T_{t-1})$ corresponds to the labels 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, whose implied vertex partition corresponds to the labels in $G_{j'}$. The proof will be 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 $i \in \{1, 2, 3\}$. Note that since the vertices of A'_1, A'_2, A'_3 are added after operation a_j, 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 and $G_{j'}$ is a join of the subgraphs induced by A_1 and A_2.

Remember that $G_{j'}$ does not contain isolated vertices. According to Lemma 9, $G_{j'}[B]$ is an lc3-graph, and according to Lemma 10, $G_{j'}[C]$ 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 according to Lemma 11. 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 9, 10 and 11.

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. Let D_3 be the set of vertices of A'_3 with at least one neighbour in the sets A_1 and A_2. Then, one of the two lc3-expressions defines $G_{j'}$ and associates partition $(A_1 \cup A_3, A_2)$ with it:

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

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

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. Then we conclude the proof. ∎

Towards a better understanding of lc3-graphs, we give interesting connections between them and cocomparability graphs and weakly chordal graphs.

Proposition 15. *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 associated with $G(T)$. First, let $T = (A, \bullet)$ for a set A of vertices. Then, $G(T)$ is an edgeless graph, and every vertex sequence for G is a cocomparability ordering for G. Furthermore, every vertex sequence for G 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 associated vertex partition (B, C), let σ' be a cocomparability ordering for $G(T')$ that respects partition (B, C), which means that the vertices of B appear consecutively and the vertices of C appear consecutively in σ'. Let A, A_1, A_2, A_3, A_4 be sets of vertices, $d \in \{l, r\}$ and p an appropriate number sequence. 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 of A_1 to the left of the vertices of B and the vertices of A_2 between the vertices of B and C. Then, σ is a cocomparability ordering for G and the vertices of $B \cup A_1 \cup A_2$ 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 of C.

- $T = (l[T'], A)$. The vertices of A are adjacent to only the vertices of C. We obtain σ from σ' by adding the vertices in A to the right of the vertices of C. Then, σ is a cocomparability ordering for G that respects the partition $(B \cup C, A)$.

- The case $T = (r[T'], A_1, A_2)$ is similar to the previous case where A_1 is to the left of B and A_2 is to the left of A_1.

- $T = T' \circ (A_1, A_2|A_3, A_4, l[12])$. The vertices of B and A_2 are adjacent and the vertices of $B \cup A_1$ 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 $(A_1|A_2, d[T'], A_3|A_4)$. We begin with the situation in Figure 3. Suppose cocomparability orderings for the graphs $G_1, G_2, G_3, H_1, H_2, H_3$ are given. We define three vertex orderings for the depicted graph. The vertices of every partition graph appear consecutively and in order defined by the given orderings. The order of the partition graphs is as follows:

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

It is easy to check that all three vertex orderings actually define cocomparability orderings for the graph (scheme) depicted in Figure 3. 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 the 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 then completes the proof. ∎

For the next proof, a chordless cycle of length at least 5 is called a *hole*, and the complements of holes are called *anti-holes*.

Proposition 16. *Lc3-graphs are weakly-chordal graphs.*

Proof From Proposition 15 and the fact that holes are not cocomparability graphs [1], we already know that lc3-graphs are hole-free graphs. It remains to show that lc3-graphs do not contain anti-holes as induced subgraphs. Since the linear clique-width of a graph is not smaller than the linear clique-width of any of its induced subgraphs, it suffices to show by Theorem 14 that the linear clique-width of anti-holes is at least 4. We apply the layout characterisation of

linear clique-width given in Theorem 1. Let $k \geq 6$ (remember that $\overline{C_5} = C_5$). Let $\delta = \langle y_1, \ldots, y_k \rangle$ be a layout for $\overline{C_k}$. We distinguish two basic cases. First, let y_{k-1} and y_k be non-adjacent. Then, $L_\delta(y_{k-1})$ induces a path in the complement of $\overline{C_k}$. Furthermore, $L_\delta(y_{k-1})$ 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, $\nu_{\overline{C_k}}(L_\delta(y_{k-1})) = 3$. For the value of $\mathrm{ad}_\delta(y_{k-1})$, it suffices to observe that y_{k-1} is not single vertex in its group in $L_\delta[y_{k-1}]$ and that the other vertex in the group, say y, is a neighbour of y_{k-1}. Hence, $\mathrm{ad}_\delta(y_{k-1}) = 1$, and $\mathrm{gw}(C_k, \delta) \geq 4$.

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 $L_\delta(y_{k-1})$, and at most one vertex can be a common non-neighbour. We consider several subcases. Let y_{k-1} and y_k have a common non-neighbour. Then, $L_\delta(y_{k-1})$ has the following groups (note that this requires at least six vertices): 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, $\nu_{C_k}(L_\delta(y_{k-1})) = 4$. Now, let y_{k-1} and y_k not have a common non-neighbour in $L_\delta(y_{k-1})$. Let $k \geq 7$. Then, $L_\delta(y_{k-1})$ has three groups. Furthermore, y_{k-1} is not single vertex in its group in $L_\delta[y_{k-1}]$ and (at least) one of the two non-neighbours of y_{k-1} is non-adjacent to a neighbour of y_{k-1} in $L_\delta(y_{k-1})$. Thus, $\mathrm{ad}_\delta(y_{k-1}) = 1$, and $\mathrm{gw}(C_k, \delta) \geq 4$. Finally, let $k = 6$. Then, $L_\delta(y_{k-1})$ induces a C_4 in $\overline{C_k}$. Similar to the previous cases, $L_\delta(y_{k-2})$ has three groups and $\mathrm{ad}_\delta(y_{k-2}) = 1$, so that $\mathrm{gw}(\overline{C_k}, \delta) \geq 4$ also in this case. We have seen that for C_k, $k \geq 6$, and an arbitrary layout δ, $\mathrm{gw}(\overline{C_k}, \delta) \geq 4$, which establishes the lower bound of $\mathrm{gw}(\overline{C_k}) \geq 4$. Therefore, lc3-graphs are also anti-hole-free, and thus, weakly-chordal graphs. ∎

Note that holes and anti-holes have linear clique-width 4-expressions. Together with the result of Proposition 16, 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.

We only mention here that there are cocomparability graphs and weakly chordal graphs that are not lc3-graphs.

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

As the last main contribution of this paper, we give an efficient algorithm for recognising graphs of linear clique-width at most 3. The algorithm tries to recursively build an lc3-expression for the input graph. Here, we distinguish two different situations: a single graph is given or a graph and a vertex partition are given. Efficient recognition requires different approaches for the two situations. We begin with the case when a graph G and a vertex partition (B, C) for G are given. The question is whether there is an lc3-expression T for G that associates $G(T)$ with partition (B, C) or its reverse. An important subroutine is to decide whether the given graph is the result of an lc3-composition that respects the given partition. This problem can be solved in linear time, and it is our first result.

Lemma 17. *There is a linear-time algorithm that, on input a graph G and a vertex partition (B, C) for G, B and C non-empty, checks whether G 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 and*

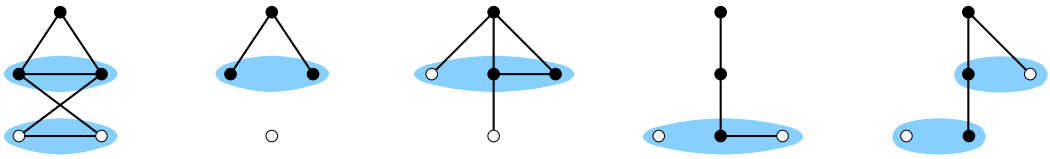


Figure 4: Different situations in a special case in the proof of Lemma 17. The first level corresponds to set B, the second and third level correspond to sets C_1 and C_2, respectively.

there is $i \in \{1,2,3\}$ such that $V(G_i) \cup V(H_i)$ is equal to one of the two partition sets. In the positive case, the algorithm can output an appropriate decomposition.

Proof The algorithm is not difficult but has to check a number of different cases. First, compute the groups in B and C only with respect to the other set of vertices; that is, sets of vertices of B (or C) having the same neighbours in C (or B). 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 both B and C have at least three groups, the algorithm rejects. Checking the three cases in Figure 3, we conclude: if a set has exactly two groups then these groups can be ordered such that the neighbourhood of the one group towards the other set is properly included in the neighbourhood of the other group. If this is not possible, the algorithm rejects. Note that, if both sets have exactly two groups, this inclusion property holds either for none or for both sets.

Now, assume that the algorithm has not yet rejected. Let exactly one set have exactly one group. Then, the other set has exactly two groups; without loss of generality, let C have two groups C_1, C_2 and let the join be between B and C_1 in G. 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 $V(G_i)$ and $V(H_i)$, respectively, 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 $G[C]$ is a simple cograph. So, accept if and only if $G[B]$ or $G[C]$ is a simple cograph. 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 to 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 scheme in Figure 3 by deleting an H-vertex. The obtained schemes are depicted in Figure 4. 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, the second level represents $G[C_1]$ and the last level represents $G[C_2]$. Note that a sixth case is missing; this case 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. Finally, let B be an independent set in G. Then, B can also correspond to $V(H_i)$. We obtain the different schemes by deleting full-circle vertices in Figure 3: two cases are not possible, since C_1 then must be empty, three cases turn out to be subcases of situations in Figure 4 (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 $G[B]$ is edgeless and $G[C_1]$ and $G[C_2]$ are a simple cograph and an arbitrary graph. This case can be handled in linear time.

As a second case, let both sets have exactly one group each. 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 or incomplete lc3-composition with four empty graphs where the two non-empty graphs go into G_2 and G_3 of Figure 3. 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 4), 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 or C have exactly two groups; without loss of generality, we assume that B has exactly two groups. If both sets have exactly two groups, we first try B and if B leads to rejection we try C. Let B_1 and B_2 be the two groups of B and let the neighbours of B_2 in C be neighbours of B_1. According to the definition of lc3-composition, B_2 is an independent set in G, and there is no edge between B_1 and B_2; otherwise the algorithm can reject (or has to retry with C). The algorithm then checks similar to the first case whether C admits an appropriate partition. This can be done in linear time for each case, so in total linear time. Hence, we can conclude the proof of the lemma. ∎

In a second step, we show that the question of whether a graph is an lc3-graph and whether it can be associated with a given vertex partition can be reduced to the question of whether a proper induced subgraph is an lc3-graph, without specifying a vertex partition. The algorithm is called SIMPLIFY and given in Figure 5. We apply this algorithm to only graphs with vertex partitions that do not have false twin vertices in the same partition set. The follwing technical lemma shows that this property is true throughout the execution of the algorithm.

Lemma 18. *Let $G = (V, E)$ be a graph, and let (B, C) be a vertex partition for G. Let G not have false twin vertices in B and in C. Then, at the beginning of every execution of the* **while***-loop of* SIMPLIFY*, G has no false twin vertices in the same partition set. Furthermore, an output graph does not contain false twin vertices.*

Proof The statement is clearly true at the beginning of the first execution. We first 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, $G-u$ cannot have two false twins in one partition set. 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 a module. Hence, false twin vertices in the output graph are false twin vertices also in the bigger graph, which would be a contradiction. ∎

Besides an answer or a graph, SIMPLIFY outputs lc3-expression operations and an additional command "turn". These operations can be used to construct an lc3-expression for the input graph. The next lemma states and proves the main properties of SIMPLIFY. The proof particularly shows that an lc3-expression can be constructed and how the "turn" command is used.

Lemma 19. *Let $G = (V, E)$ be a graph, and let (B, C) be a vertex partition for G. Let G not have false twin vertices in B and in C.*

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

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

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 "$\otimes(\emptyset|\{u\}, \bullet)$" or "$\otimes(\bullet, \emptyset|\{u\})$" or "$\otimes(\{u\}|\emptyset, \bullet)$" or "$\otimes(\bullet, \{u\}|\emptyset)$"
   "$\oplus(\emptyset|\{u\}, \bullet)$" or "$\oplus(\bullet, \emptyset|\{u\})$" or "$\oplus(\{u\}|\emptyset, \bullet)$" or "$\oplus(\bullet, \{u\}|\emptyset)$";
   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 $G := G - \{u, v\}$ and $(B, C) := (B \setminus \{u, v\}, C \setminus \{u, v\})$
5. **else-if** B or C is singleton set containing vertex 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.

Figure 5: The simplification procedure.

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

(3) If SIMPLIFY *applied to G and (B, C) returns a graph G' then G is an lc3-graph and there is an lc3-expression T for G that associates $G(T)$ with vertex partition (B, C) or (C, B) if and only if G' is an lc3-graph. Furthermore, G' is a module and subgraph of G that is induced by vertices only in B or in 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: the algorithm returns ACCEPT or a graph and the input graph is an lc3-graph with lc3-expression T that associates $G(T)$ with vertex partition (B, C) or (C, B).

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 that associates G 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 associated vertex partition $(V(G), \emptyset)$. If G is no lc3-graph, there is no lc3-expression for G. Thus, if B or C is empty, G is an lc3-graph with associated 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 associated vertex partition (B, C) or (C, B) if and only if the returned graph is an lc3-graph due to Lemmata 11 and 9. 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 9. 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 $G-u$ that associates $G(T')$ 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 associated vertex partition (B, C) or (C, B). The choice of l and r depends on whether the "turn" command is 'active' or 'inactive'. If the associated vertex partition is (B, C) then the "turn" command becomes 'inactive', otherwise 'active'. If G' is no lc3-graph, then G is no lc3-graph. This completes the first part of the proof.

Now, let G be an lc3-graph, and let T be an lc3-expression for G that associates $G(T)$ 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 9. 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 9 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 $\odot \in \{\otimes, \oplus\}$. It is important to note that an operation $\oplus(A_1|A_2, \bullet)$ can be partitioned into two operations, $\oplus(A_1|\emptyset, \bullet) \oplus (\emptyset|A_2, \bullet)$. So, let the last operation be of the form $(d[T'], A)$ for T' an lc3-expression and $d \in \{l, r\}$. Since G does not contain false twin vertices in the same partition set, 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 associated with $G(T')$. Then, $N_G(u) = B'$ or $N_G(u) = C'$, which means that $G-u$ is an lc3-graph and has an lc3-expression that associates 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 10. This is a contradiction, and we complete this 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 11. So, assume that either B or C contains exactly one vertex; let this vertex be u. 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 that

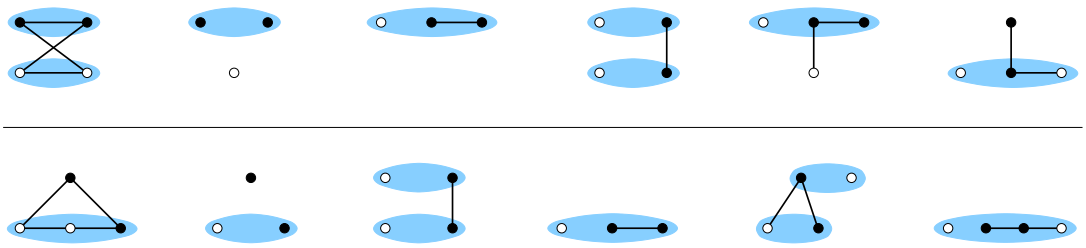


Figure 6: Subcases of the general lc3-composition, considered in the proof of Lemma 19.

associates partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse with $G-u$. Let $G_1, G_2, G_3, H_1, H_2, H_3$ be the composition graphs in the sense of Figure 3, 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 and H_i with respect to the vertex partition $(N_G(u), V(G) \setminus N_G[u])$. In Figure 6, we find all twelve cases 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 $G-N_G[u]$. By checking every case, we finally see that every case except one can be obtained from an appropriate lc3-composition respecting the new vertex partition. The most interesting case probably is the first case in the second row. This case 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 union with edgeless graphs. The cases with only one level cannot happen since u then would be a universal or an isolated vertex. In the 'good' cases we apply Lemma 11 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 given in Figure 7. The applied procedure SimplifyMod is a variant of Simplify, with the only difference that it outputs 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

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.

set G **:=** $G/\sim_{\mathrm{ft}}$;

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 the degree of v is $|V(G)| - 2$ such that
   SimplifyMod on $G - \{u, v\}$ 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.

Figure 7: The recognition algorithm for lc3-graphs.

the graphs of linear clique-width at most 3.

Theorem 20. *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 14, a graph G is an lc3-graph if and only if $G/\sim_{\mathrm{ft}}$ is an lc3-graph. Hence, we can restrict to consider only graphs without non-trivial 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 proof the statement by induction over the number of vertices of the input graph. A graph 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. (Otherwise G would contain false twin vertices, contradicting the assumption that G does not contain non-trivial independent-set modules.) 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 9. If G is not an lc3-graph, then G' cannot be an lc3-graph due to Lemma 11. 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 19. By induction hypothesis, $G-u$ is an

lc3-graph, and due to Lemma 19, there is an lc3-expression T for $G-u$ that associates $G(T)$ 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 does not contain universal or isolated vertices; 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' that associates $G(T')$ with the vertex partition $(N_G(u), V(G) \setminus N_G[u])$ or its reverse. Due to Lemma 19, SimplifyMod outputs 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 associated with $G(T)-u$. 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. 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 will 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 17. We show next that Simplify has a linear-time implementation.

Lemma 21. *The 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 only in 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 17 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 20. Care has to be taken of only the "turn" command. This completes the proof. ∎

As a second step, we show that there is a linear-time algorithm for checking whether a graph is the result of an lc3-composition.

Lemma 22. *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 can output 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 3 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 3. 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 3 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 3 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. 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, too. 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 too, 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 3.

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 3), 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 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. ∎

We combine the three subresults to the following main result of this section.

Theorem 23. *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 clique-width 3-expression is output.*

Proof By Theorem 14 and Theorem 20 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 result of an lc3-composition (Lemma 22) 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 21. This shows the total $\mathcal{O}(n^2m)$ running time. The linear clique-width 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 14. Note that the length of both expressions is linear in the number of vertices. ∎

8 Final remarks

We have introduced the class of lc3-graphs in order to characterise graphs of linear clique-width at most 3. We have shown that such graphs are cocomparability and weakly-chordal graphs, which provides a set of graphs that are not induced subgraphs of lc3-graphs. In order to better understand graphs of linear clique-width at most 3 and even graphs of clique-width at most 3, a complete characterisation of lc3-graphs by a set of forbidden subgraphs is an important future goal.

References

[1] A. Brandstädt, V.B. Le, and 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, and U. Rotics. Polynomial time recognition of clique-width ≤ 3 graphs. Proceedings of Latin American Theoretical Informatics, LATIN 2000, LNCS 1776, pp. 126–134, 2000.

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

[4] B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. J. Comput. System Sci. 46, pp. 218–270, 1993.

[5] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. Theory Comput. Syst. 33, pp. 125–150, 2000.

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

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

[8] M.R. Fellows, F.A. Rosamond, U. Rotics, and S. Szeider. Clique-width Minimization is NP-hard. Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing, STOC'06, pp. 354–362, 2006.

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

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

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

[12] P. Hliněný, S. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. Computer Journal, to appear.

[13] D. Kratsch and L. Stewart. Domination on cocomparability graphs. SIAM Journal on Discrete Mathematics 6, pp. 400-417, 1993.

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