A characterisation of clique-width through nested partitions

Bruno Courcelle* Pinar Heggernes† Daniel Meister‡ Charis Papadopoulos§
Udi Rotics¶

Abstract

Clique-width of graphs is defined algebraically through operations on graphs with vertex labels. We characterise the clique-width in a combinatorial way by means of partitions of the vertex set, using trees of nested partitions where partitions are ordered bottom-up by refinement. We show that the correspondences in both directions, between combinatorial partition trees and algebraic terms, preserve the tree structures and that they are computable in polynomial time. We apply our characterisation to linear clique-width. And we relate our characterisation to a clique-width characterisation by Heule and Szeider that is used to reduce the clique-width decision problem to a satisfiability problem.

1 Introduction

Hierarchical graph decompositions are useful for the design of efficient graph algorithms. This usefulness is the foundation of the theory of fixed-parameter tractability, that emerged from the large body of graph algorithms working on tree decompositions [8]. There are many graph decompositions with interesting algorithmic applications, such as modular decomposition [12], tree decomposition [8, 10], and rank decomposition [19]. In most cases, a *width* notion is associated with a graph decomposition, and the width of a graph is the minimum achievable width for the graph using this decomposition. Fixed-parameter tractability results are often of the form: a graph problem is tractable on graphs of bounded width, and the width of the input graph is a measure for the complexity of determining the solution.

Modular decompositon, tree decomposition and rank decomposition can be defined combinatorially. Tree decomposition and rank decomposition can be defined also algebraically [3, 4]. An algebraic definition is built on operations from an appropriate graph algebra. Algebraic descriptions yield clean definitions of finite automata, that themselves can be seen as abstract descriptions of efficient parametrised algorithms (see [2] or Chapter 6 of [3] for a deeper treatment of the matter).

Clique-width is a graph complexity measure [3, 5, 7]. The original definition of clique-width is by means of algebraic terms. These terms use operations that create edgeless graphs, that add

*LaBRI, CNRS, 351 Cours de la Libération, 33405 Talence, France. Email: courcell@labri.fr
†Department of Informatics, University of Bergen, Norway. Email: pinar.heggernes@ii.uib.no
‡Theoretical Computer Science, University of Trier, Germany. Email: daniel.meister@uni-trier.de
§Department of Mathematics, University of Ioannina, Greece. Email: charis@cs.uoi.gr
¶Netanya Academic College, Netanya, Israel. Email: rotics@netanya.ac.il

edges between specified vertices, and that combine graphs. Clique-width is a practically useful graph measure, because the results by Courcelle et al. in [6] in combination with the results by Oum and Seymour in [19] show that the model-checking problem for monadic second-order logic on graphs is fixed-parameter tractable when choosing the clique-width of the input graph as the parameter (see the monographs by Downey and Fellows [8] and by Flum and Grohe [10] for a deep and comprehensive consideration of fixed-parameter tractability).

The algebraic definition of clique-width, the same as similar characterisations of treewidth and rank-width, use vertex labels to specify classes of vertices. We can say that vertex labels express a "neighbourhood similarity" relation, since vertices with the same label have the same "one-sided" neighbourhood. The clique-width of a graph is the smallest number of necessary different labels in a clique-width expression. Computing the clique-width of a graph is hard [9], and the problem of choosing appropriate labels for vertices is a strong contribution to the hardness of the problem [18]. Combinatorial characterisations avoid labels and the necessity of choosing labels.

In this paper, we give a purely combinatorial characterisation of clique-width. Our characterisation uses rooted trees and nested partitions of sets of vertices. The partitions are ordered by the refinement relation, and they satisfy an adjacency condition. The resulting notion is *partition tree* for graphs. With partition trees, we associate a *width* notion, that measures the size of the nested partitions. In Section 3, we introduce partition trees and describe their relationship to clique-width. As the main result of this paper, we obtain a characterisation of clique-width by partition trees, that shows the equivalence between the clique-width of a graph and the width of its partition trees. Our results also show and provide efficient transformation algorithms between algebraic clique-width expressions and partition trees, and we provide a space-efficient representation of partition trees, that is of interest for efficient algorithms. Our combinatorial characterisation of clique-width is based on, combines, and extends combinatorial clique-width characterisations by Heggernes et al. [15] and Heule and Szeider [16].

Tree-based graph decompositions have linear variants: pathwidth is the linear variant of treewidth, linear rank-width is the linear variant of rank-width, and linear clique-width is the linear variant of clique-width. The linear variants are obtained when restricting the decomposition tree to a path-like tree, a *caterpillar*, more precisely. In Section 4, we apply our clique-width characterisation result to linear clique-width, obtaining an analogue combinatorial characterisation of linear clique-width. We also study combinatorial properties of our partition trees, that provide an alternative approach to corresponding known results.

Finally, in Section 5, we review the clique-width characterisation by Heule and Szeider [16]. This characterisation is based on a sequence of pairs of partitions of the vertex set of graphs. We relate this partition-sequence characterisation to our partition-tree characterisation, show the equivalence of the two characterisations, and also provide efficient transformation algorithms.

2 Graph preliminaries and clique-width

Graph preliminaries. The studied graphs in this paper are simple, finite, undirected. In addition, we use directed graphs and rooted trees for the representation of information about undirected graphs. All definitions, if not otherwise said, are for undirected graphs. For rooted trees and

directed graphs, we use only standard terminology, that we do not always define explicitly.

A graph G is an ordered pair (V, E) where $V = V(G)$ is the *vertex set* and $E = E(G)$ is the *edge set* of G. Edges are denoted as uv. Let u, v be a vertex pair of G with $u \neq v$. If uv is an edge of G then u and v are *adjacent* in G, and u is a *neighbour* of v in G; if uv is not an edge of G then u and v are *non-adjacent* in G. The *neighbourhood* of u in G, $N_G(u)$, is the set of neighbours of u in G. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For $X \subseteq V(G)$, the subgraph of G *induced* by X, $G[X]$, has X as its vertex set, and for each vertex pair u, v from X, uv is an edge of $G[X]$ if and only if $u \neq v$ and uv is an edge of G. For $R \subseteq E(G)$, $G \setminus R$ denotes the graph $(V(G), E(G) \setminus R)$, that is a subgraph of G. For $X, Y \subseteq V(G)$, where $X \cap Y = \emptyset$, we denote by $X \times Y$ the set of all possible edges between vertices in X and vertices in Y, that is $X \times Y = \{xy : x \in X \text{ and } y \in Y\}$.

An important graph operation throughout the paper is the disjoint union of graphs. Let $H_1, \ldots, H_p$, where $p \geq 2$, be pairwise vertex-disjoint graphs, which means $V(H_i) \cap V(H_j) = \emptyset$ for every $1 \leq i < j \leq p$. The *disjoint union* of $H_1, \ldots, H_p$, denoted as $\bigoplus(H_1, \ldots, H_p)$, is the graph $\big(V(H_1) \cup \cdots \cup V(H_p), E(H_1) \cup \cdots \cup E(H_p)\big)$.

Clique-width and linear clique-width. Let $k \geq 1$. A *k-labelled graph* is an ordered pair (G, ℓ) where G is a graph and $\ell : V(G) \to \{1, \ldots, k\}$ is a mapping, that assigns a label from $\{1, \ldots, k\}$ to every vertex of G. The *vertices* and *edges* of (G, ℓ) are the vertices and edges of G. For (G, ℓ) a k-labelled graph, $(G, \ell)^\circ$ denotes the underlying graph G without the labels, and ℓ is the *label function* of (G, ℓ).

Consider the following inductive definition of k-expressions and linear k-expressions, slightly generalising the original definition in [5]:

- for $o \in \{1, \ldots, k\}$ and u a vertex name, $o(u)$ is a k-expression and a linear k-expression
- for δ a k-expression and $s, o \in \{1, \ldots, k\}$ with $s \neq o$, $\eta_{s,o}(\delta)$ and $\rho_{s \to o}(\delta)$ are k-expressions; if δ is a linear k-expression then $\eta_{s,o}(\delta)$ and $\rho_{s \to o}(\delta)$ are linear k-expressions
- for $p \geq 2$, and for $\delta_1, \ldots, \delta_p$ pairwise vertex-disjoint k-expressions, $\bigoplus(\delta_1, \ldots, \delta_p)$ is a k-expression;
  if δ_1 is a linear k-expression and $\delta_2 = o_2(u_2), \ldots, \delta_p = o_p(u_p)$ then $\bigoplus(\delta_1, \ldots, \delta_p)$ is a linear k-expression.

We call two k-expressions *vertex-disjoint* if the vertex names occurring in the two k-expressions are pairwise different. Note that a vertex name occurring in a k-expression occurs exactly once.

The graphs defined by k-expressions are defined inductively. Let α be a k-expression. The labelled graph *defined* by α is denoted as $\mathrm{val}(\alpha)$, and it is defined as follows:

- if $\alpha = o(u)$ then $\mathrm{val}(\alpha)$ is the k-labelled graph with the single vertex u and u has label o
- if $\alpha = \eta_{s,o}(\delta)$ then $\mathrm{val}(\alpha)$ is obtained from $\mathrm{val}(\delta)$ by adding all missing edges between vertices with label s and vertices with label o, and

  if $\alpha = \rho_{s \to o}(\delta)$ then $\mathrm{val}(\alpha)$ is obtained from $\mathrm{val}(\delta)$ by replacing every occurring label s by label o
- if $\alpha = \bigoplus(\delta_1, \ldots, \delta_p)$ then $\mathrm{val}(\alpha)$ is obtained as the disjoint union of $\mathrm{val}(\delta_1), \ldots, \mathrm{val}(\delta_p)$.

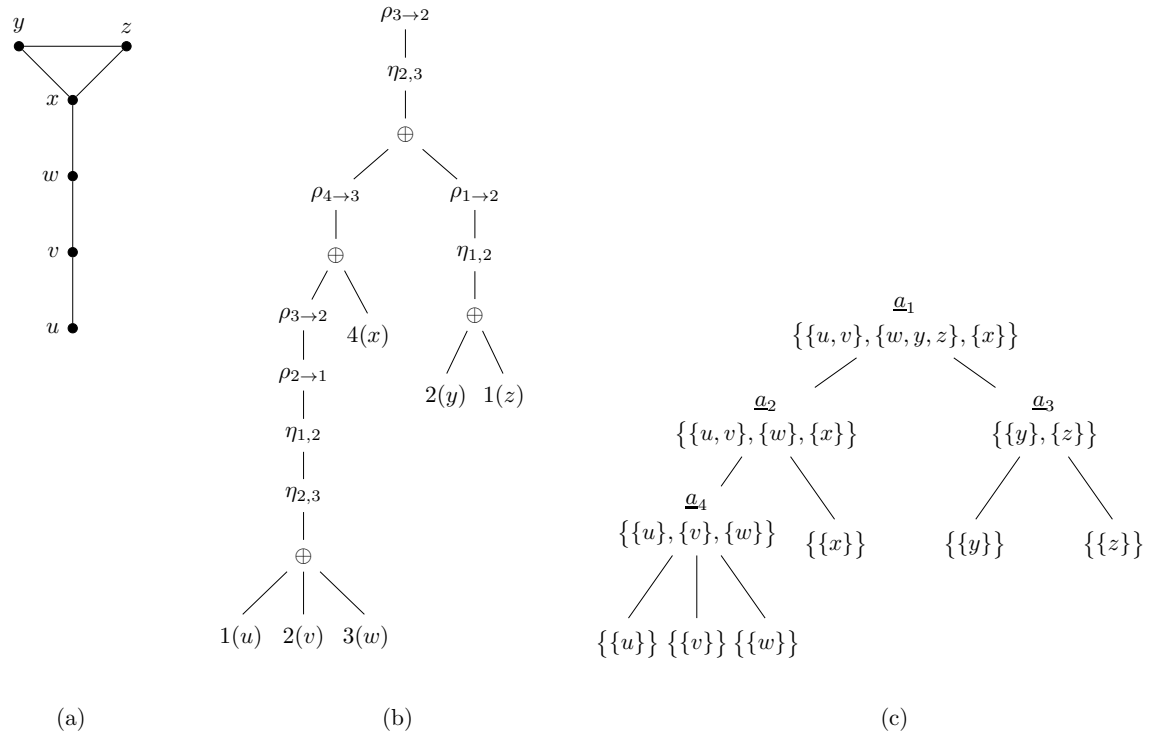

Figure 1: (a) our example graph G, (b) a 4-expression for G, that is given on its syntactic tree, and (c) a partition tree (T, f) for G of width 3 with its assigned partial partition labels $f(\underline{a}_1), \ldots, f(\underline{a}_4)$ for the inner nodes and $f(\underline{a}_u), \ldots, f(\underline{a}_z)$ at the leaves.

The disjoint union of k-labelled graphs is obtained from the disjoint union of the underlying graphs and the union of the label functions. The underlying graph without labels defined by α, that is $(\mathrm{val}(\alpha))^\circ$, is denoted shortly as $\mathrm{val}^\circ(\alpha)$. We say that α is a k-expression for a graph G if $G = \mathrm{val}^\circ(\alpha)$. If α is a k-expression for G and α is a linear k-expression, we say that α is a linear k-expression for G.

Let G be a graph. The *clique-width* of G, $\mathrm{cwd}(G)$, is the smallest integer k such that G has a k-expression. And the *linear clique-width* of G, $\mathrm{lcwd}(G)$, is the smallest integer k such that G has a linear k-expression. Recall from [5] that clique-width and linear clique-width are originally defined using only the binary restriction of $\bigoplus$. It is straightforward to verify that our generalisation here to arbitrary arity of $\bigoplus$ does not alter the clique-width or linear clique-width of a graph.

The goal of our work is to give a combinatorial characterisation of clique-width. The definition of clique-width is algebraic, and it represents graphs, in the meaning of *α is a k-expression for G*, by algebraic terms. Terms have a tree structure. As an example, consider the two left-side figures of Figure 1: figure (a) depicts a graph G on six vertices, and figure (b) shows a 4-expression for G, that is shown in its tree structure. This tree structure is often called the *syntactic tree* of the term. Note that G itself is not a labelled graph.

For the central characterisation of clique-width in this paper, that we present in the next

section, we recall these definitions. Let M be a set, and let $\mathcal{F}$ be a family of subsets of M. If the members of $\mathcal{F}$ are non-empty and pairwise disjoint then $\mathcal{F}$ is a *partial partition* of M. If $\mathcal{F}$ is a partial partition of M and if the members of $\mathcal{F}$ cover M, i.e., if $M = \bigcup_{F\in\mathcal{F}} F$, then $\mathcal{F}$ is a *partition* of M. Let $\mathcal{F}$ and $\mathcal{F}'$ be partial partitions of M. We say that $\mathcal{F}'$ *refines* $\mathcal{F}$, $\mathcal{F}' \sqsubseteq \mathcal{F}$, if for every $F' \in \mathcal{F}'$, there is $F \in \mathcal{F}$ such that $F' \subseteq F$, and $\bigcup_{F'\in\mathcal{F}'} F' = \bigcup_{F\in\mathcal{F}} F$.

For our results of Section 5, we need this restricted refinement notion. We say that $\mathcal{F}'$ *strongly refines* $\mathcal{F}$, $\mathcal{F}' \sqsubseteq_{\mathrm{str}} \mathcal{F}$, if $\mathcal{F}'$ refines $\mathcal{F}$ and additionally for every $F \in \mathcal{F}$, $F \in \mathcal{F}'$ implies $|F| = 1$. We can say that $\mathcal{F}'$ strongly refines $\mathcal{F}$ if $\mathcal{F}' \sqsubseteq \mathcal{F}$ and each non-trivial partition class of $\mathcal{F}$ is the union of at least two partition classes of $\mathcal{F}'$.

3 Partition trees and clique-width

Trees are connected acyclic graphs. The vertices of trees are called *nodes*, that we denote as $\underline{a}$. We consider rooted trees, and we distinguish between *inner nodes*, that are nodes with *children*, and *leaves*, that are nodes without children. We use rooted trees to represent partitions of sets.

Definition 3.1 (Partition trees for sets)**.** *Let S be a non-empty finite set. A* partition tree *for S is an ordered pair (T, f) where T is a rooted tree whose inner nodes have at least two children and f is an assignment of partial partitions of S to the nodes of T that satisfies the following two conditions, where for a node $\underline{a}$ of T, $\overline{f}(\underline{a})$ denotes the union of the members of $f(\underline{a})$:*

1) for every $x \in S$, there is a leaf $\underline{a}$ of T such that $f(\underline{a}) = \big\{\{x\}\big\}$

2) for every inner node $\underline{a}$ of T with $\underline{b}_1, \ldots, \underline{b}_p$ its children, where $p \geq 2$, $\overline{f}(\underline{b}_1), \ldots, \overline{f}(\underline{b}_p)$ are pairwise disjoint and $f(\underline{b}_1) \cup \cdots \cup f(\underline{b}_p) \sqsubseteq f(\underline{a})$.

The size *of (T, f) is the number of nodes of T, and the* width *of (T, f) is the maximum cardinality $|f(\underline{a})|$ taken over the nodes $\underline{a}$ of T.*

The following observations are easy consequences of the properties of partial partitions. Let (T, f) be a partition tree for a non-empty finite set S. For every leaf $\underline{a}$ of T, there is $x \in S$ such that $f(\underline{a}) = \big\{\{x\}\big\}$. In fact, f when restricted to the leaves of T establishes a bijection between the leaves of T and the elements of S. Then notice that the size of (T, f) is at most $2|S|$ since every node of T has at least two children. Also, for $\underline{r}$ the root of T, $f(\underline{r})$ is a partition of S.

We extend the definition of partition trees for sets to partition trees for labelled graphs, and subsequently to partition trees for graphs. Let (G, ℓ) be a labelled graph. The *label relation* of (G, ℓ) relates the vertices of G with the same label. The label relation is an equivalence relation on $V(G)$. The set of the equivalence classes of the label relation of (G, ℓ) is the *label partition* of (G, ℓ).

Definition 3.2 (Partition trees for labelled graphs)**.** *Let (G, ℓ) be a labelled graph. A* partition tree *for (G, ℓ) is a partition tree (T, f) for $V(G)$ that satisfies the two conditions:*

1) Compatibility condition
for every inner node $\underline{a}$ of T with $\underline{b}_1, \ldots, \underline{b}_p$ its children: for every adjacent vertex pair x, y of G and every index pair i, j where $i \neq j$ and every pair X, Y of members of $f(\underline{a})$, if $x \in X$, $y \in Y$, $x \in \overline{f}(\underline{b}_i)$, and $y \in \overline{f}(\underline{b}_j)$ then $X \neq Y$ and $X \times Y \subseteq E(G)$

2) for $\underline{r}$ the root of T, $f(\underline{r})$ refines the label partition of (G, ℓ).

Definition 3.3 (Partition trees for graphs)**.** *Let G be a graph. A* partition tree *for G is a partition tree for (G, ℓ) where $\ell : V(G) \to \{1, \ldots, k\}, x \mapsto 1$.*

It is important to note that a partition tree for a graph does not *represent* the graph. This is the case, since the adjacency relation is not represented in the partition tree. Furthermore, in the definition of partition trees, we could allow a node $\underline{a}$ to have only one child $\underline{b}$ and require that $f(\underline{b})$ refines $f(\underline{a})$. Such a node $\underline{a}$ can be eliminated by contracting the edge to $\underline{b}$. Performing all such contractions yields a partition tree as in Definition 3.1.

The objective of this section is to show that partition trees of width at most k are equivalent to k-expressions, that is to show that a graph has a partition tree of width at most k if and only if it has a k-expression, and thus, it is a graph of clique-width at most k. Before we proceed, we consider the graph G of Figure 1 again. The right-side figure of Figure 1 shows a partition tree for $V(G)$: the two conditions of Definition 3.1 are verified straightforward. Additionally, the partition tree also satisfies the Compatibility condition of Definition 3.2. As an example, consider node $\underline{a}_1$ and the edge xy: $x \in \overline{f}(\underline{a}_2)$ and $y \in \overline{f}(\underline{a}_3)$, and x and y are contained in different members, say X and Y, of $f(\underline{a}_1)$. The Compatibility condition requires $X \times Y \subseteq V(G)$, that is indeed the case because of $X = \{x\}$ and $Y = \{w, y, z\}$ and $\{xw, xy, xz\} \subseteq E(G)$.

We want to show the equivalence between partition trees and expressions. We also want to show that the one can be obtained from the other by an efficient transformation algorithm. Since straightforward representations of partition trees contain much redundancy, we devise a data structure for partition trees that reduces redundancy and is of linear space for partition trees of bounded width.

Definition 3.4 (Representation graphs of partition trees for sets)**.** *Let S be a non-empty finite set. Let (T, f) be a partition tree for S. The* representation graph *of (T, f) is the directed graph D with the following vertex and arc set:*

1) the vertex set of D is $V(T) \cup S \cup \bigcup_{\underline{a} \in V(T)} \{(\underline{a}, Y) : \ Y \in f(\underline{a})\}$

2) the arc set of D consists of the arcs of the following four types

- *type 0: $\underline{a} \to \underline{b}$ where $\underline{a}$ is an inner node of T and $\underline{b}$ is a child of $\underline{a}$*
- *type 1: $\underline{a} \to (\underline{a}, Y)$ where $\underline{a}$ is a node of T and $Y \in f(\underline{a})$*
- *type 2: $(\underline{a}, Y) \to (\underline{b}, Z)$ where $\underline{a}$ is an inner node of T, $\underline{b}$ is a child of $\underline{a}$, and $Z \subseteq Y$*
- *type 3: $(\underline{a}, \{x\}) \to x$ where $\underline{a}$ is a leaf of T.*

Definition 3.5 (Representation graphs of partition trees for labelled graphs)**.** *Let (G, ℓ) be a labelled graph. Let (T, f) be a partition tree for (G, ℓ). The* representation graph *of (T, f) is a partially labelled directed graph (D, h) where D is the representation graph of (T, f) for $V(G)$ and h is a partial labelling for the vertices of D such that, for $\underline{r}$ the root of T, the vertices $(\underline{r}, Y)$ of D are labelled by h with $\ell(y)$ for $y \in Y$.*

Observe that the label $\ell(y)$ of a vertex $(\underline{r}, Y)$ is well-defined in Definition 3.5, since all vertices in Y have the same label in (G, ℓ) according to the second condition of Definition 3.2.

Also observe that $(\underline{r}, Y)$ and $(\underline{r}, Y')$ for $Y \neq Y'$ can have the same label, which is the case if $f(\underline{r})$ is a proper refinement of the label partition of (G, ℓ).

Representation graphs offer an efficient and succinct representation of partition trees and of the assignment function f. To see the improvement on the succinctness of the representation, consider a labelled graph (G, ℓ) on n vertices, and let (T, f) be a partition tree for (G, ℓ). A straightforward representation – or *implementation* – of (T, f) lists $f(\underline{a})$ for each node $\underline{a}$ of T, which requires $\mathcal{O}(n)$ space, and results in an $\mathcal{O}(n^2)$-space representation in total. This space is independent of the width of (T, f). We use representation graphs to benefit from a small width of the partition tree: if (T, f) is of width at most k and T has p nodes then the representation graph of (T, f) has at most $p+n+p\cdot k \leq p(2+k)$ vertices and at most $(p-1)+(p\cdot k)+(p\cdot k)+n \leq 2p(1+k)$ arcs. Since p is of order n, we can consider such a representation as very succinct.

We show that efficient transformations exist between expressions and representation graphs of partition trees. For $k \geq 1$, a *width-k partition tree* is a partition tree of width at most k.

Proposition 3.6. *There is an algorithm that transforms a k-expression α into the representation graph of a width-k partition tree for* $\mathrm{val}(\alpha)$ *in time $\mathcal{O}(k \cdot |\alpha|)$, where $|\alpha|$ is the size of α.*

Proof. The algorithm constructs a partition tree and its representation graph recursively, according to the inductive definition of k-expressions.

Operation 1: $\alpha = o(u)$

Observe that $\mathrm{val}(\alpha)$ is a labelled graph on a single vertex. A width-k partition tree for $\mathrm{val}(\alpha)$ of size 1 and its representation graph are constructed straightforward. □

Operation 2: $\alpha = \eta_{s,o}(\delta)$

Let (T, f) be a partition tree for $\mathrm{val}(\delta)$. We show that (T, f) is a partition tree also for $\mathrm{val}(\alpha)$. It suffices to verify the Compatibility condition. Let $\underline{a}$ be an inner node of T, and let $\underline{b}$ and $\underline{c}$ be two children of $\underline{a}$, where $\underline{b} \neq \underline{c}$. Let x, y be an adjacent vertex pair of $\mathrm{val}(\alpha)$ and assume $x \in \overline{f}(\underline{b})$ and $y \in \overline{f}(\underline{c})$. Let $X, Y \in f(\underline{a})$ and assume $x \in X$ and $y \in Y$. If x, y is an adjacent vertex pair of $\mathrm{val}(\delta)$ then $X \times Y \subseteq E(\mathrm{val}^\circ(\delta)) \subseteq E(\mathrm{val}^\circ(\alpha))$.

Otherwise, $xy \notin E(\mathrm{val}^\circ(\delta))$, and we can assume without loss of generality that x has label s and y has label o in $\mathrm{val}(\alpha)$ and $\mathrm{val}(\delta)$. Let L_s and L_o be the sets of the vertices of $\mathrm{val}(\alpha)$ with label s and with label o, respectively. Observe: $L_s \times L_o \subseteq E(\mathrm{val}^\circ(\alpha))$. So, if $X \subseteq L_s$ and $Y \subseteq L_o$ then $X \times Y \subseteq E(\mathrm{val}^\circ(\alpha))$, as is required. For $\underline{r}$ the root of T, there exist $X', Y' \in f(\underline{r})$ such that $X \subseteq X'$ and $Y \subseteq Y'$, by recursively applying the second condition of Definition 3.1. Since $f(\underline{r})$ refines the label partition of $\mathrm{val}(\delta)$ and thus of $\mathrm{val}(\alpha)$, it follows that $X' \subseteq L_s$ and $Y' \subseteq L_o$, which implies that $X \subseteq L_s$ and $Y \subseteq L_o$ is indeed the case.

Since each partition tree for $\mathrm{val}(\delta)$ is a partition tree for $\mathrm{val}(\alpha)$ and since $\mathrm{val}(\delta)$ and $\mathrm{val}(\alpha)$ have the same label partitions, the representation graph for (T, f) of $\mathrm{val}(\delta)$ is the representation graph for (T, f) of $\mathrm{val}(\alpha)$. □

Operation 3: $\alpha = \rho_{s\to o}(\delta)$

Let (T, f) be a partition tree for $\mathrm{val}(\delta)$. Since $\mathrm{val}^\circ(\delta) = \mathrm{val}^\circ(\alpha)$, (T, f) is also a partition tree for $\mathrm{val}(\alpha)$, especially since the label partition of $\mathrm{val}(\delta)$ refines the label partition of $\mathrm{val}(\alpha)$.

Let (D, h) be the representation graph of (T, f) for $\mathrm{val}(\delta)$. Let h' be obtained from h as follows, for every $x \in V(D)$: if $h(x) \neq s$ then $h'(x) =_{\mathrm{def}} h(x)$, and if $h(x) = s$ then $h'(x) =_{\mathrm{def}} o$. It is straightforward to observe that (D, h') is the representation graph of (T, f) for $\mathrm{val}(\alpha)$. □

Operation 4: $\alpha = \bigoplus(\delta_1, \ldots, \delta_p)$ where $p \geq 2$

Let $(T_1, f_1), \ldots, (T_p, f_p)$ be width-k partition trees for respectively $\mathrm{val}(\delta_1), \ldots, \mathrm{val}(\delta_p)$. Let $\underline{r}_1, \ldots, \underline{r}_p$ be the roots of respectively $T_1, \ldots, T_p$. Let $\underline{r}$ be a new node. We obtain T from the disjoint union of $T_1, \ldots, T_p$ by adding $\underline{r}$ as the root and making $\underline{r}$ adjacent to $\underline{r}_1, \ldots, \underline{r}_p$. Let f for T be defined as follows. For every node $\underline{a}$ of T, if $\underline{a}$ is a node of T_i, where $1 \leq i \leq p$, then $f(\underline{a}) =_{\text{def}} f_i(\underline{a})$, and if $\underline{a}$ is not a node of any of $T_1, \ldots, T_p$ then $\underline{a} = \underline{r}$ and we let $f(\underline{a})$ be the label partition of $\mathrm{val}(\alpha)$.

We verify that (T, f) is a partition tree for $\mathrm{val}(\alpha)$. Since $\mathrm{val}(\delta_1), \ldots, \mathrm{val}(\delta_p)$ are pairwise vertex-disjoint and since $f_i(\underline{r}_i)$ refines the label partition of $\mathrm{val}(\delta_i)$ for every $1 \leq i \leq p$, $f(\underline{r}_1) \cup \cdots \cup f(\underline{r}_p) \sqsubseteq f(\underline{r})$. And since each edge of $\mathrm{val}(\alpha)$ is an edge of one of $\mathrm{val}(\delta_1), \ldots, \mathrm{val}(\delta_p)$, the Compatibility condition for (T, f) is verified straightforward. Thus, (T, f) is a partition tree for $\mathrm{val}(\alpha)$, and since α is a k-expression and the label partition of $\mathrm{val}(\alpha)$ is of size at most k, (T, f) is of width at most k.

We construct the representation graph (D, h) of (T, f) for $\mathrm{val}(\alpha)$. Let $(D_1, h_1), \ldots, (D_p, h_p)$ be the representation graphs of $(T_1, f_1), \ldots, (T_p, f_p)$, respectively. We define D:

- vertex set of D
  $V(D) = V(D_1) \cup \cdots \cup V(D_p) \cup \{\underline{r}\} \cup \{(\underline{r}, Y) : Y \in f(\underline{r})\}$
- arc set of D
  $A(D) = A(D_1) \cup \cdots \cup A(D_p) \cup$
  $\{\underline{r} \to \underline{r}_i : 1 \leq i \leq p\} \cup \{\underline{r} \to (\underline{r}, Y) : Y \in f(\underline{r})\} \cup$
  $\{(\underline{r}, Y) \to (\underline{r}_i, Z) : 1 \leq i \leq p \text{ and } Y \in f(\underline{r}) \text{ and } Z \in f(\underline{r}_i) \text{ and } Z \subseteq Y\}$.

It is straightforward to observe that D is the representation graph of (T, f) indeed. The labelling function h is defined on $\{(\underline{r}, Y) : Y \in f(\underline{r})\}$ only, according to Definition 3.5. It directly follows that (D, h) is the representation graph of (T, f) for $\mathrm{val}(\alpha)$. □

We have described an incremental algorithm to construct the representation graph of a partition tree for $\mathrm{val}(\alpha)$. It remains to determine the running time of the algorithm. The construction of the representation graph with operation 1 takes constant time per vertex. Operation 2 does not require any construction time. With operation 3, the modification of the label function to obtain h' takes time $\mathcal{O}(k)$, by examining the vertices of D of the form $(\underline{r}, Y)$. Recall here that α being a k-expression implies that $f(\underline{r})$ is of size at most k.

Finally, with operation 4, let o_i be the $\bigoplus$-operations in α, and for each operation o_i, let p_i be the arity of o_i. For each o_i, at most $k+1$ new vertices and at most $p_i + k + p_i \cdot k$ new arcs are added. Adding the type-0 and type-1 arcs is easy. For the type-2 arcs, it suffices to observe that the vertices in Y and Z have the same label. This together requires time $\mathcal{O}(p_i \cdot k)$. Summing up the running times, all such operations take time $\mathcal{O}(k \cdot |\alpha|)$, since $p_1 + p_2 + p_3 + \cdots \leq |\alpha|$.

In total, our algorithm constructs a representation graph for $\mathrm{val}(\alpha)$ in time $\mathcal{O}(k \cdot |\alpha|)$. ∎

Proposition 3.7. *There is an algorithm that transforms a graph G and the representation graph (D, h) of a width-k partition tree for G into a k-expression for G in time $\mathcal{O}(k \cdot |D| + n^2)$, where $|D|$ is the number of vertices of D and n is the number of vertices of G.*

Proof. Let (T, f) be the partition tree represented by (D, h). Observe that (T, f) is uniquely defined by (D, h). Let $\underline{r}$ be the root of T. Since (T, f) is of width at most k, we simply assume that the vertices $(\underline{r}, Y)$ of D for $Y \in f(\underline{r})$ have pairwise different labels. We will see in the course of the construction that this assumption is no restriction to the general situation, where different vertices may have the same label.

We begin our algorithm with a preparation step, that extends the labelling function h for D to further vertices of D. The root of T, $\underline{r}$, is already processed. In a top-down manner on T, starting at $\underline{r}$, we continue. Let $\underline{b}$ be a child of an already processed inner node $\underline{a}$ of T. We process $\underline{b}$ in two steps:

a) for each $Y \in f(\underline{a})$, arbitrarily choose $Z \in f(\underline{b})$ such that $Z \subseteq Y$, if it exists, and assign the label of $(\underline{a}, Y)$ to $(\underline{b}, Z)$

b) arbitrarily assign labels to the still unlabelled vertices of the form $(\underline{b}, Z)$ for $Z \in f(\underline{b})$ such that all vertices of the form $(\underline{b}, Z)$ for $Z \in f(\underline{b})$ have pairwise different labels.

Using the arcs of type 1 and 2 of D, the algorithm can assign the labels in linear time. Note here in particular that step a of the procedure can be seen as simply picking a type-2 arc, and step b is easily executable, since $f(\underline{b})$ is of size at most k, and therefore, k colours suffice to label the vertices $(\underline{b}, Z)$ with pairwise different labels.

We will use these assigned labels later.

Our transformation algorithm is a bottom-up algorithm on T, that defines a k-expression $\alpha(\underline{a})$ for each node $\underline{a}$ of T such that $\mathrm{val}^\circ(\alpha(\underline{a}))$ is equal to $G[\overline{f}(\underline{a})]$ and the labels of $\mathrm{val}(\alpha(\underline{a}))$ correspond to the labels assigned to the vertices $(\underline{a}, Y)$ of D in the preparation step. So, let $\underline{a}$ be a node of T. If $\underline{a}$ is a leaf of T then let $\alpha(\underline{a}) =_{\mathrm{def}} o(x)$ for $f(\underline{a}) = \big\{\{x\}\big\}$ and o the label of $(\underline{a}, \{x\})$. The claims about $\alpha(\underline{a})$ are clearly satisfied.

Next, assume that $\underline{a}$ is an inner node of T and its children $\underline{b}_1, \ldots, \underline{b}_p$ have already been considered and appropriate k-expressions $\alpha(\underline{b}_1), \ldots, \alpha(\underline{b}_p)$ have already been defined. We obtain $\alpha(\underline{a})$ in three steps: *relabelling*, *combining*, *adding edges*.

R) Relabelling $\alpha(\underline{b}_1), \ldots, \alpha(\underline{b}_p)$

Let $Y \in f(\underline{a})$ and $Z \in f(\underline{b}_i)$ such that $Z \subseteq Y$. Note here that $(\underline{a}, Y) \to (\underline{b}_i, Z)$ is a type-2 arc of D. Let o and s be the labels of the vertices $(\underline{a}, Y)$ and $(\underline{b}_i, Z)$ of D, respectively. Recall that the vertices in Z are the vertices of $\mathrm{val}(\alpha(\underline{b}_i))$ with label s, and due to the preparation step, some vertex of $\mathrm{val}(\alpha(\underline{b}_1)), \ldots, \mathrm{val}(\alpha(\underline{b}_p))$ must have label o; the latter is of utmost importance, since this guarantees that in the construction of $\mathrm{val}(\alpha(\underline{a}))$, described below, all vertices of Y receive label o, as required. If $o \neq s$ then we extend $\alpha(\underline{b}_i)$ through $\rho_{s\to o}$ into $\rho_{s\to o}(\alpha(\underline{b}_i))$.

We repeat this extension procedure as often as possible, and obtain a k-expression $\alpha'(\underline{b}_i)$ for $\mathrm{val}^\circ(\alpha(\underline{b}_i))$ such that the label of each vertex y of $\mathrm{val}(\alpha'(\underline{b}_i))$ is the label of the vertex $(\underline{a}, Y)$ of D where $y \in Y$.

C) Combining $\alpha'(\underline{b}_1), \ldots, \alpha'(\underline{b}_p)$

Let $\beta(\underline{a}) =_{\mathrm{def}} \bigoplus(\alpha'(\underline{b}_1), \ldots, \alpha'(\underline{b}_p))$. It suffices to observe that $\beta(\underline{a})$ is a k-expression for the disjoint union of the graphs $\mathrm{val}^\circ(\alpha(\underline{b}_1)), \ldots, \mathrm{val}^\circ(\alpha(\underline{b}_p))$, the label partition of $\mathrm{val}(\beta(\underline{a}))$

is equal to $f(\underline{a})$, and the labels of the vertices of $\mathrm{val}(\beta(\underline{a}))$ are as the corresponding labels of the vertices $(\underline{a}, Y)$ of D.

A) Adding edges to $\beta(\underline{a})$

Let $X, Y \in f(\underline{a})$ where $X \neq Y$, and let s and o be the labels of the vertices $(\underline{a}, X)$ and $(\underline{a}, Y)$ of D, respectively. Recall from the label assignment in the preparation step that $s \neq o$ is the case. Let x, y be an adjacent vertex pair of G, and let i, j be with $x \in \overline{f}(\underline{b}_i)$ and $y \in \overline{f}(\underline{b}_j)$. Recall that the definition of partition trees implies the uniqueness of i for x and j for y. If $x \in X$ and $y \in Y$ then we extend $\beta(\underline{a})$ through $\eta_{s,o}$ into $\eta_{s,o}(\beta(\underline{a}))$.

We claim that $\eta_{s,o}$ adds only edges of G. Since (T, f) satisfies the Compatibility condition and since the assumptions of the Compatibility condition are satisfied, $X \times Y \subseteq E(G)$ is the case, and so, only edges of G are indeed added.

We repeat the extension procedure as often as possible, and obtain a k-expression $\alpha(\underline{a})$. We claim that $\mathrm{val}^\circ(\alpha(\underline{a}))$ is equal to $G[\overline{f}(\underline{a})]$. Let x, y be an adjacent vertex pair of G. If $x, y \in \overline{f}(\underline{b}_i)$ for some $1 \leq i \leq k$ then xy is an edge of $\mathrm{val}(\alpha(\underline{b}_i))$, and thus of $\mathrm{val}(\alpha(\underline{a}))$. Otherwise, $x \in \overline{f}(\underline{b}_i)$ and $y \in \overline{f}(\underline{b}_j)$ for $1 \leq i, j \leq k$ and $i \neq j$. Then, there are $X' \in f(\underline{b}_i)$ and $Y' \in f(\underline{b}_j)$ such that $x \in X'$ and $y \in Y'$, and there are $X, Y \in f(\underline{a})$ such that $X' \subseteq X$ and $Y' \subseteq Y$. The Compatibility condition yields $X \neq Y$, and our extension procedure has made x and y adjacent.

By induction, we obtain $\alpha(\underline{r})$ as the desired k-expression for G.

We consider the running time of our algorithm. Recall that the preparation step takes time linear in D. We consider the transformation algorithm and determine the time taken per node of T. In overall linear time, the leaves of T can be identified, and processing a leaf of T takes constant time. Next, let $\underline{a}$ be an inner node of T. We consider the three construction steps.

R) Relabelling

It suffices to observe that the label pairs can be determined by using the type-1 and type-2 arcs of D associated with $\underline{a}$, which yields a linear running time in the number of these arcs.

C) Combining

We use a pointer as a variable and hereby avoid an explicit combination of the p expressions. This step takes time linear in p.

A) Adding edges

We need to determine the pairs X, Y from $f(\underline{a})$ that contain vertices from different $\overline{f}(\underline{b}_i)$. Observe that a pair X, Y is not of this form if and only if $X \cup Y \subseteq \overline{f}(\underline{b}_i)$ for some $1 \leq i \leq k$. This can be checked by using the type-2 arcs of D. So, if X and Y contain vertices from different $\overline{f}(\underline{b}_i)$, there are children $\underline{b}$ and $\underline{b}'$ of $\underline{a}$ where $\underline{b} \neq \underline{b}'$ and $X' \in f(\underline{b})$ and $Y' \in f(\underline{b}')$ such that $X' \subseteq X$ and $Y' \subseteq Y$, i.e., such that $(\underline{a}, X) \to (\underline{b}, X')$ and $(\underline{a}, Y) \to (\underline{b}', Y')$ are arcs of D; any such pair of arcs can serve as a witness in the following.

We need to decide whether X' and Y' contain adjacent vertices; if this is the case then $X \times Y \subseteq E(G)$ due to the Compatibility condition, and any pair of vertices from X' and

Y' would be a suitable witness. Observe here that D has no direct representation of the vertices in X' and Y'. However, in a bottom-up manner during a preprocessing step, we can associate to each vertex $(\underline{u}, Z)$ of D a vertex z of G satisfying $z \in Z$. This can be achieved in time linear in D.

We sum up the running times. At most $k \cdot |f(\underline{a})|$ pairs X, Y are considered, suitable representative vertices are determined in constant time, and the adjacency relation of G is represented by an adjacency matrix.

As the overall total running time, we obtain $\mathcal{O}(k \cdot |D| + n^2)$, where $\mathcal{O}(n^2)$ is the time taken for building the adjacency matrix of G.

To complete the construction and obtain a k-expression for G that also respects the labels in h, it suffices to recall that the label partition of $\mathrm{val}(\alpha(\underline{r}))$ refines the label partition defined by h, and we simply extend $\alpha(\underline{r})$ applying the above Relabelling step. ▪

We can state our main characterisation result for clique-width. A partition tree (T, f) is called *binary* if each inner node of T has exactly two children.

Theorem 3.8. *Let $k \geq 1$, and let G be a graph. The following statements are equivalent:*

1) $\mathrm{cwd}(G) \leq k$

2) G has a width-k partition tree

3) G has a binary width-k partition tree.

Proof. The equivalence of statements 1 and 2 is a consequence of Propositions 3.6 and 3.7. Since binary partition trees are partition trees obviously, statement 3 implies statement 2.

We show that statement 2 implies statement 3, by transforming a partition tree into a binary partition tree of the same width. Let (T, f) be a partition tree that is not binary. Let $\underline{a}$ be an inner node of T with $\underline{b}_1, \ldots, \underline{b}_p$ its children where $p \geq 3$. Let $\underline{a}'$ be a new node. We obtain (T', f') from (T, f) by adding $\underline{a}'$ as a new child of $\underline{a}$, making $\underline{b}_2, \ldots, \underline{b}_p$ children of $\underline{a}'$ instead of $\underline{a}$, and extending f into f' by letting $f'(\underline{a}') =_{\mathrm{def}} \{X \setminus \overline{f}(\underline{b}_1) : X \in f(\underline{a}) \text{ and } X \not\subseteq \overline{f}(\underline{b}_1)\}$. It is not difficult to verify $f'(\underline{b}_2) \cup \cdots \cup f'(\underline{b}_p) \sqsubseteq f'(\underline{a}')$ and $f'(\underline{a}') \cup f'(\underline{b}_1) \sqsubseteq f'(\underline{a})$. It directly follows that (T', f') is a width-k partition tree. And since the Compatibility condition is satisfied by $f(\underline{a})$ in (T, f) and since $f'(\underline{a}')$ is a restriction of $f(\underline{a})$, the Compatibility condition is satisfied by $f'(\underline{a}')$ and $f'(\underline{a})$ in (T', f').

If (T', f') is not yet a binary partition tree, we repeat the above transformation procedure as often as necessary. Eventually, a binary width-k partition tree for G is obtained. ▪

A computational problem that is related to expressions, and thus to partition trees, is the relative clique-width problem [17]. Using our results about the equivalence of expressions and partition trees and the properties of the two transformation algorithms of Propositions 3.6 and 3.7, the relative clique-width problem can be equivalently stated as follows: given an integer k, a graph G on n vertices, a rooted binary tree T with n leaves, and a bijection λ between the leaves of T and the vertices of G, it is to decide whether an assignment function f exists such that (T, f) is a binary width-k partition tree for G and $f(\underline{a}) = \big\{\{\lambda(\underline{a})\}\big\}$ for every leaf $\underline{a}$ of T. Müller and Urner showed that the relative clique-width problem is NP-hard [18].

At the beginning of this section, we were interested in a succinct representation of partition trees, and we used the representation graphs mainly to succinctly represent the partial partitions assigned to the nodes of the tree. The hardness, or intractability, result for the relative clique-width problem explains that an algorithmically useful representation of partition trees, from which the partial partitions are extractable efficiently, must represent the partitions necessarily.

4 Partition trees and linear clique-width

A variant of clique-width is linear clique-width, that is considered the clique-width analogue of pathwidth in relation to treewidth. Characterisations of linear clique-width and closure properties are known [1, 11, 13, 14, 17]. In this section, we discuss some consequences of our clique-width characterisation, Theorem 3.8, when applied to linear clique-width.

We use a special class of rooted trees. In our applications of this section, a *caterpillar* is a rooted tree whose inner nodes have at most one child that is not a leaf. Partition trees that are caterpillars are called *partition caterpillar trees*.

Proposition 4.1. *Let $k \geq 1$, and let G be a graph. The linear clique-width of G is at most k if and only if G has a width-k partition caterpillar tree.*

Proof. If $\mathrm{lcwd}(G) \leq k$ then G has a linear k-expression, and the algorithm of Proposition 3.6 generates a width-k partition tree for G that is a caterpillar, in particular since $\delta_2 = o_2(u_2), \ldots, \delta_p = o_p(u_p)$ for operation type 4.

Conversely, applying the transformation algorithm of Proposition 3.7 to the representation graph of a width-k partition caterpillar tree for G yields a linear k-expression for G, in particular since $\underline{b}_2, \ldots, \underline{b}_p$ in the second step *combining* are leaves and $\mathrm{val}^\circ(\alpha(\underline{b}_2)), \ldots, \mathrm{val}^\circ(\alpha(\underline{b}_p))$ are single-vertex graphs, thus proving $\mathrm{lcwd}(G) \leq k$. ∎

Proposition 4.1 characterises linear clique-width in analogy to the characterisation of clique-width in Theorem 3.8. Further characterisations of linear clique-width are known [11, 13, 17], that are mainly based on vertex layouts. Common to the vertex layout characterisations of linear clique-width is the classification of already processed vertices with respect to their remaining unprocessed neighbours. We discuss here how the partition caterpillar trees relate to the vertex layout characterisations. A central tool is a combinatorial description of the partition classes in the assigned partition labels of the partition trees.

Let G be a graph, and let $R \subseteq E(G)$. Let $H =_{\mathrm{def}} G \setminus R$. Let u, v be a vertex pair of G. We say that u, v satisfies the *group condition* in H if $N_H(u) = N_H(v)$, i.e., if u and v have the same neighbours in H. For $X \subseteq V(G)$, we call X a *group* of H if each vertex pair from X satisfies the group condition in H. Groups are sets of vertices of G, and H, with the same neighbourhood in H. As an example, reconsider graph G of Figure 1. The groups of $G \setminus \{vw, yz\}$ with at least two vertices are $\{w, y\}, \{w, z\}, \{y, z\}, \{w, y, z\}$. Observe that $\{u, v\}$ is not a group of $G \setminus \{vw, yz\}$, particularly since u and v are adjacent in $G \setminus \{vw, yz\}$.

The notion of a group generalises that of a module in modular decomposition. For $B \subseteq V(G)$, denote by $E[B]$ the set $\{uv \in E(G) : u, v \in B\}$, that is the edge set of $G[B]$. A module of G is a set M of vertices of G such that M is a group of $G \setminus E[M]$. The relationship between clique-width and modular decomposition is fully known [7].

As a first result, we consider the partition labels of partition trees. Let G be a graph, let (T, f) be a partition tree for G, and let $\underline{a}$ be a node of T. By G_a^-, we denote the graph $G \setminus E[\overline{f}(\underline{a})]$, that is the residue graph of G when deleting the edges between the vertices in $\overline{f}(\underline{a})$.

Lemma 4.2. *Let G be a graph, and let (T, f) be a partition tree for G. Let $\underline{a}$ be an inner node of T with child $\underline{b}$, and let $X \in f(\underline{a})$. The following hold true:*

1) X is a group of G_a^-, and $X \cap \overline{f}(\underline{b})$ is a group of G_b^-

2) no vertex from $X \cap \overline{f}(\underline{b})$ is adjacent to a vertex from $X \setminus \overline{f}(\underline{b})$ in G.

Proof. Let $B =_{\text{def}} \overline{f}(\underline{b})$. Observe that no vertex from $X \cap B$ is adjacent to a vertex from $X \setminus B$ due to the Compatibility condition of Definition 3.2, which proves the second claim.

We prove the second part of the first claim, by proving its contraposition. Recall $G_b^- = G \setminus E[B]$. Let $Y =_{\text{def}} X \cap B$, and assume that Y is not a group of G_b^-. Then, there are $u, v \in Y$ and $z \in V(G)$ such that $uz \in E(G_b^-)$ and $vz \notin E(G_b^-)$. Recall that $z \notin B$ must hold, since the vertices in B are pairwise non-adjacent in G_b^-.

Let $\underline{a}'$ be the inner node of T that has children $\underline{b}'$ and $\underline{b}''$ satisfying $u, v \in \overline{f}(\underline{b}')$ and $z \in \overline{f}(\underline{b}'')$; note that $\underline{a}'$, $\underline{b}'$ and $\underline{b}''$ do exist. Choose $X' \in f(\underline{a}')$ such that $u \in X'$. Due to the Compatibility condition for $\underline{a}'$, $X' \subseteq N_G(z)$ is the case, which implies $v \notin X'$. Descending from $\underline{a}'$ to $\underline{a}$ in T and iteratively applying the refinement property of the second condition of Definition 3.1, we conclude that u and v do not appear in the same partition class of $f(\underline{a})$.

The first part of the first claim follows as an application of the second part. If $\underline{a}$ is the root of T, $V(G)$ is a group of G_a^-, and X in particular. Assume that $\underline{a}$ is not the root of T; let $\underline{a}'$ be the parent of $\underline{a}$. According to the second condition of Definition 3.1, there exists $X' \in f(\underline{a}')$ such that $X \subseteq X'$, and $X' \cap \overline{f}(\underline{a})$ is a group of G_a^- according to the second part of the first claim, and X is a group of G_a^- because of $X \subseteq X' \cap \overline{f}(\underline{a})$. ∎

Let G be a graph, let $A \subseteq V(G)$, and let $R \subseteq E(G)$. A *maximal group* of $G \setminus R$ with respect to A is a group X of $G \setminus R$ with $X \subseteq A$ that is not properly contained in any group X' of $G \setminus R$ with $X' \subseteq A$. Since the group condition for $G \setminus R$ defines an equivalence relation on $V(G)$, A is partitioned uniquely into maximal classes, that are the maximal groups. With the example of Figure 1, the maximal groups of $G \setminus \{vw, yz\}$ with respect to $\{u, v, w, x, y\}$ are $\{u\}, \{v\}, \{x\}, \{w, y\}$.

Let (T, f) be a partition tree for G, and let $\underline{a}$ be a node of T. By $M(\underline{a})$, we denote the set of the maximal groups of G_a^- with respect to $\overline{f}(\underline{a})$.

Corollary 4.3. *Let $k \geq 1$, let G be a graph, and let (T, f) be a binary width-k partition tree for G. Let*

$$\vartheta \ =_{\text{def}} \ \max\left\{|M(\underline{a})| : \underline{a} \text{ is a node of } T\right\}.$$

Then, $\vartheta \leq k$ and $\operatorname{cwd}(G) \leq 2\vartheta$.

Proof. For the first inequality, observe for each node $\underline{a}$ of T: every maximal group of G_a^- with respect to $\overline{f}(\underline{a})$ consists of the union of one or more members of $f(\underline{a})$ by the first claim of Lemma 4.2. Therefore, the number of maximal groups of G_a^- with respect to $\overline{f}(\underline{a})$ is at most $|f(\underline{a})|$. This means $|M(\underline{b})| \leq |f(\underline{a})|$. Since $|f(\underline{a})| \leq k$ clearly, $\vartheta \leq k$ follows.

For the second inequality, we define a new partition assignment function g for T. For $\underline{a}$ a node of T:

$$g(\underline{a}) \;=_{\text{def}}\; \begin{cases} f(\underline{a}) & \text{, if } \underline{a} \text{ is a leaf} \\ M(\underline{b}_1) \cup M(\underline{b}_2) & \text{, if } \underline{a} \text{ is an inner node with } \underline{b}_1 \text{ and } \underline{b}_2 \text{ its children}\,. \end{cases}$$

If (T, g) is a partition tree for G, the result of the first paragraph directly show that (T, g) is of size at most 2ϑ. We show that (T, g) is a partition tree for G indeed. We verify the conditions of Definitions 3.1 and 3.2 for the inner nodes of T. Let $\underline{a}$ be an inner node of T with $\underline{b}_1$ and $\underline{b}_2$ its children.

We verify $g(\underline{b}_1) \cup g(\underline{b}_2) \sqsubseteq g(\underline{a})$. Observe that G^-_a is a subgraph of $G^-_{b_1}$ and of $G^-_{b_2}$. Thus, each group of $G^-_{b_1}$ and of $G^-_{b_2}$ is a group also of G^-_a. So, $g(\underline{a})$ is a set of groups of G^-_a, and since $\overline{g}(\underline{a}) = \overline{g}(\underline{b}_1) \cup \overline{g}(\underline{b}_2)$, we conclude $g(\underline{b}_1) \cup g(\underline{b}_2) \sqsubseteq M(\underline{b}_1) \cup M(\underline{b}_2) \sqsubseteq g(\underline{a}) \sqsubseteq M(\underline{a})$.

We verify the Compatibility condition. Let x, y be an adjacent vertex pair of G and let $X, Y \in g(\underline{a})$, and assume $x \in X$ and $y \in Y$ and $x \in \overline{g}(\underline{b}_1)$ and $y \in \overline{g}(\underline{b}_2)$. Then, $X \in M(\underline{b}_1)$ and $Y \in M(\underline{b}_2)$, and $X \neq Y$ in particular, and since xy is an edge of $G^-_{b_1}$, and of $G^-_{b_2}$, each vertex in X is adjacent to y in G, so that $X \subseteq N_G(y)$, and thus, $X \subseteq N_G(y')$ for every vertex $y' \in Y$, and thus, $X \times Y \subseteq E(G)$.

We conclude that (T, g) is a partition tree for G indeed, and the size of (T, g) is at least $\mathrm{cwd}(G)$ due to Theorem 3.8. ∎

We come back to partition caterpillar trees and linear clique-width: we apply Corollary 4.3 and the construction in the proof to a graph G and a partition caterpillar tree for G of width $\mathrm{lcwd}(G)$, that exists due to Proposition 4.1, and we conclude $\vartheta \leq \mathrm{lcwd}(G) \leq \vartheta + 1$. We show that the ideas of Lemma 4.2 extend into a characterisation of linear clique-width. For convenience, we restrict to binary partition caterpillar trees.

Theorem 4.4. *Let $k \geq 1$, and let G be a graph. The linear clique-width of G is at most k if and only if there is a binary partition caterpillar tree (T, f) for $V(G)$ of width at most k that satisfies the following conditions for every inner node $\underline{a}$ of T with $\underline{b}_1$ and $\underline{b}_2$ its children, where $\underline{b}_2$ is a leaf and $\overline{f}(\underline{b}_2) = \{w\}$:*

- *for $W \in M(\underline{a})$ with $w \in W$, if there is $W' \in M(\underline{b}_1)$ such that $W' \subseteq W$ and $N_G(w) \subseteq N_G(w')$ for each $w' \in W'$ then $f(\underline{a}) = \big(M(\underline{b}_1) \setminus \{W'\}\big) \cup \big\{W' \cup \{w\}\big\}$*
- *otherwise, $f(\underline{a}) = M(\underline{b}_1) \cup \big\{\{w\}\big\}$.*

Proof. Assume that a binary partition caterpillar tree (T, f) for $V(G)$ of the given properties exists. We show that (T, f) is a partition tree for G. Let $\underline{a}$ be an inner node of T with $\underline{b}_1$ and $\underline{b}_2$ its children, where $\underline{b}_2$ is a leaf and $\overline{f}(\underline{b}_2) = \{w\}$.

Note that (T, f) is indeed a partition tree: $f(\underline{b}_1) \cup f(\underline{b}_2) \sqsubseteq M(\underline{b}_1) \cup \big\{\{w\}\big\} \sqsubseteq f(\underline{a}) \sqsubseteq M(\underline{a})$, analogous to the proof of Corollary 4.3.

Next, for verifying the Compatibility condition, let $x \in \overline{f}(\underline{b}_1)$ with $xw \in E(G)$ and let $X, Y \in f(\underline{a})$ with $x \in X$ and $w \in Y$. If $\{w\} \in f(\underline{a})$ then $Y = \{w\}$, and $X \neq Y$, and since xw is an edge of $G^-_{b_1}$ and X is a group of $G^-_{b_1}$, $X \subseteq N_G(w)$ follows. If $\{w\} \notin f(\underline{a}\}$ then $Y = W' \cup \{w\}$, and since $ww' \notin E(G)$ for each $w' \in W'$, $X \neq Y$ follows, and since X is a group of $G^-_{b_1}$, $X \subseteq N_G(w)$ follows, and thus, $X \subseteq N_G(y)$ for each $y \in Y$ according to the choice of W'.

Thus, (T, f) is a partition tree for G, and $\mathrm{lcwd}(G)$ is at most the width of (T, f) due to Proposition 4.1.

For the converse, let (T, g) be a binary partition caterpillar tree for G. Obtain (T, f) from (T, g) by defining f according to the theorem, based on (T, g). We compare the widths of (T, f) and (T, g).

Let $\underline{a}$ be an inner node of T with $\underline{b}_1$ and $\underline{b}_2$ its children, where $\underline{b}_2$ is a leaf and $\overline{f}(\underline{b}_2) = \overline{g}(\underline{b}_2) = \{w\}$. Observe $|f(\underline{a})| \leq |M(\underline{b}_1)| + 1$. Assume $|g(\underline{a})| \leq |M(\underline{b}_1)|$. Every maximal group of G_a^- with respect to $\overline{g}(\underline{a})$ consists of the union of one or more members of $g(\underline{a})$ by the first claim of Lemma 4.2. Thus, the number of maximal groups of G_a^- with respect to $\overline{g}(\underline{a})$ is at most $|g(\underline{a})|$. This means $|M(\underline{b}_1)| \leq |g(\underline{a})|$, and therefore $M(\underline{b}_1) = \big\{X \cap \overline{g}(\underline{b}_1) : X \in g(\underline{a})\big\}$.

Let $Y \in g(\underline{a})$ with $w \in Y$. So, $Y \setminus \{w\} \in M(\underline{b}_1)$. Observe that Y is a group of G_a^- by the first claim of Lemma 4.2. The group condition and the Compatibility condition together show $N_G(w) \subseteq N_G(y)$ for each $y \in Y \setminus \{w\}$, and $f(\underline{a})$ is defined according to the first case. This means $|f(\underline{a})| = |M(\underline{b}_1)| = |g(\underline{a})|$ in particular.

We conclude: the width of (T, f) is at most the width of (T, g). Choosing (T, g) of width at most $\mathrm{lcwd}(G)$, which is possible according to Proposition 4.1, proves the theorem. ∎

Theorem 4.4 shows that a partition assignment function of smallest width for a binary caterpillar tree can be obtained simply by an extension from the labels of the leaves. This partition label assignment is at the heart of the vertex layout characterisations of linear clique-width in [11, 13, 17]. The characterisation in [13] employs an indicator function that determines whether the first or the second case about the partition label in Theorem 4.4 must be chosen.

5 On a characterisation of clique-width by Heule and Szeider

A novel algorithmic approach to clique-width was presented by Heule and Szeider [16]. They use SAT-solvers to solve the clique-width decision problem. As a central tool, they gave a characterisation of clique-width by means of nested partitions. In this section, we review and compare their characterisation and relate it to that of Theorem 3.8.

We will also use the notion of *strong refinement* defined at the end of Section 2.

Definition 5.1 (Derivation over sets)**.** *Let S be a set. A* derivation *over S is a sequence $(C_i, D_i)_{0 \leq i \leq q}$ of partition pairs of S satisfying the following conditions:*

1) *$C_0 = D_0 = \big\{\{x\} : x \in S\big\}$, and $C_q = \big\{S\big\}$*

2) *$C_i \sqsubseteq C_{i+1}$ and $D_i \sqsubseteq D_{i+1}$ for every $0 \leq i < q$, and $D_i \sqsubseteq C_i$ for every $0 \leq i \leq q$.*

The derivation $(C_i, D_i)_{0 \leq i \leq q}$ is called strict *if $|C_i| > |C_{i+1}|$ for every $0 \leq i < q$, and it is called* strong *if $C_i \sqsubseteq_{\mathrm{str}} C_{i+1}$ for every $0 \leq i < q$.*

Observe that strong derivations are strict, unless there is $0 \leq i < q$ such that $C_i = C_{i+1} = C_0$.

Definition 5.2 (Derivations of graphs)**.** *Let G be a graph. A* derivation *of G is a derivation $(C_i, D_i)_{0 \leq i \leq q}$ over $V(G)$ satisfying for every $0 \leq i < q$ and every vertex pair u, v of G where $uv \in E(G)$ and u and v appear in different partition classes of C_i:*

1) u and v appear in different partition classes of D_{i+1}

2) for every vertex pair w, x of G, if x and u are in the same partition class of D_{i+1} and w and v are in the same partition class of D_{i+1} then $wx \in E(G)$.

The width *of $(C_i, D_i)_{0 \leq i \leq q}$ is the maximum number of partition classes of D_i that are contained in the same partition class of C_i for $0 \leq i \leq q$, and q is the* length *of $(C_i, D_i)_{0 \leq i \leq q}$.*

Clique-width and derivations of graphs are equivalent. The following result is proved as Proposition 1 and Lemma 1 in [16].

Theorem 5.3 ([16])**.** *Let $k \geq 1$, and let G be a graph on n vertices. The clique-width of G is at most k if and only if G has a derivation of width at most k, if and only if G has a strict derivation of width at most k and of length at most $n - k + 1$.*

Theorem 5.3 can be strengthened by asking for strong derivations in place of strict derivations, as we will show in Proposition 5.4 below. As a consequence of Theorem 5.3, Theorem 3.8 can be extended by adding derivations of graphs of bounded width as a fourth statement.

Before we prove the equivalence of partition trees and derivations, we want to discuss the application of derivations to clique-width computation as applied by Heule and Szeider [16]. First, observe the difference between our Definition 5.2 and the original definition in [16]: our condition 2 is originally expressed in two separate conditions:

2.1) for w a vertex different from u and v that is in the same partition class of D_{i+1} as v, $uw \in E(G)$

2.2) for w, x a vertex pair such that u, v, w, x are pairwise different, if x and u are in a same partition class of D_{i+1} and w and v are in the same partition class of D_{i+1} and if $uw, vx \in E(G)$ then $wx \in E(G)$.

Observe that conditions 2.1 and 2.2 are consequences of condition 2. For the converse, condition 2 for the particular case of $x = u$ or $v = w$ is concluded from condition 2.1, and the case of $x \neq u$ and $v \neq w$ is concluded from condition 2.2.

The alternative definition of derivations in Definition 5.2 is interesting for the SAT-solver application of [16], as we briefly explain. Let G be a graph on n vertices, and let $k \geq 1$. We create Boolean variables and clauses to express the existence of a strict k-derivation for G of width at most k and of length $n - k + 1$. A typical variable is $c_{u,v,i}$, whose value `true` has the meaning that the vertices u and v are in a same partition class of C_i. This way, we create $n^2(n - k + 2)$ variables. We also create variables $d_{u,v,i}$ to express that u and v are in the same partition class of D_i. Furthermore, we assume a labelling of the partition classes of D_i with labels from $\{1, \ldots, k\}$ such that two partition classes of D_i that are contained in the same partition class of C_i have different labels. We create variables $l_{v,a,i}$ to express that the partition class of D_i containing v has label a. In total, $(2n^2 + nk)(n - k + 2)$ variables are created.

Clauses over the defined variables depend on the edges of G. Clauses express that an truth-value assignment encodes a strict derivation for G of width at most k and of length $n - k + 1$. The number of clauses is $O(n^4 \cdot (n - k))$.

The different formulations of derivations have direct efficiency consequences. The translation of conditions 1 and 2 of Definition 5.2 needs more clauses than the translation of conditions 1 and 2.1 and 2.2, especially since the variables w and x have a smaller range in condition 2.2. Using strong derivations instead of strict derivations adds more constraints and results in quicker negative answers, which means a quicker response in case of $\mathrm{cwd}(G) > k$.

We want to compare some computational aspects of the construction of derivations from partition trees, and vice versa.

Let G be a graph, and let (T, f) be a partition tree for G. Recall the definition of *depth* in rooted trees: the depth of a node is its distance to the root. The *height* of T is the largest depth among the leaves of T (the right-side partition tree of Figure 1 has height 3). Let q be the height of T. For every $0 \leq i \leq q$, we define the partition pair (C_i, D_i). Let L_i be the set of the nodes that are of depth $q - i$ in T or that are leaves of T of depth less than $q - i$. Note that L_0 is the set of the leaves of T exactly, and L_q contains the root of T as the only node. We define:

$$C_i \ =_{\mathrm{def}} \ \big\{\overline{f}(\underline{a}) : \underline{a} \in L_i\big\} \quad \text{and} \quad D_i \ =_{\mathrm{def}} \ \bigcup_{\underline{a} \in L_i} f(\underline{a})\,.$$

Let $\mathrm{Deriv}(T, f) =_{\mathrm{def}} (C_i, D_i)_{0 \leq i \leq q}$ be the constructed sequence of partition pairs. We will show that $\mathrm{Deriv}(T, f)$ is a strong derivation of G.

As an example, consider the right-side partition tree of Figure 1: the height is 3, the sets of nodes are $L_0 = \{\underline{a}_u, \underline{a}_v, \underline{a}_w, \underline{a}_x, \underline{a}_y, \underline{a}_z\}, L_1 = \{\underline{a}_4, \underline{a}_x, \underline{a}_y, \underline{a}_z\}, L_2 = \{\underline{a}_2, \underline{a}_3\}, L_3 = \{\underline{a}_1\}$, and the constructed derivation $(C_i, D_i)_{0 \leq i \leq 3}$ is as follows:

$$\begin{aligned}
C_0 &= \big\{\{u\}, \{v\}, \{w\}, \{x\}, \{y\}, \{z\}\big\} & D_0 &= \big\{\{u\}, \{v\}, \{w\}, \{x\}, \{y\}, \{z\}\big\} \\
C_1 &= \big\{\{u, v, w\}, \{x\}, \{y\}, \{z\}\big\} & D_1 &= \big\{\{u\}, \{v\}, \{w\}, \{x\}, \{y\}, \{z\}\big\} \\
C_2 &= \big\{\{u, v, w, x\}, \{y, z\}\big\} & D_2 &= \big\{\{u, v\}, \{w\}, \{x\}, \{y\}, \{z\}\big\} \\
C_3 &= \big\{\{u, v, w, x, y, z\}\big\} & D_3 &= \big\{\{u, v\}, \{w, y, z\}, \{x\}\big\}\,.
\end{aligned}$$

For the converse, let G be a graph, and let $\mathcal{D} = (C_i, D_i)_{0 \leq i \leq q}$ be a derivation of G. We define a rooted tree T as follows:

- the nodes of T are in 1-to-1 correspondence with the members of $\bigcup_{i=0}^{q} C_i$, where each node is labelled with its corresponding member
- two nodes $\underline{a}$ and $\underline{b}$ of T are adjacent if and only if, for A and B the member labels of respectively $\underline{a}$ and $\underline{b}$, $B \subset A$ or $A \subset B$ and there is no member X in $\bigcup_{i=0}^{q} C_i$ such that $B \subset X \subset A$ or $A \subset X \subset B$
- the root of T is the node with label $V(G)$, that is the unique member of C_q.

It is clear that $C_0 \sqsubseteq \cdots \sqsubseteq C_q$ and the node labels from $\bigcup_{i=0}^{q} C_i$ together implies that T is a rooted tree whose inner nodes have at least two children indeed. We define a partition assignment function f for T. Let $\underline{a}$ be a node of T, and let A be the label of $\underline{a}$. Let $0 \leq t \leq q$ be smallest such that $A \in C_t$. Then, let

$$f(\underline{a}) \ =_{\mathrm{def}} \ \big\{X \in D_t : X \subseteq A\big\}\,.$$

Let $\mathrm{Parti}(\mathcal{D}) =_{\mathrm{def}} (T, f)$ be the constructed pair. We show that $\mathrm{Parti}(\mathcal{D})$ is indeed a partition tree for G.

Proposition 5.4. *Let $k \geq 1$, and let G be a graph.*

1) If (T, f) is a width-k partition tree for G then $\mathrm{Deriv}(T, f)$ is a strong derivation of G of width at most k.

2) If $\mathcal{D}$ is a derivation of G of width at most k then $\mathrm{Parti}(\mathcal{D})$ is a width-k partition tree for G.

Proof. We prove the first claim. Let (T, f) be a width-k partition tree for G. Let $\mathrm{Deriv}(T, f) = (C_i, D_i)_{0 \leq i \leq q}$, and let $L_0, \ldots, L_q$ be the corresponding sets of nodes of T. We show that $\mathrm{Deriv}(T, f)$ is a strong derivation over $V(G)$, and verify the conditions of Definition 5.1:

1) since $V(G) = \bigcup_{\underline{a} \in L_0} \overline{f}(\underline{a})$ and $L_q = \{\underline{r}\}$ for $\underline{r}$ the root of T and $\overline{f}(\underline{r}) = V(G)$, the first condition of Definition 5.1 is satisfied

2)
- for $0 \leq i \leq q$: $D_i \sqsubseteq C_i$ follows from $f(\underline{a}) \sqsubseteq \{\overline{f}(\underline{a})\}$
- for $0 \leq i < q$: $D_i \sqsubseteq D_{i+1}$ follows from the second condition of Definition 3.1
- for $0 \leq i < q$: $C_i \sqsubseteq_{\mathrm{str}} C_{i+1}$ follows from the assumption that each inner node of T have at least two children.

Next, we verify the two conditions of Definition 5.2. Let $0 \leq i < q$, let u, v, w, x be a vertex quadruple of G, and assume $uv \in E(G)$ and that u and v appear in different partition classes of C_i. Let j be smallest with $0 \leq j \leq q$ such that $u, v \in \overline{f}(\underline{a})$ for some node $\underline{a}$ in L_j. The definition of $C_0, \ldots, C_q$ and the above shown refinement structure implies $j > i$, and $\underline{a}$ is an inner node of T. Let $\underline{b}'$ and $\underline{b}''$ be two children of $\underline{a}$ such that $\underline{b}' \neq \underline{b}''$ and $u \in \overline{f}(\underline{b}')$ and $v \in \overline{f}(\underline{b}'')$.

1) Let $\underline{c} \in L_i$ such that $u \in \overline{f}(\underline{c})$. If $\underline{c} \in L_{i+1}$ then $\underline{c}$ is a leaf of T and $\overline{f}(\underline{c}) = \{u\}$, and u and v appear in different partition classes of D_{i+1}. Otherwise, for $\underline{d}$ the parent of $\underline{c}$, $\underline{d} \in L_{i+1}$, and if $v \notin \overline{f}(\underline{d})$ then u and v appear in different partition classes because of $D_{i+1} \sqsubseteq C_{i+1}$, and if $v \in \overline{f}(\underline{d})$ then $\underline{d} = \underline{a}$, and the Compatibility condition for $\underline{a}$, $\underline{b}'$ and $\underline{b}''$ implies that u and v appear in different partition classes of D_{i+1}.

2) Assume that u and x appear in the same partition class of D_{i+1} and that v and w appear in the same partition class of D_{i+1}. Then, u and x appear in the same member of $f(\underline{a})$ and v and w appear in the same member of $f(\underline{a})$. The Compatibility condition for $\underline{a}$, $\underline{b}'$ and $\underline{b}''$ together with $uv \in E(G)$ implies $\{u, x\} \times \{v, w\} \subseteq E(G)$, and $wx \in E(G)$ in particular.

It remains to observe that if (T, f) has width at most k then $\mathrm{Deriv}(T, f)$ also has width at most k.

We prove the second claim. Let $\mathcal{D} = (C_i, D_i)_{0 \leq i \leq q}$ be a derivation of G of width at most k. Recall from Definition 5.1 that $C_0 = \{\{x\} : x \in V(G)\}$ and $C_q = \{V(G)\}$. This particularly means that there is a 1-to-1 correspondence between the leaves of T and the members of C_0, thereby certifying the first condition of Definition 3.1 and proving $|\overline{f}(\underline{a})| \geq 2$ for every inner node $\underline{a}$ of T.

Let $\underline{a}$ be an inner node of T with $\underline{b}_1, \ldots, \underline{b}_p$ its children, where $p \geq 2$, as we already verified at the definition of Parti($\mathcal{D}$). Let A and $B_1, \ldots, B_p$ be the members of $\bigcup_{i=0}^{q} C_i$ that respectively $\underline{a}$ and $\underline{b}_1, \ldots, \underline{b}_p$ are labelled with. Let t be smallest with $0 \leq t \leq q$ such that $A \in C_t$. Note that $t \geq 1$, that $B_1, \ldots, B_p \in \bigcup_{i=0}^{t-1} C_i$, and that $B_1, \ldots, B_p$ are pairwise disjoint. Also note $B_1 \cup \cdots \cup B_p = A = \overline{f}(\underline{a})$.

We show $f(\underline{b}_1) \cup \cdots \cup f(\underline{b}_p) \sqsubseteq f(\underline{a})$: $f(\underline{a}) \subseteq D_t$, and $D_0 \sqsubseteq \cdots \sqsubseteq D_{t-1} \sqsubseteq D_t$, and $f(\underline{b}_1), \ldots, f(\underline{b}_p) \subseteq \bigcup_{i=0}^{t-1} D_i$. This verifies the second condition of Definition 3.1.

We verify the Compatibility condition. Let x, y be an adjacent vertex pair from $\overline{f}(\underline{a})$, let $x \in \overline{f}(\underline{b}_i)$ and $y \in \overline{f}(\underline{b}_j)$, and assume $i \neq j$. Let $X, Y \in f(\underline{a})$ be such that $x \in X$ and $y \in Y$. Recall $X, Y \in D_t$ according to the definition of $f(\underline{a})$ for Parti($\mathcal{D}$). Since $\overline{f}(\underline{a}) \notin C_{t-1}$ and thus $\overline{f}(\underline{b}_i), \overline{f}(\underline{b}_j) \in C_{t-1}$ and $\overline{f}(\underline{b}_i) \neq \overline{f}(\underline{b}_j)$, which means that x and y appear in different partition classes of C_{t-1}, we can apply the two conditions of Definition 5.2 and conclude about X and Y in D_t that $X \neq Y$ by the first condition and $X \times Y \subseteq E(G)$ by the second condition.

It remains to observe that the width of Parti($\mathcal{D}$) is equal to the width of $\mathcal{D}$. ∎

6 Conclusions

The main results of the paper are characterisations of clique-width and linear clique-width, that we obtained in Theorem 3.8, in Proposition 4.1, and in Theorem 4.4. We applied these results to obtain results that are equivalent to results from the literature. The first statement of Lemma 4.2 is related to Lemma 2 of [17], the two inequalities of Corollary 4.3 relate to Lemma 1 and Theorem 1 of [17]. The linear clique-width characterisation of Theorem 4.4 relates to results of [11, 13, 17].

We considered the related characterisation of clique-width by Heule and Szeider in [16]. We related our clique-width characterisation by partition trees to that by derivations, and we proved the equivalence of the two characterisations.

We defined a space-efficient representation of partition trees and gave efficient algorithms for converting expressions, partition trees, and derivations into one another.

As a result, clique-width admits three characterisations of different flavours: by algebraic terms (expressions), by partition trees, and, very similarly, by derivations. Each of these characterisations provides a different view on the clique-width notion, has particular advantages for algorithmic applications, and enables to use results from different areas, because they can be efficiently transferred by efficient algorithms.

References

[1] B. Courcelle. Clique-width and edge contraction. *Information Processing Letters*, 114:42–44, 2014.

[2] B. Courcelle and I. Durand. Automata for the verification of monadic second-order graph properties. *Journal of Applied Logic*, 10:368–409, 2012.

[3] B. Courcelle and J. Engelfriet. *Graph structure and monadic second-order logic, a language theoretic approach*. Encyclopedia of Mathematics and its Application 138, Cambridge University Press, 2012.

[4] B. Courcelle and M. M. Kanté. Graph operations characterizing rank-width. *Discrete Applied Mathematics*, 157:627–640, 2009.

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

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

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

[8] R. Downey and M. Fellows. *Parameterized complexity*. Springer, 1999.

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

[10] J. Flum and M. Grohe. *Parametrized complexity theory*. Springer, 2006.

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

[12] M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4:41–59, 2010.

[13] P. Heggernes, D. Meister, C. Papadopoulos. Graphs of linear clique-width at most 3. *Theoretical Computer Science*, 412:5466–5486, 2011.

[14] P. Heggernes, D. Meister, C. Papadopoulos. Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. *Discrete Applied Mathematics*, 160:888–901, 2012.

[15] P. Heggernes, D. Meister, U. Rotics. Computing the clique-width of large path powers in linear time via a new characterisation of clique-width. *Proceedings of CSR 2011*, Springer LNCS, 6651:233–246, 2011.

[16] M. J. H. Heule and S. Szeider. A SAT Approach to Clique-Width. *Proceedings of SAT 2013*, Springer LNCS, 7962:318–334, 2013.

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

[18] H. Müller and R. Urner. On a disparity between relative cliquewidth and relative NLC-width. *Discrete Applied Mathematics*, 158:828–840, 2010.

[19] S. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96:514–528, 2006.