A SAT Approach to Clique-Width

MARIJN J. H. HEULE, Department of Computer Sciences, The University of Texas at Austin
STEFAN SZEIDER, Institute of Computer Graphics and Algorithms, Vienna University of Technology

Clique-width is a graph invariant that has been widely studied in combinatorics and computational logic. Computing the clique-width of a graph is an intricate problem, the exact clique-width is not known even for very small graphs. We present a new method for computing clique-width via an encoding to propositional satisfiability (SAT) which is then evaluated by a SAT solver. Our encoding is based on a reformulation of clique-width in terms of partitions that utilizes an efficient encoding of cardinality constraints. Our SAT-based method is the first to discover the exact clique-width of various small graphs, including famous named graphs from the literature as well as random graphs of various density. With our method we determined the smallest graphs that require a small pre-described clique-width. We further show how our method can be modified to compute the linear clique-width of graphs, a variant of clique-width that has recently received considerable attention. In an appendix we provide certificates for tight upper bounds for the clique-width and linear clique-width of famous named graphs.

Categories and Subject Descriptors: Theory of computation [**Logic**]: Constraint and logic programming; Mathematics of computing [**Discrete mathematics**]: Graph theory

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: clique-width, linear clique-width, satisfiability, k-expression, cardinality constraint, SAT solver, SAT encoding

ACM Reference Format:
Marijn J. H. Heule and Stefan Szeider, 2015. A SAT approach to clique-width. *ACM Trans. Comput. Logic* 0, 0, Article 00 (January 2015), 29 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Clique-width is a fundamental graph invariant that has been widely studied in combinatorics and computational logic. Clique-width measures in a certain sense the complexity of a graph. It is defined via a graph construction process which starts from single-vertex graphs as axioms and builds from this larger graphs by means of three operations (disjoint union, vertex relabeling, and edge insertion; see Section 2.2 for details). Vertices that share the same label at a certain point of the construction process must be treated uniformly in subsequent steps. The clique-width of a graph is the smallest number of labels that suffices to construct the graph. This graph composition mechanism was first considered by Courcelle, Engelfriet, and Rozenberg [1991; 1993] and has since then been an important topic in combinatorics and computer science.

A preliminary and shortened version appeared in the proceedings of SAT'13, The 16th International Conference on Theory and Applications of Satisfiability Testing
Heule's research was supported in part by the National Science Foundation under grant CNS-0910913 and DARPA contract number N66001-10-2-4087.
Szeider's research was supported by the ERC, grant reference 239962 (COMPLEX REASON).

Graphs of small clique-width have advantageous algorithmic properties. Algorithmic meta-theorems show that large classes of NP-hard optimization problems and #P-hard counting problems can be solved in *linear time* on classes of graphs of bounded clique-width [Courcelle et al. 2000; 2001]. Similar results hold for the graph invariant *treewidth*, however, clique-width is more general in the sense that graphs of small treewidth also have small clique-width, but there are graphs of small clique-width but arbitrarily large treewidth [Courcelle and Olariu 2000; Corneil and Rotics 2005]. Unlike treewidth, dense graphs (e.g., cliques) can also have small clique-width.

All these algorithms for graphs of small clique-width require that a certificate for the graph having small clique-width is provided. However, it seems that computing the certificate, or just deciding whether the clique-width of a graph is bounded by a given number, is a very intricate combinatorial problem. More precisely, given a graph G and an integer k, deciding whether the clique-width of G is at most k is NP-complete [Fellows et al. 2009]. Even worse, the clique-width of a graph with n vertices of degree greater than 2 cannot be approximated by a polynomial-time algorithm with an absolute error guarantee of n^ϵ unless P = NP, where $0 \leq \epsilon < 1$ [Fellows et al. 2009]. In fact, it is even unknown whether graphs of clique-width at most 4 can be recognized in polynomial time [Corneil et al. 2012]. One can decide in exponential time $(2k+1)^n n^{O(1)}$ whether the clique-width of a graph with n vertices is at most k [Wahlström 2011]. There are approximation algorithms with an exponential error that, for fixed k, compute $f(k)$-expressions for graphs of clique-width at most k in polynomial time (where $f(k) = (2^{3k+2} - 1)$ by Oum and Seymour [2006], and $f(k) = 8^k - 1$ by Oum [2008]). Because of this intricacy of this graph invariant, the exact clique-width is not known even for very small graphs.

Our Approach Clique-width via SAT

In this work we propose a new approach to compute the clique-width of graphs. Our approach is based on the recent advancements in the area of propositional satisfiability (SAT) (see, for instance [Gomes et al. 2008; Biere et al. 2009; Sakallah and Marques-Silva 2011; Malik and Zhang 2009]). State-of-the-art solvers are highly scalable and deal routinely with industrial instances with millions of variables. Key application areas are hardware and software verification. SAT solvers have also been successfully applied to various problems in combinatorial design [Zhang 2009].

We present a new method for determining the clique-width based on a sophisticated SAT encoding that makes use of the following two main ingredients.

(1) *Reformulation*. The conventional construction method for determining the clique-width of a graph consists of many steps. In the worst case, the number of steps is quadratic in the number of vertices. Translating this construction method to SAT results in large instances. We propose a reformulation of the problem in such a way that the number of steps is less than the number of vertices. The alternative construction method allows us to compute the clique-width of much larger graphs.
(2) *Representative encoding*. Applying the frequently-used direct encoding [Walsh 2000] on the reformulation results in instances where unit propagation finds conflicts much later than required. We develop a new encoding that is compact and avoids this shortcoming (it realizes arc consistency in the sense of [Gent 2002]).

Experimental Results

The implementation of our method allows us for the first time to determine the exact clique-width of various graphs, including famous graphs known from the literature and random graphs of various density.

(1) *Clique-width of small random graphs*. We determined experimentally how the clique-width of random graphs depends on the density. The clique-width is small for dense and sparse graphs and reaches its maximum for edge-probability 0.5. The larger n, the steeper the increase towards 0.5. These results complement the asymptotic results of Lee et al. [2012].
(2) *Smallest graphs of certain clique-width*. In general it is not known how many vertices are required to form a graph of a certain clique-width. We provide these numbers for clique-width $k \in \{1, \ldots, 7\}$. In fact, we could compute the exact number of different connected graphs (modulo isomorphism) with a certain clique-width with up to 10 vertices. For instance, there are only 7 connected graphs with 8 vertices and clique-width 5 (modulo isomorphism), and 68 connected graphs with 10 vertices and clique-width 6.
(3) *Clique-width of famous named graphs*. Over the last 50 years, researchers in graph theory have considered a large number of special graphs. These special graphs have been used as counterexamples for conjectures or for showing the tightness of combinatorial results. We considered 21 prominent graphs from the literature and computed their exact clique-width. These results may be of interest for people working in combinatorics and graph theory. We provide certificates for tight clique-width upper bounds in an appendix.

Linear Clique-Width

Linear clique-width is a variant of clique-width, introduced by Gurski and Wanke [2005], that has received significant attention, see, e.g., [Adler and Kanté 2013; Courcelle and Olariu 2000; Fellows et al. 2009; Heggernes et al. 2011; 2012; Lozin and Rautenbach 2007]. The definition of linear clique-width is based on the same graph construction process as clique-width but with the restriction that whenever a disjoint union is performed, at most one of the two graphs put together contains more than one vertex. Hence the overall process can be seen as a linear sequence of graphs. Consequently, for every graph the linear clique-width is equal or larger than the clique-width. We extend our methods from clique-width to linear clique-width.

(4) *Reformulation and SAT encoding of linear clique-width*. We show that a modification of our reformulation of clique-width provides a reformulation for linear clique-width. In turn, this provides an efficient SAT encoding of linear clique-width.
(5) *Linear Clique-width of Famous Named Graphs*. We tested the SAT encoding of linear clique-width on the 21 famous named graphs considered above. Interestingly, it turned out that the difference between clique-width and linear clique-width is small: for only one of the considered graphs the difference is 2, for six of the considered graphs it is 1, and for the remaining graphs, linear clique-width and clique-width coincide.

1.1. Related Work

We are not aware of any implemented algorithms that compute clique-width exactly. Several algorithms have been implemented that compute upper bounds on other width-based graph invariants, including *treewidth* [Dow and Korf 2007; Gogate and Dechter 2004; Koster et al. 2001], *branchwidth* [Smith et al. 2012], *Boolean-width* [Hvidevold et al. 2012], and *rank-width* [Beyß 2013]. Durand and Courcelle [2013] mention a program they used to obtain upper bounds on the clique-width of some graphs; however, they do not provide any details on the program or its performance. Samer and Veith [2009] proposed a SAT encoding for the exact computation of treewidth. Boolean-width and rank-width can be used to approximate clique-width, however, the error can be exponential in the clique-width; in contrast, treewidth and branchwidth can be arbitrarily far from the clique-width, hence the approximation error is unbounded [Bui-Xuan et al. 2011].

Our SAT encoding is based on a new characterization of clique-width that is based on partitions instead of labels. A similar partition-based characterization of clique-width was proposed by Heggernes et al. [2011]. There are two main differences to our reformulation. Firstly, our characterization of clique-width uses three individual properties that can be easily expressed by clauses. Secondly, our characterization admits the "parallel" processing of several parts of the graph that are later joined together.

2. PRELIMINARIES

2.1. Formulas and Satisfiability

We consider propositional formulas in Conjunctive Normal Form (*CNF formulas*, for short), which are conjunctions of clauses, where a clause is a disjunction of literals, and a literal is a propositional variable or a negated propositional variables. A CNF formula is *satisfiable* if its variables can be assigned true or false (denoted 1 or 0, respectively), such that each clause contains either a variable set to true or a negated variable set to false. The satisfiability problem (SAT) asks whether a given formula is satisfiable.

2.2. Graphs and Clique-width

All graphs considered are finite, undirected, and without self-loops. We denote a graph G by an ordered pair $(V(G), E(G))$ of its set of vertices and its set of edges, respectively. An edge between vertices u and v is denoted uv or equivalently vu. For basic terminology on graphs we refer to a standard text book [Diestel 2000].

Let k be a positive integer. A *k-graph* is a graph whose vertices are labeled by integers from $\{1, \ldots, k\}$. We call the k-graph consisting of exactly one vertex v (say, labeled by i) an *initial* k-graph and denote it by $i(v)$. The *clique-width* of a graph G, denoted $\text{cwd}(G)$, is the smallest integer k such that G can be constructed from initial k-graphs by means of repeated application of the following three operations.

(1) Disjoint union (denoted by $\oplus$);
(2) Relabeling: changing all labels i to j (denoted by $\rho_{i\to j}$);
(3) Edge insertion: connecting all vertices labeled by i with all vertices labeled by $j, i \neq j$ (denoted by $\eta_{i,j}$ or $\eta_{j,i}$); already existing edges are not doubled.

A construction of a k-graph using the above operations can be represented by an algebraic term composed of $\oplus$, $\rho_{i\to j}$, and $\eta_{i,j}$ ($i, j \in \{1, \ldots, k\}$, and $i \neq j$). Such a term is called a *k-expression* defining G. Thus, the clique-width of a graph G is the smallest integer k such that G can be defined by a k-expression.

EXAMPLE 2.1. The graph $P_4 = (\{a, b, c, d\}, \{ab, bc, cd\})$ is defined by the 3-expression

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

Hence $\text{cwd}(P_4) \leq 3$. In fact, one can show that P_4 has no 2-expression, and thus $\text{cwd}(P_4) = 3$ [Courcelle and Olariu 2000]. ◻

2.3. Partitions

As partitions play an important role in our reformulation of clique-width, we recall some basic terminology. A *partition* of a set S is a set P of nonempty subsets of S such that any two sets in P are disjoint and S is the union of all sets in P. The elements of P are called *equivalence classes*. Let P, P' be partitions of S. Then P' is a *refinement* of P if for any two elements $x, y \in S$ that are in the same equivalence class of P' are also in the same equivalence class of P (this entails the case $P = P'$).

3. A REFORMULATION OF CLIQUE-WIDTH WITHOUT LABELS

Initially, we developed a SAT encoding of clique-width based on k-expressions. Even after several optimization steps, we were only able to determine the clique-width of graphs consisting of up to 8 vertices. We therefore developed a new encoding based on a reformulation of clique-width which does not use k-expressions. In this section we explain this reformulation; in the next section we will discuss how it can be encoded into SAT efficiently.

Consider a finite set V of vertices, the *universe*. A *template* T consists of two partitions $\mathrm{cmp}(T)$ and $\mathrm{grp}(T)$ of V. We call the equivalence classes in $\mathrm{cmp}(T)$ the *components* of T and the equivalence classes in $\mathrm{grp}(T)$ the *groups* of T. For some intuition about these concepts, imagine that components represent induced subgraphs, and groups represent sets of vertices in some component with the same label. A *derivation* of length t is a sequence $\mathcal{D} = (T_0, \dots, T_t)$ satisfying the following conditions.

D1 $|\mathrm{cmp}(T_0)| = |V|$ and $|\mathrm{cmp}(T_t)| = 1$.
D2 $\mathrm{grp}(T_i)$ is a refinement of $\mathrm{cmp}(T_i)$, $0 \le i \le t$.
D3 $\mathrm{cmp}(T_{i-1})$ is a refinement of $\mathrm{cmp}(T_i)$, $1 \le i \le t$.
D4 $\mathrm{grp}(T_{i-1})$ is a refinement of $\mathrm{grp}(T_i)$, $1 \le i \le t$.

We would like to note that D1 and D2 together imply that $|\mathrm{grp}(T_0)| = |V|$. Thus, in the first template T_0 all equivalence classes (groups and components) are singletons, and when we progress through the derivation, some of these sets are merged, until all components are merged into a single component in the last template T_t.

The *width* of a component $c \in \mathrm{cmp}(T)$ is the number of groups $g \in \mathrm{grp}(T)$ such that $g \subseteq c$. The width of a template is the maximum width over its components, and the width of a derivation is the maximum width over its templates. A *k-derivation* is a derivation of width at most k. A derivation $\mathcal{D} = (T_0, \dots, T_t)$ is a derivation *of* a graph $G = (V, E)$ if V is the universe of the derivation and the following three conditions hold for all $1 \le i \le t$.

Edge Property: For any two vertices $u, v \in V$ such that $uv \in E$, if u, v are in the same group in T_i, then u, v are in the same component in T_{i-1}.

Neighborhood Property: For any three vertices $u, v, w \in V$ such that $uv \in E$ and $uw \notin E$, if v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.

Path Property: For any four vertices $u, v, w, x \in V$, such that $uv, uw, vx \in E$ and $wx \notin E$, if u, x are in the same group in T_i and v, w are in the same group in T_i, then u, v are in the same component in T_{i-1}.

In a certain sense, the edge property makes sure that a k-expression corresponding to the k-derivation contains some position where the edge uv can be inserted with an η-operation. The neighborhood property and the path property ensure that this η-operation does not introduce unwanted edges. These considerations are made more precise in the proofs of Lemmas 3.4 and 3.6.

The neighborhood property and the path property could be combined into a single property by not insisting that all mentioned vertices are distinct. However, two separate properties provide a more compact SAT encoding.

The following example illustrates that a derivation can define more than one graph, in contrast to a k-expression, which defines exactly one graph.

EXAMPLE 3.1. Consider the derivation $\mathcal{D} = (T_0, \dots, T_3)$ with universe $V = \{a, b, c, d\}$ and

$$\begin{aligned}
\text{cmp}(T_0) &= \{\{a\}, \{b\}, \{c\}, \{d\}\}, & \text{grp}(T_0) &= \{\{a\}, \{b\}, \{c\}, \{d\}\},\\
\text{cmp}(T_1) &= \{\{a, b\}, \{c\}, \{d\}\}, & \text{grp}(T_1) &= \{\{a\}, \{b\}, \{c\}, \{d\}\},\\
\text{cmp}(T_2) &= \{\{a, b, c\}, \{d\}\}, & \text{grp}(T_2) &= \{\{a\}, \{b\}, \{c\}, \{d\}\},\\
\text{cmp}(T_3) &= \{\{a, b, c, d\}\}, & \text{grp}(T_3) &= \{\{a, b\}, \{c\}, \{d\}\}.
\end{aligned}$$

The width of $\mathcal{D}$ is 3. Consider the graph $G = (V, \{ab, ad, bc, bd\})$. To see that $\mathcal{D}$ is a 3-derivation of G, we need to check the edge, neighborhood, and path properties. We observe that a, b are the only two vertices such that $ab \in E(G)$ and both vertices appear in the same group of some T_i (here, we have $i = 3$). To check the edge property, we only need to verify that a, b are in the same component of T_2, which is true. For the neighborhood property, the only relevant choice of three vertices is a, b, c ($bc \in E(G)$, $ac \notin E(G)$, and a, b in a group of T_3). The neighborhood property requires that b, c are in the same component in T_2, which is the case. The path property is satisfied since there is no template in which two pairs of vertices belong to the same group, respectively.

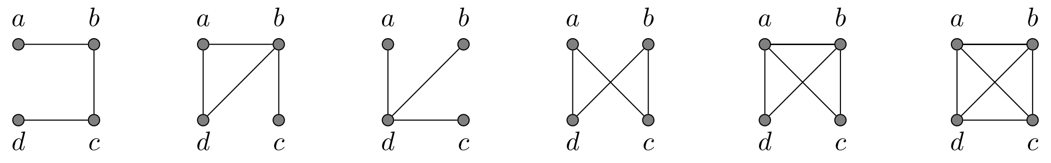

Fig. 1. All connected graphs with four vertices (up to isomorphism). The 3-derivation of Example 3.1 defines all six graphs. The clique-width of all but the first graph is 2.

Similarly we can verify that $\mathcal{D}$ is a derivation of the graph $G' = (V, \{ab,\ bc,\ cd\})$. In fact, for all connected graphs with four vertices, there exists an isomorphic graph that is defined by $\mathcal{D}$ (see Figure 1). However, $\mathcal{D}$ is not a derivation of the graph $G'' = (V, \{ab, ac, bd, cd\})$ since the neighborhood property is violated: $bd \in E(G'')$ and $ad \notin E(G'')$, a, b belong to the same group in T_3, while a, d do not belong to the same component in T_2. $\square$

We call a derivation $(T_0, \dots, T_t)$ *strict* if $|\text{cmp}(T_{i-1})| > |\text{cmp}(T_i)|$ holds for all $1 \leq i \leq t$.

LEMMA 3.2. *Every k-derivation of a graph G contains as subsequence a strict k-derivation of G.*

PROOF. Let $\mathcal{D} = (T_0, \dots, T_t)$ be a k-derivation of G. Assume there is some $1 \leq i \leq t$ such that $\text{cmp}(T_{i-1}) = \text{cmp}(T_i)$. If also $\text{grp}(T_{i-1}) = \text{grp}(T_i)$, then $T_{i-1} = T_i$, and we can safely remove T_{i-1} and still have a k-derivation of G. Hence assume $\text{grp}(T_{i-1}) \neq \text{grp}(T_i)$. This implies that $i > 1$. If $i = t$, then we can safely remove T_t from the derivation and $(T_0, \dots, T_{t-1})$ is clearly a k-derivation of G. Hence it remains to consider the case $1 < i \leq t - 1$. We show that by dropping T_i we get a sequence $\mathcal{D}' = (T_0, \dots, T_{i-1}, T_{i+1}, \dots, T_t)$ that is a k-derivation of G.

The new sequence $\mathcal{D}'$ is clearly a k-derivation. It remains to verify that $\mathcal{D}'$ is a derivation of G. The template T_{i+1} is the only one where these properties might have been violated by the removal of T_i. However, since all three properties impose a restriction on the set of components of the template preceding T_{i+1}, and since $\text{cmp}(T_{i-1}) = \text{cmp}(T_i)$, the properties are not affected by the deletion of T_i. Hence $\mathcal{D}'$ is indeed a k-derivation of G.

By repeated application of the above shortening we can turn any k-derivation into a strict k-derivation. $\square$

LEMMA 3.3. *Every strict k-derivation of a graph with n vertices has length at most $n-1$.*

PROOF. Let $(T_0,\dots,T_t)$ be a strict k-derivation of a graph with n vertices. Since $|\text{cmp}(T_0)| = n$ and $|\text{cmp}(T_0)| = 1$, it follows that $t \le n-1$. $\square$

In the proofs of the next two lemmas we need the following concept of a *k-expression tree*, which is the parse tree of a k-expression equipped with some additional information. Let ϕ be a k-expression for a graph $G=(V,E)$. Let Q be the parse tree of ϕ with root r. That is, Q contains a node for each occurrence of an operation $\oplus$, $\rho_{i\to j}$, and $\eta_{i,j}$ in ϕ and for each initial k-graph $i(v)$ in ϕ; the initial k-graphs are the leaves of Q, and the other nodes have as children the nodes which represent the two subexpressions of the respective operation. Consider a node q of Q and let ϕ_q be the subexpression of ϕ whose parse tree is the subtree of Q rooted at q. Then q is labeled with the k-graph G_q constructed by the k-expression ϕ_q. Thus the leaves of Q are labeled with initial k-graphs and the root r is labeled with a labeled version of G. We call a non-leaf node of Q an $\oplus$-node, η-node, or ρ-node, according to the operation it represents.

Figure 2 exhibits the 3-expression tree of the 3-expression of Example 2.1. For instance, the 3-graph G_{q_3} has as vertices a, b, c and edges ab, bc where a and b have label 2, and c has label 3.

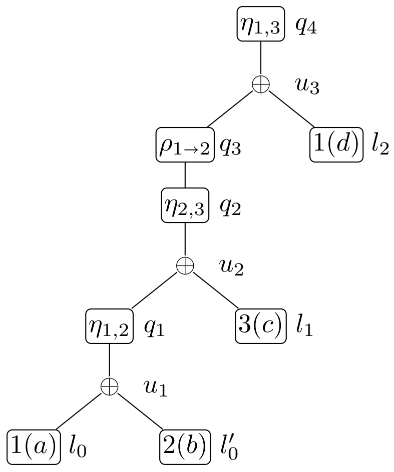

Fig. 2. 3-expression tree for the 3-expression of Example 2.1.

One $\oplus$-node of the parse tree can represent several directly subsequent $\oplus$-operations (e.g., the operation $(x \oplus y) \oplus z$ can be represented by a single node with three children). For technical reasons we will also allow $\oplus$-nodes with a single child.

Each k-expression gives rise to a k-expression tree where each $\oplus$-node has no $\oplus$-nodes as children, let us call such a k-expression tree to be *succinct*. Evidently, k-expressions and their (succinct) k-expression trees can be effectively transformed into each other.

LEMMA 3.4. *From a k-expression of a graph G we can obtain a k-derivation of G in polynomial time.*

PROOF. Let ϕ be a k-expression of $G = (V, E)$ and let Q be the corresponding succinct k-expression tree with root r. For a node $q \in V(Q)$ let $R(q)$ denote the number of $\oplus$-nodes that appear on the path from r to q. We write U and L for the set of $\oplus$-nodes and the set of leaves of Q, respectively. We let $t := \max_{q \in L} R(q)$. For $0 \leq i \leq t$ we define $U_i = \{\, q \in U : R(q) = t - i + 1 \,\}$ and $L_i = \{\, q \in L : R(q) < t - i + 1 \,\}$. Example 3.5 shows the sets U_i, L_i for the k-expression tree of Figure 2. We observe that for each $v \in V$ and $1 \leq i \leq t$ there is exactly one $q \in U_i \cup L_i$ such that $v \in G_q$.

We define a derivation $\mathcal{D} = (T_0, \dots, T_t)$ as follows. For $0 \leq i \leq t$ we put cmp$(T_i) = \{\, V(G_q) : q \in U_i \cup L_i \,\}$ and grp$(T_i) = \bigcup_{q \in U_i \cup L_i}$ grp(G_q) where grp(G_q) denotes the partition of $V(G_q)$ into sets of vertices that have the same label. By construction, $\mathcal{D}$ is a derivation with universe V. Furthermore, since ϕ is a k-expression, $|\text{grp}(G_q)| \leq k$ for all nodes q of Q. Hence $\mathcal{D}$ is a k-derivation. It remains to show that $\mathcal{D}$ is a k-derivation of G. Let $1 \leq i \leq t$.

To show that the *edge property* holds, consider two vertices $u, v \in V$ such that $uv \in E$ and u, v are in the same group in T_i. Assume to the contrary that u, v belong to different components c_1, c_2 in T_{i-1}. Since u, v are in the same group in T_i, they are also in the same component of T_i. Hence there is an $\oplus$-node $q \in U_i$ with $u, v \in V(G_q) \in$ cmp(T_i). Let q_1, q_2 be the children of q with $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. Hence $uv \notin E(G_{q_1}) \cup E(G_{q_2})$. However, since u, v are in the same group in T_i, this means that u, v have the same label in G_q. Thus the edge uv cannot be introduced by an η-operation, and so $uv \notin E(G_r) = E$, a contradiction. Hence the edge property holds.

To show that the *neighborhood property* holds, consider three vertices $u, v, w \in V$ such that $uv \in E$, $uw \notin E$, and v, w are in the same group of T_i. Assume to the contrary that u, v are in different components of T_{i-1}, say in components c_1 and c_2, respectively. Since v, w are in the same group of T_i, they are also in the same component c of T_i. Let $q \in U_i$ be the $\oplus$-node such that $v, w \in V(G_q) = c \in$ cmp(T_i), and let q_1, q_2 be the children of q with $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. Clearly $uv \notin E(G_{q_1}) \cap E(G_{q_2})$, hence there must be an η-node p somewhere on the path between q and r where the edge uv is introduced. However, since v and w share the same label in G_q, they share the same label in G_p. Consequently, the η-operation that introduces the edge uv also introduces the edge uw. However, this contradicts the assumption that $uw \notin E$. Hence the neighborhood property holds as well.

To show that the *path property* holds, we proceed similarly. Consider four vertices $u, v, w, x \in V$, such that $uv, uw, vx \in E$ and $wx \notin E$. Assume that u, x are in the same group in T_i and v, w are in the same group in T_i. Assume to the contrary that u, v are in different components of T_{i-1}, say in components c_1 and c_2, respectively. Above we have shown that the neighborhood property holds. Hence we conclude that u, w belong to the same component of T_{i-1}, and v, x belong to the same component of T_{i-1}. Since u, x are in the same group in T_i, they are also in the same component of T_i, say in component c. Since u, w belong to the same component of T_{i-1}, they also belong to the same component of T_i, thus $w \in c$. By a similar argument we conclude that $v \in c$. Thus all four vertices u, v, w, x belong to c. Let $q \in U_i$ be the $\oplus$-node with $V(G_q) = c \in$ cmp(T_i), and let q_1, q_2 be the children of q with $V(G_{q_1}) = c_1$ and $V(G_{q_2}) = c_2$. Clearly $uv \notin E(G_{q_1}) \cup E(G_{q_2})$, hence there must be an η-node p somewhere on the path between q and r where the edge uv is introduced. However, since v and w share the same label in G_q, and u and x share the same label in G_q, this also holds in G_p. Hence the η-operation that introduces the edge uv also introduces the edge xw. However, this contradicts the assumption that $xw \notin E$. Hence the path property holds as well. We conclude that $\mathcal{D}$ is indeed a k-derivation of G.

The above procedure for generating the k-derivation can clearly be carried out in polynomial time. $\square$

EXAMPLE 3.5. Consider the 3-expression ϕ for the graph P_4 of Example 2.1 and the 3-expression tree in Figure 2. We apply the procedure described in the proof of Lemma 3.4. We have:

$$\begin{aligned} U_0 &= \emptyset, & L_0 &= \{l_0, l'_0, l_1, l_2\}, \\ U_1 &= \{u_1\}, & L_1 &= \{l_1, l_2\}, \\ U_2 &= \{u_2\}, & L_2 &= \{l_2\}, \\ U_3 &= \{u_3\}, & L_3 &= \emptyset. \end{aligned}$$

From that we obtain the 3-derivation $\mathcal{D}$ of Example 3.1. □

LEMMA 3.6. *From a k-derivation of a graph G we can obtain a k-expression of G in polynomial time.*

PROOF. Let $\mathcal{D} = (T_0, \ldots, T_t)$ be a k-derivation of $G = (V, E)$. Using the construction of the proof of Lemma 3.2 we can obtain a strict k-derivation of G from any given k-derivation of G. Hence we may assume, w.l.o.g., that $\mathcal{D}$ is strict. Let $C = \bigcup_{i=0}^{t} \text{cmp}(T_i)$. We are going to construct in polynomial time a k-expression tree for G, which can clearly be turned into a k-expression for G in polynomial time.

We proceed in three steps, see Figure 3 for an illustration.

First we construct a k-expression tree $Q_\oplus$ that only contains $\oplus$-nodes and leaves. For each component $c = \{v\}$ of T_0 we introduce a leaf $q(c, 0)$ with label $1(v)$. For each $1 \le i \le t$ and each component $c \in \text{cmp}(T_i)$ we introduce an $\oplus$-node $q(c, i)$. We add edges to $Q_\oplus$ such that $q(c', i-1)$ is a child of $q(c, i)$ if and only if $c' \subseteq c$. Properties D1 and D3 of a derivation ensure that $Q_\oplus$ is a tree. Note that $Q_\oplus$ is not necessarily succinct, and may contain $\oplus$-nodes that have only one child.

In the next step we add to $Q_\oplus$ certain ρ-nodes to obtain the k-expression tree $Q_{\oplus,\rho}$. We visit the $\oplus$-nodes of $Q_\oplus$ in a depth-first ordering. Let $q(c, i)$ be the currently visited node. Between $q(c, i)$ and each child $q(c', i-1)$ of $q(c, i)$ we add at most k ρ-nodes (the edge between $q(c, i)$ and $q(c', i-1)$ becomes a path) such that afterwards $q(c, i)$ has a child q' with $\text{grp}(G_{q'}) = \{\, g \in \text{grp}(G_{q(c,i)}) : g \subseteq c \,\} \subseteq \text{grp}(T_i)$. This is possible because of properties D2 and D4 of a derivation and since a partition can be obtained from any of its refinements by a sequence of steps each merging two equivalence classes.

As a final step, we add η-nodes to $Q_{\oplus,\rho}$ and obtain the k-expression tree Q. Let $uv \in E$ be an edge of G. We show that there is an $\oplus$-node q in $Q_{\oplus,\rho}$ above which we can add an η-node p (q is a child of p) which introduces edges including uv but does not introduce any edge not present in E .

Let $q(c, i)$ be the $\oplus$-node of $Q_{\oplus,\rho}$ with smallest i such that $u, v \in V(G_{q(c,i)})$. We write $q = q(c, i)$ and $c = V(G_q)$ and observe that $c \in \text{cmp}(T_i)$. Among the children of q are two distinct nodes q_1, q_2 such that $u \in V(G_{q_1})$ and $v \in V(G_{q_2})$. It follows that there are distinct components $c_1, c_2 \in T_{i-1}$ with $u \in c_1$ and $v \in c_2$. By the edge property, u and v belong to different groups of T_i, and so u and v have different labels in G_q, say the labels a and b, respectively. We add an η-node p above q (q is a child of p) representing the operation $\eta_{a,b}$. This inserts the edge uv to G_q. It remains to show that $\eta_{a,b}$ does not add any edges that are not in E. We show this by establishing that for all pairs of vertices $u', v' \in c$ where u' has label a and v' has label b in G_q, the edge $u'v'$ is in E.

We consider four cases.

Case 1: $u = u', v = v'$. Trivially, $u'v' = uv \in E$.

Case 2: $u = u', v \neq v'$. Assume to the contrary that $u'v' \notin E$. Since v and v' have the same label in G_q, they belong to the same group of T_i. The neighborhood property implies that u and v belong to the same component of T_{i-1}, a contradiction to the minimal choice of i. Hence $u'v' \in E$.

Case 3: $u \neq u', v = v'$. This case is symmetric to Case 2.

Case 4: $u \neq u', v \neq v'$. Assume to the contrary that $u'v' \notin E$. It follows from Cases 2 and 3 that $uv', vu' \in E$. The path property implies that u and v belong to the same component of T_{i-1}, a contradiction to the minimal choice of i. Hence $u'v' \in E$.

Consequently, we can successively add η-nodes to $Q_{\oplus,\rho}$ until all edges of E are inserted, but no edge outside of E. Hence we obtain indeed a k-expression tree for G.

This procedure for generating the k-expression tree can clearly be carried out in polynomial time, hence the lemma follows. $\square$

We note that we could have saved some ρ-operations in the proof of Lemma 3.6. In particular the k-expression produced may contain ρ-operations where the number of different labels before and after applying the ρ-operation remains the same. It is easy to see that such ρ-operations can be omitted by changing labels of some initial k-graphs accordingly.

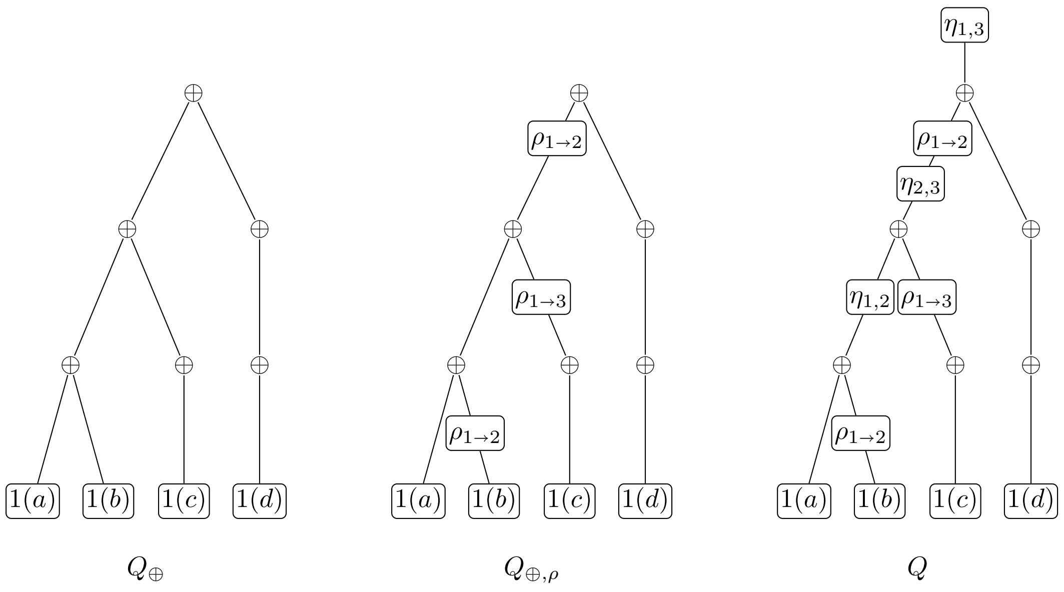

Fig. 3. 3-expression trees obtained by the construction of the proof of Lemma 3.6 applied to the derivation $\mathcal{D}$ of Example 3.1.

By Lemma 3.3 we do not need to search for k-derivations of length $> n-1$ when the graph under consideration has n vertices. The next lemma improves this bound to $n-k+1$ which provides a significant improvement for our SAT encoding, especially when the graph under consideration has large clique-width.

LEMMA 3.7. *Let $1 \leq k \leq n$. If a graph with n vertices has a k-derivation, then it has a k-derivation of length $n-k+1$.*

PROOF. Let $k \geq 1$ be fixed. We define the *k-length* of a derivation as the number of templates that contain at least one component of size larger than k (these templates form a suffix of the derivation). Let $\ell(n,k)$ be the largest k-length of a strict derivation over a universe of size n. Before we show the lemma, we establish three claims. For these claims, the groups of the considered derivations are irrelevant and hence will be ignored.

Claim 1: $\ell(n,k) < \ell(n+1,k)$.

To show the claim, consider a strict derivation $\mathcal{D} = (T_0, \dots, T_t)$ over a universe V of size n with k-length ℓ. We take a new element a and form a strict derivation $\mathcal{D}'$ over the universe $V \cup \{a\}$ by adding the singleton $\{a\}$ to cmp(T_i) for $0 \le i \le t$ and adding a new template T_{t+1} with cmp$(T_{t+1}) = \{V \cup \{a\}\}$. The new derivation $\mathcal{D}'$ has k-length $\ell + 1$.

Claim 2: Let $\mathcal{D} = (T_0, \dots, T_t)$ be a strict derivation over a universe V of size n of k-length $\ell(n,k)$. Then, $T_{t-\ell(n,k)+1}$ has exactly one component of size $k+1$ and all other components are singletons.

We proceed to show the claim. Let $j = t - \ell(n,k)$, and observe that j is the largest index where all components of T_j have size at most k. Let $c_1, \dots, c_r$ be the components of T_{j+1} of size greater than 1 such that $|c_1| \ge |c_2| \ge \dots \ge |c_r|$. Thus $|c_1| > k$. We show that $r = 1$. Assume to the contrary that $r > 1$. We pick some element $a_i \in c_i$, $2 \le i \le r$, and set $X = \bigcup_{i=2}^{r} c_i \setminus \{a_i\}$. The derivation $\mathcal{D}$ induces a strict derivation $\mathcal{D}'$ over the universe $V' = V \setminus X$. Observe that $n' = |V'| < |V| = n$. Evidently $\mathcal{D}'$ has the same k-length as $\mathcal{D}$, hence $\ell(n',k) \ge \ell(n,k)$, a contradiction to Claim 1. Hence $r = 1$, and c_1 is the only component in T_{j+1} of size greater than k, all other components of T_{j+1} are singletons. We show that $|c_1| = k+1$. We assume to the contrary that $|c_1| > k+1$. We pick $k+1$ vertices $b_1, \dots, b_{k+1} \in c_1$ and set $X = c_1 \setminus \{b_1, \dots, b_{k+1}\}$. Similarly as above, we observe that $\mathcal{D}$ induces a strict derivation $\mathcal{D}''$ over the universe $V'' = V' \setminus X$, and that $\mathcal{D}''$ has the same k-length as $\mathcal{D}$. Since $|V''| < |V|$ we have again a contradiction to Claim 1. Hence Claim 2 is established.

Claim 3: $\ell(n,k) \le n - k$.

To see the claim, let $\mathcal{D} = (T_0, \dots, T_t)$ be a strict derivation over a universe V of size n of k-length $\ell(n,k)$. Let $j = t - \ell(n,k)$. By Claim 2 we know that T_{j+1} has exactly one component of size $k+1$ and all other components are singletons (hence there are $n-k-1$ singletons). We conclude that $|\text{cmp}(T_{j+1})| = n-k$. Since $\mathcal{D}$ is strict, we have $n-k = |\text{cmp}(T_{j+1})| > |\text{cmp}(T_{j+2})| > \dots > |\text{cmp}(T_t)| = 1$. Thus $\ell(n,k) = t - j \le n-k$, and the claim follows.

We are now in the position to establish the statement of the lemma. Let $\mathcal{D} = (T_0, \dots, T_t)$ be a k-derivation of a graph $G = (V,E)$ with $|V| = n$. By Lemma 3.2 we may assume that $\mathcal{D}$ is strict. Let ℓ be the k-length of $\mathcal{D}$ and let $j = t - \ell$. By Claim 3 we know that $\ell \le n-k$. We define a new template T_j' with cmp(T_j') = cmp(T_j) and grp(T_j') = grp(T_0), and we set $\mathcal{D}' = (T_0, T_j', T_{j+1}, \dots, T_t)$. We claim that $\mathcal{D}'$ is a k-derivation of G. Clearly $\mathcal{D}'$ is a derivation, but we need to check the edge, neighborhood, and path property for T_j' and T_{j+1} in $\mathcal{D}'$. The properties hold trivially for T_j' since all its groups are singletons. For T_{j+1} the properties hold since T_j' has the same components as T_j. Thus $\mathcal{D}'$ is indeed a k-derivation of G. The length of $\mathcal{D}'$ is $\ell + 1 \le n - k + 1$, hence the lemma follows. □

EXAMPLE 3.8. Again, consider the derivation $\mathcal{D}$ of Example 3.1. $\mathcal{D}$ defines P_4 which has clique-width 3 [Courcelle and Olariu 2000]. According to Lemma 3.7, it should have a derivation of length $n - k + 1 = 4 - 3 + 1 = 2$. We can obtain such a derivation by removing T_1 from $\mathcal{D}$, which gives $\mathcal{D}' = (T_0, T_2, T_3)$. □

By combining Lemmas 3.4, 3.6, and 3.7, we arrive at the main result of this section.

PROPOSITION 3.9. *Let $1 \le k \le n$. A graph G with n vertices has clique-width at most k if and only if G has a k-derivation of length at most $n - k + 1$.*

In Section 7.3, we will show that the bound is tight for some graphs: the Sousselier Graph has 16 vertices and clique-width 6. The shortest possible 6-derivation for this graph has length 11, which is equal to the bound.

4. ENCODING A DERIVATION OF A GRAPH

Let $G = (V, E)$ be a graph and $t > 0$ an integer. We are going to construct a CNF formula $F_{\text{der}}(G, t)$ that is satisfiable if and only if G has a derivation of length t. We assume that the vertices of G are given in some arbitrary but fixed linear order $<$.

For any two distinct vertices u and v of G and any $0 \le i \le t$ we introduce a *component variable* $c_{u,v,i}$. Similarly, for any two distinct vertices u and v of G with $u < v$ and any $0 \le i \le t$ we introduce a *group variable* $g_{u,v,i}$. Intuitively, $c_{u,v,i}$ or $g_{u,v,i}$ are true if and only if u and v are in the same component or group, respectively, in the ith template of an implicitly represented derivation of G.

The formula $F_{\text{der}}(G, t)$ is the conjunction of all the clauses described below.
The following clauses represent the conditions D1–D4:

$$(\bar{c}_{u,v,0}) \wedge (c_{u,v,t}) \wedge (c_{u,v,i} \vee \bar{g}_{u,v,i}) \wedge (\bar{c}_{u,v,i-1} \vee c_{u,v,i}) \wedge (\bar{g}_{u,v,i-1} \vee g_{u,v,i}) \quad \text{for } u, v \in V,\ u < v,\ 0 \le i \le t.$$

We further add clauses that ensure that the relations of being in the same group and of being in the same component are transitive:

$$\begin{aligned}&(\bar{c}_{u,v,i} \vee \bar{c}_{v,w,i} \vee c_{u,w,i}) \wedge (\bar{c}_{u,v,i} \vee \bar{c}_{u,w,i} \vee c_{v,w,i}) \wedge (\bar{c}_{u,w,i} \vee \bar{c}_{v,w,i} \vee c_{u,v,i}) \wedge \\ &(\bar{g}_{u,v,i} \vee \bar{g}_{v,w,i} \vee g_{u,w,i}) \wedge (\bar{g}_{u,v,i} \vee \bar{g}_{u,w,i} \vee g_{v,w,i}) \wedge (\bar{g}_{u,w,i} \vee \bar{g}_{v,w,i} \vee g_{u,v,i}) \\ &\qquad \text{for } u, v, w \in V,\ u < v < w,\ 0 \le i \le t.\end{aligned}$$

In order to enforce the *edge property* we add the following clauses for any two vertices $u, v \in V$ with $u < v$, $uv \in E$ and $1 \le i \le t$:

$$(c_{u,v,i-1} \vee \bar{g}_{u,v,i}).$$

Further, to enforce the *neighborhood property*, we add for any three vertices $u, v, w \in V$ with $uv \in E$ and $uw \notin E$ and $1 \le i \le t$, the following clauses:

$$(c_{\min(u,v),\max(u,v),i-1} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

Finally, to enforce the *path property* we add for any four vertices u, v, w, x, such that $uv, uw, vx \in E$, and $wx \notin E$, $u < v$ and $1 \le i \le t$ the following clauses:

$$(c_{u,v,i-1} \vee \bar{g}_{\min(u,x),\max(u,x),i} \vee \bar{g}_{\min(v,w),\max(v,w),i})$$

The following statement is a direct consequence of the above definitions.

LEMMA 4.1. *$F_{\text{der}}(G, t)$ is satisfiable if and only if G has a derivation of length t.*

5. ENCODING A K-DERIVATION OF A GRAPH

In this section, we describe how the formula $F_{\text{der}}(G, t)$ can be extended to encode a derivation of width at most k. First we will describe the conventional direct encoding [Walsh 2000] followed by our representative encoding which where unit propagation is triggered earlier in case of an inconsistency, and thus supports to find the solution faster.

5.1. Direct Encoding

We introduce new variables $l_{v,a,i}$ for $v \in V$, $1 \le a \le k$, and $0 \le i \le t$. The purpose is to assign each vertex for each template a *group number*, which is an integer between 1 and k. The intended meaning of a variable $l_{v,a,i}$ is that in T_i, vertex v has group number a. Let $F(G, k, t)$ denote the formula obtained from $F_{\text{der}}(G, t)$ by adding the following three sets of clauses. The first ensures that every vertex has at least one group number, the second ensures that every vertex has at most one group number, and the third ensures that two vertices of the same group share the same group number.

$$
\begin{aligned}
&(l_{v,1,i} \vee l_{v,2,i} \vee \cdots \vee l_{v,k,i}) \qquad \text{for } v \in V,\ 0 \le i \le t,\\
&(\bar{l}_{v,a,i} \vee \bar{l}_{v,b,i}) \qquad \text{for } v \in V,\ 1 \le a < b \le k,\ 0 \le i \le t,\\
&(\bar{l}_{u,a,i} \vee \bar{l}_{v,a,i} \vee \bar{c}_{u,v,i} \vee g_{u,v,i}) \wedge (\bar{l}_{u,a,i} \vee l_{v,a,i} \vee \bar{g}_{u,v,i}) \wedge (l_{u,a,i} \vee \bar{l}_{v,a,i} \vee \bar{g}_{u,v,i})\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{for } u, v \in V,\ u < v,\ 1 \le a \le k,\ 0 \le i \le t.
\end{aligned}
$$

Together with Lemma 4.1 this construction directly yields the following statement.

PROPOSITION 5.1. *Let $G = (V, E)$ be graph and $t = |V| - k + 1$. Then $F(G, k, t)$ is satisfiable if and only if* $\mathrm{cwd}(G) \le k$.

EXAMPLE 5.2. Let $G = (V, E)$ and $k = 2$. Vertices $u, v, w \in V$ in template T_i, are in one component, but in different groups. Hence the corresponding component variables are true, and the corresponding group variables are false. The clauses containing the variables $l_{u,a,i}, l_{v,a,i}, l_{w,a,i}$ with $a \in \{1, 2\}$ after removing falsified literals are:

$$
\begin{aligned}
&(l_{u,1,i} \vee l_{u,2,i}) \wedge (l_{v,1,i} \vee l_{v,2,i}) \wedge (l_{w,1,i} \vee l_{w,2,i}) \wedge (\bar{l}_{u,1,i} \vee \bar{l}_{v,1,i}) \wedge (\bar{l}_{u,1,i} \vee \bar{l}_{w,1,i}) \wedge\\
&(\bar{l}_{v,1,i} \vee \bar{l}_{w,1,i}) \wedge (\bar{l}_{u,2,i} \vee \bar{l}_{v,2,i}) \wedge (\bar{l}_{u,2,i} \vee \bar{l}_{w,2,i}) \wedge (\bar{l}_{v,2,i} \vee \bar{l}_{w,2,i})
\end{aligned}
$$

These clauses cannot be satisfied, yet unit propagation will not result in a conflict. Therefore, a SAT solver may not be able to cut off the current branch. □

5.2. Representative Encoding

To overcome the unit propagation problem of the direct encoding, as described in Example 5.2, we propose the *representative encoding* which uses two types of variables. First, for each $v \in V$ and $1 \le i \le t$ we introduce a representative variable $r_{v,i}$. This variable, if assigned to true, expresses that vertex v is the representative of a group in template T_i. In each group, only one vertex can be the representative and we choose to make the first vertex in the lexicographical ordering the representative. This results in the following clauses:

$$
(r_{v,i} \vee \textstyle\bigvee_{u \in V, u<v} g_{u,v,i}) \wedge \textstyle\bigwedge_{u \in V, u<v} (\bar{r}_{v,i} \vee \bar{g}_{u,v,i}) \qquad \text{for } v \in V,\ 0 \le i \le t
$$

Additionally we introduce auxiliary variables to efficiently encode that the number of representative vertices in a component is at most k. These auxiliary variables are based on the *order encoding* [Tamura et al. 2009]. Consider a (non-Boolean) variable $L_{v,i}$ with domain $D = \{1, \dots, k\}$, whose elements denote the group number of vertex v in template T_i. In the direct encoding, we used k variables $l_{v,a,i}$ with $a \in D$. Assigning $l_{v,a,i} = 1$ in that encoding means $L_{v,i} = a$. Alternatively, we can use *order variables* $o^{>}_{v,a,i}$ with $v \in V$, $a \in D \setminus \{k\}$, $0 \le i \le t$. Assigning $o^{>}_{v,a,i} = 1$ means $L_{v,i} > a$. Consequently, $o^{>}_{v,a,i} = 0$ means $L_{v,i} \le a$.

EXAMPLE 5.3. Given an assignment to the order variables $o^{>}_{v,a,i}$, one can easily construct the equivalent assignment to the variables in the direct encoding (and the other way around). Below is a visualization of the equivalence relation with $k = 5$. In the middle is a binary representation of each of the k labels by concatenating the Boolean values to the order variables.

$$\begin{array}{lllllll}
L_v = 1 & \leftrightarrow & l_{v,1,i} = 1 & \leftrightarrow & \texttt{0000} & \leftrightarrow & o^{>}_{v,1,i} = o^{>}_{v,2,i} = o^{>}_{v,3,i} = o^{>}_{v,4,i} = 0 \\
L_v = 2 & \leftrightarrow & l_{v,2,i} = 1 & \leftrightarrow & \texttt{1000} & \leftrightarrow & o^{>}_{v,1,i} = 1, o^{>}_{v,2,i} = o^{>}_{v,3,i} = o^{>}_{v,4,i} = 0 \\
L_v = 3 & \leftrightarrow & l_{v,3,i} = 1 & \leftrightarrow & \texttt{1100} & \leftrightarrow & o^{>}_{v,1,i} = o^{>}_{v,2,i} = 1, o^{>}_{v,3,i} = o^{>}_{v,4,i} = 0 \\
L_v = 4 & \leftrightarrow & l_{v,4,i} = 1 & \leftrightarrow & \texttt{1110} & \leftrightarrow & o^{>}_{v,1,i} = o^{>}_{v,2,i} = o^{>}_{v,3,i} = 1, o^{>}_{v,4,i} = 0 \\
L_v = 5 & \leftrightarrow & l_{v,5,i} = 1 & \leftrightarrow & \texttt{1111} & \leftrightarrow & o^{>}_{v,1,i} = o^{>}_{v,2,i} = o^{>}_{v,3,i} = o^{>}_{v,4,i} = 1
\end{array} \quad \square$$

Although our encoding is based on the variables from the order encoding, we use none of the associated clauses. We implemented the original order [Tamura et al. 2009], which resulted in many long clauses and the performance was comparable to the direct encoding.

Instead, we combine the representative and order variables. Our use of the order variables can be seen as the encoding of a sequential counter [Sinz 2005]. We would like to point out that if u and v are both representative vertices in the same component of template T_i and $u < v$, then $o^{>}_{u,a,i} = 0$ and $o^{>}_{v,a,i} = 1$ must hold for some $1 \leq a < k$. Consequently, $o^{>}_{u,k-1,i} = 0$ (vertex u has not the highest group number in T_i), $o^{>}_{v,1,i} = 1$ (vertex v has not the lowest group number in T_i), and $o^{>}_{u,a,i} \rightarrow o^{>}_{v,a+1,i}$. These constraints can be expressed by the following clauses.

$$\begin{array}{l}
(\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee \bar{o}^{>}_{u,k-1,i}) \wedge (\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee o^{>}_{v,1,i}) \wedge \\
\bigwedge_{1 \leq a < k-1} (\bar{c}_{u,v,i} \vee \bar{r}_{u,i} \vee \bar{r}_{v,i} \vee \bar{o}^{>}_{u,a,i} \vee o^{>}_{v,a+1,i}) \qquad \text{for } u, v \in V,\ u < v,\ 0 \leq i \leq t.
\end{array}$$

EXAMPLE 5.4. Consider a graph $G = (V, E)$ with $u, v, w, x \in V$ and the representative encoding with $k = 3$. We will show that if u,v,w, and x are all in the same component and they are all representatives of their respective group numbers in template T_i, then unit propagation will result in a conflict (because there are four representatives and only three group numbers). Observe that all corresponding component and representative variables are true. This example, with falsified literals removed, contains the clauses $(\bar{o}^{>}_{u,2,i})$, $(\bar{o}^{>}_{u,1,i} \vee \boldsymbol{o^{>}_{v,2,i}})$, $(o^{>}_{v,1,i})$, $(\bar{o}^{>}_{u,2,i})$, $(\bar{o}^{>}_{u,1,i} \vee \boldsymbol{o^{>}_{w,2,i}})$, $(o^{>}_{w,1,i})$, $(\bar{o}^{>}_{u,2,i})$, $(\bar{o}^{>}_{u,1,i} \vee o^{>}_{x,2,i})$, $(o^{>}_{x,1,i})$, $(\bar{o}^{>}_{v,2,i})$, $(\boldsymbol{\bar{o}^{>}_{v,1,i}} \vee \boldsymbol{o^{>}_{w,2,i}})$, $(o^{>}_{w,1,i})$, $(\bar{o}^{>}_{v,2,i})$, $(\boldsymbol{\bar{o}^{>}_{v,1,i}} \vee o^{>}_{x,2,i})$, $(o^{>}_{x,1,i})$, $(\bar{o}^{>}_{w,2,i})$, $(\boldsymbol{\bar{o}^{>}_{w,1,i}} \vee o^{>}_{x,2,i})$, $(o^{>}_{x,1,i})$. Literals that are falsified by unit clauses are shown in bold. Notice that $(\bar{o}^{>}_{v,1,i} \vee o^{>}_{w,2,i})$ is falsified, i.e., a conflicting clause. $\square$

Both the direct and representative encoding require $n(n+k-1)(n-k+2)$ variables. The number of clauses depends on the graph and is in the worst case $\mathcal{O}(n^5 - n^4 k)$.

6. LINEAR CLIQUE-WIDTH

In this section we show that our approach can be used to compute the *linear* clique-width of graphs, a variant of clique-width introduced by Gurski and Wanke [2005]. A k-expression is *linear* if every occurrence of a disjoint union operation $\oplus$ has at least one operand that is an initial k-graph $i(v)$. The *linear clique-width* of a graph G, denoted $\text{lcwd}(G)$, is the smallest integer k such that G can be defined by a linear k-expression. For instance, the 3-expression given in Example 2.1 is linear, hence $\text{lcwd}(P_4) \leq 3$.

The relationship between clique-width and linear clique-width is similar to the relationship between treewidth and pathwidth. The linear clique-width of a graph is always at least as large as its clique-width, but the difference between these two graph invariants can be arbitrarily large [Courcelle and Olariu 2000; Fellows et al. 2009]. It is NP-hard to compute the linear clique-width of a graph [Fellows et al. 2009]. The linear clique-width of trees can be computed in linear time [Adler and Kanté 2013]. Graphs with linear clique-width at most k, for $k \leq 4$, can be recognized in polynomial time, however, the complexity of the recognition problem for fixed $k > 4$ is open [Heggernes et al. 2011; 2012].

6.1. Linear k-Derivations

We call a k-derivation $\mathcal{D} = (T_0, \dots, T_t)$ to be *linear* if for every $0 \leq i \leq t$, $\text{cmp}(T_i)$ contains at most one component of size larger than 1. We refer to such a component in $\text{cmp}(T_i)$ as the *main component* of T_i.

We show "linear versions" of Lemma 3.7 and Proposition 3.9.

LEMMA 6.1. *Let $1 \leq k \leq n$. If a graph with n vertices has a linear k-derivation, then it has a linear k-derivation of length $n - k + 1$.*

PROOF. First we observe that Claims 1–3 of the proof of Lemma 3.7 clearly hold for linear k-derivations by the very same arguments. Let $\mathcal{D} = (T_0, \dots, T_t)$ be a linear k-derivation of a graph $G = (V, E)$ with $|V| = n$; by Lemma 3.2 we may assume that $\mathcal{D}$ is strict. Let ℓ be the k-length of $\mathcal{D}$ and let $j = t - \ell$. By Claim 3 of the proof of Lemma 3.7 we know that $\ell \leq n - k$. We define a new template T'_j with $\text{cmp}(T'_j) = \text{cmp}(T_j)$ and $\text{grp}(T'_j) = \text{grp}(T_0)$, and we set $\mathcal{D}' = (T_0, T'_j, T_{j+1}, \dots, T_t)$. From the proof of Lemma 3.7 it follows that $\mathcal{D}'$ is a k-derivation of G, and since $\mathcal{D}$ was linear, also $\mathcal{D}'$ is linear. Hence the lemma follows. □

PROPOSITION 6.2. *Let $1 \leq k \leq n$. A graph G with n vertices has linear clique-width at most k if and only if G has a linear k-derivation of length at most $n - k + 1$.*

PROOF. Let $G = (V, E)$ be a graph on n vertices and let $k \geq 1$ be an integer. Assume that $\text{lcwd}(G) \leq k$. Hence G has a linear k-expression. We observe that the k-derivation obtained from this k-expression by the construction given at the beginning of the proof of Lemma 3.4 is linear. It follows now from Lemma 6.1 that G has a linear k-derivation of length at most $n - k + 1$.

Conversely, assume G has a linear k-derivation $\mathcal{D}$. Let Q be the k-expression tree obtained by the three steps described in the proof of Lemma 3.6. Since $\mathcal{D}$ is linear, it follows that every $\oplus$-node q of Q has at most one child p such that $|V(G_p)| > 1$. Consequently, Q corresponds to a linear k-expression of G, and thus $\text{lcwd}(G) \leq k$. □

6.2. SAT Encoding of Linear Clique-Width

We extend the SAT encoding described in Section 5 to linear clique-width.

Let $G = (V, E)$ be graph. As above we assume that that the vertices of G are given in some arbitrary but fixed linear order $<$. Let k, t be integers and let $F(G, k, t)$ be the CNF formula as constructed in Section 5, using either the direct, or the representative encoding. $F(G, k, t)$ is satisfiable if and only if G has a k-derivation of length t.

We obtain a formula $L(G, k, t)$ from $F(G, k, t)$ as follows.

For each $v \in V$ and $1 \leq i \leq t$ we introduce a new variable $m_{v,i}$. This variable is true if the component $c \in \text{cmp}(T_i)$ with $v \in c$ is the main component of T_i. We add the following clauses:

$$(\neg c_{u,v,i} \vee m_{u,i}) \wedge (\neg c_{u,v,i} \vee m_{v,i}) \wedge (\neg m_{u,i} \vee \neg m_{v,i} \vee c_{u,v,i}) \qquad \text{for } v \in V, u < v, 1 \leq i \leq t.$$

These clauses enforce that for any two distinct vertices $u, v \in V$ and any $1 \leq i \leq t$ the following holds: u, v belong to the same component $c \in \text{cmp}(T_i)$ if and only if c is the main component of T_i.

This definition allows us to extend Proposition 5.1 as follows.

Table I. Runtimes in seconds of the direct and representative encoding on a random graph with 20 vertices and 95 edges for different values of k. Up to $k = 9$ the formulas are unsatisfiable, afterwards they are satisfiable. Timeout (TO) is 20,000 seconds.

k	3	4	5	6	7	8	9	10	11	12	13	14	15	16
direct	1.39	14.25	101.1	638.5	18,337	TO	TO	TO	TO	30.57	0.67	0.50	0.10	0.10
repres	0.62	2.12	8.14	12.14	33.94	102.3	358.6	9.21	0.40	0.35	0.32	0.29	0.29	0.28

PROPOSITION 6.3. *Let $G = (V, E)$ be graph and $t = |V| - k + 1$. Then $L(G, k, t)$ is satisfiable if and only if* $\text{lcwd}(G) \leq k$.

7. EXPERIMENTAL RESULTS

In this section we report the results we obtained by running our SAT encoding on various classes of graphs.[1] Given a graph $G = (V, E)$, we compute that G has clique-width k by determining for which value of $k \in \{1, \ldots, |V|\}$ it holds that

— $F(G, k, |V| - k + 1)$ is satisfiable and
— $F(G, k - 1, |V| - k + 2)$ is unsatisfiable.

We use the same approach for linear clique-width, taking the formulas $L(G, k, |V| - k + 1)$ and $L(G, k - 1, |V| - k + 2)$. We used the SAT solver `Glucose` version 2.2 [Audemard and Simon 2009] to solve the encoded problems. `Glucose` solved the hardest instances about twice as fast (or more) as other state-of-the-art solvers such as `Lingeling` [Biere 2012], `Minisat` [Eén and Sörensson 2004] and `Clasp` [Gebser et al. 2007]. We used a 4-core Intel Xeon CPU E31280 3.50GHz, 32 Gb RAM machine running Ubuntu 10.04.

Although the direct and representative encodings result in CNF formulas of almost equal size, there is a huge difference in costs to solve these instances. To determine the clique-width of the famous named graphs (see below) using the direct encoding takes about two to three orders of magnitude longer when compared to the representative encoding. For example, using the representative encoding, we can establish that the Paley graph with 13 vertices has clique-width 9 within a few seconds, while the solver requires over an hour using the direct encoding. Because of the huge difference in speed, we discard the use of the direct encoding in the remainder of this section.

We noticed that upper bounds (satisfiable formulas) are obtained much faster than lower bounds (unsatisfiable formulas). The reason is twofold. First, the whole search space needs to be explored for lower bounds, while for upper bounds, one can be "lucky" and find a solution fast. Second, due to our encoding, upper bound formulas are smaller (due to a smaller t) which makes them easier. Table I shows this for a random graph with 20 vertices for the direct encoding and the representative encoding.

We examined whether adding symmetry-breaking predicates could improve performance. We used `Saucy` version 3 for this purpose [Katebi et al. 2012]. After the addition of the clauses with representative variables, the number of symmetries is drastically reduced. However, one can generate symmetry-breaking predicates for $F_{\text{der}}(G, t)$ and add those instead. Although it is helpful in some cases, the average speed-up was between 5 to 10%.

7.1. Random Graphs

The clique-width of random graphs has been studied by Lee et al. [2012]. Their results show that for random graphs on n vertices the following holds asymptotically almost surely: If the graphs are very sparse, with an edge probability below $1/n$, then the clique-width is at most 5; if the edge probability is larger than $1/n$, then the clique-

[1]The sources of the encoding are available on https://bitbucket.org/mjhheule/cwd-encode/.

width grows at least linearly in n. Our first group of experiments complements these asymptotic results and provides a detailed picture on the clique-width of small random graphs. We have used the SAT encoding to compute the clique-width of graphs with 10, 15, and 20 vertices, with the edge probability ranging from 0 to 1. A plot of the distribution is displayed in Figure 4. It is interesting to observe the symmetry at edge probability $1/2$, and how the steepness of the curve increases with the number of vertices. Note the "shoulders" of the curve for very sparse and very dense graphs.

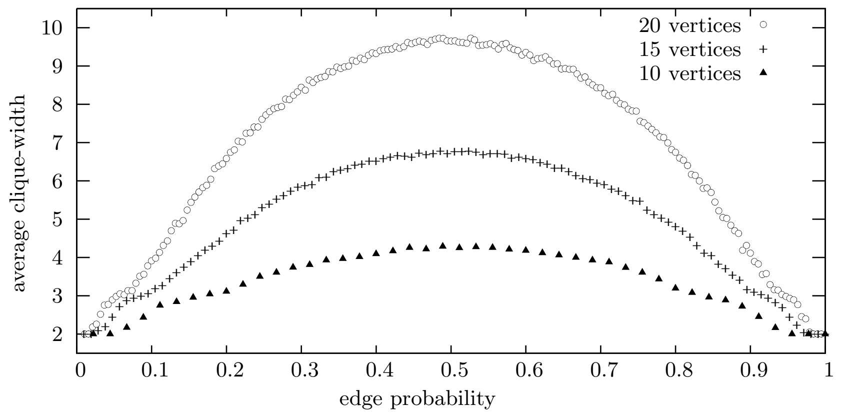

Fig. 4. Average clique-width of random graphs with edge probabilities between 0 and 1. Each dot in the graph represents the average clique-width of 100 graphs.

7.2. The Clique-Width Numbers

For every $k > 0$, let n_k denote the smallest number such that there exists a graph with n_k vertices that has clique-width k. We call n_k the kth *clique-width number*. From the characterizations known for graphs of clique-width 1, 2, and 3, respectively [Corneil et al. 2012], it is easy to determine the first three clique-width numbers (1, 2, and 4). However, determining n_4 is not straightforward, as it requires nontrivial arguments to establish clique-width lower bounds. We would like to point out that a similar sequence of numbers for the graph invariant *treewidth* is easy to determine, as the complete graph on n vertices is the smallest graph of treewidth $n - 1$. One of the very few known graph classes of unbounded clique-width for which the exact clique-width can be determined in polynomial time are grids [Heggernes et al. 2011]; the $k \times k$ grid with $k \geq 3$ has clique-width $k + 1$ [Golumbic and Rotics 2000]. Hence grids provide the upper bounds $n_4 \leq 9$, $n_5 \leq 16$, $n_6 \leq 25$, and $n_7 \leq 36$. With our experiments we have determined that $n_4 = 6$, $n_5 = 8$, $n_6 = 10$, $n_7 = 11$, $n_8 \leq 12$, and $n_9 \leq 13$. It is known that the path on four vertices (P_4) is the unique smallest graph in terms of the number of vertices with clique-width 3. We could determine that the triangular prism (3-Prism) is the unique smallest graph with clique-width 4, and that there are exactly 7 smallest graphs with clique-width 5. There are 68 smallest graphs with clique-width 6 and one of them has only 18 edges. See Figure 5 for an illustration. Additionally, we found several graphs with 11 vertices and clique-width 7 by extending a graph with 10 vertices and clique-width 6.

PROPOSITION 7.1. *The first seven clique-width numbers are* 1, 2, 4, 6, 8, 10, 11.

We used the software package Nauty [McKay 1981] to avoid checking isomorphic copies of the same graph. There are several other preprocessing methods that can speed up the search for small graphs of prescribed clique-width $k \geq 2$. Obviously, we can limit the search to *connected* graphs, as the clique-width of a graph is clearly the maximum clique-width of its connected components. We can also ignore graphs that contain *twins*—two vertices that have exactly the same neighbors—as we can delete one of them without changing the clique-width. Similarly, we can ignore graphs with a *universal vertex*, a vertex that is adjacent to all other vertices, as it can be deleted without changing the clique-width. All these filtering steps are subsumed by the general concept of *prime graphs*. Consider a graph $G = (V, E)$. A vertex $u \in V$ *distinguishes* vertices $v, w \in V$ if $uv \in E$ and $uw \notin E$. A set $M \subseteq V$ is a *module* if no vertex from $V \setminus M$ distinguishes two vertices from M. A module M is *trivial* if $|M| \in \{0, 1, |V|\}$. A graph is *prime* if it contains only trivial modules. It is well known that the clique-width of a graph is either 2 or the maximum clique-width of its induced prime subgraphs [Courcelle and Olariu 2000]. Hence, in our search, we can ignore all graphs that are not prime. We can efficiently check whether a graph is prime [Habib and Paul 2010]. The larger the number of vertices, the larger the fraction of non-prime graphs (considering connected graphs modulo isomorphism). Table II gives detailed results.

Table II. Number of connected and prime graphs with specified clique-width, modulo isomorphism.

			clique-width				
$\|V\|$	connected	prime	2	3	4	5	6
4	6	1	0	1	0	0	0
5	21	4	0	4	0	0	0
6	112	26	0	25	1	0	0
7	853	260	0	210	50	0	0
8	11,117	4,670	0	1,873	2,790	7	0
9	261,080	145,870	0	16,348	125,364	4,158	0
10	11,716,571	8,110,354	0	142,745	5,520,350	2,447,190	68

7.3. Famous Named Graphs

The graph theoretic literature contains several graphs that have names, sometimes inspired by the graph's topology, and sometimes named after their discoverer. We have computed the clique-width of several named graphs, the results are given in Table III (definitions of all considered graphs can be found in MathWorld [Weisstein]). The *Paley graphs*, named after the English mathematician Raymond Paley (1907–1933), stick out as having large clique-width. Our results on the clique-width of Paley graphs imply some upper bounds on the 9th and 11th clique-width numbers: $n_9 \leq 13$ and $n_{11} \leq 17$. For the Petersen graph and the McGee graph, we identified the exact clique-width of 5 and 6, respectively (we provide k-expressions for these graphs in the appendix). This improves upon the upper bounds provided by Durand and Courcelle [2013] of 6 and 10, respectively.

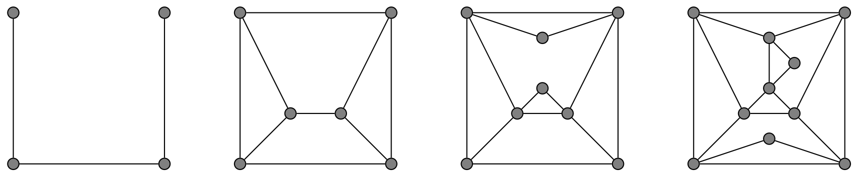
Fig. 5. Smallest graphs with clique-width 3, 4, 5, and 6 (from left to right).

Table III. Clique-width of named graphs. Sizes are reported for the unsatisfiables.

graph	$\|V\|$	$\|E\|$	cwd	length	variables	clauses	UNSAT	SAT
Brinkmann	21	42	10	7 (-5)	8,526	163,065	3,932.56	1.79
Chvátal	12	24	5	5 (-3)	1,800	21,510	0.40	0.09
Clebsch	16	40	8	3 (-6)	3,872	60,520	191.02	0.09
Desargues	20	30	8	5 (-8)	7,800	141,410	3,163.70	0.26
Dodecahedron	20	30	8	7 (-6)	7,800	141,410	5,310.07	0.33
Errera	17	45	8	4 (-6)	4,692	79,311	82.17	0.16
Flower Snark	20	30	7	5 (-9)	8,000	148,620	276.24	3.9
Folkman	20	40	5	5 (-11)	8,280	168,190	11.67	0.36
Franklin	12	18	4	6 (-3)	1,848	21,798	0.07	0.04
Frucht	12	18	5	4 (-4)	1,800	20,223	0.39	0.02
Hoffman	16	32	6	7 (-4)	4,160	64,968	8.95	0.46
Kittell	23	63	8	8 (-8)	12,006	281,310	179.62	18.65
McGee	24	36	8	7 (-10)	13,680	303,660	8,700.94	59.89
Paley-13	13	39	9	4 (-1)	1,820	22,776	12.73	0.05
Paley-17	17	68	11	6 (-1)	3,978	72,896	194.38	0.12
Pappus	18	27	8	5 (-6)	5,616	90,315	983.67	0.14
Petersen	10	15	5	5 (-1)	1,040	9,550	0.10	0.02
Poussin	15	39	7	5 (-4)	3,300	50,145	9.00	0.21
Robertson	19	38	9	7 (-4)	6,422	112, 461	478.83	0.76
Shrikhande	16	48	9	5 (-3)	3,680	59,688	129.75	0.11
Sousselier	16	27	6	11 (0)	4,160	63,564	3.67	11.75

The fifth column of Table III ("length") shows the length of shortest k-derivations for graphs of clique-width k; the value in parenthesis indicates the difference between this length and the upper bound provided by Lemma 3.7. We notice that this difference is substantial for most of the considered graphs but only 0 for the Sousselier Graph. The numbers in the following columns (i.e., the runtimes and the number of variables and clauses) are based on the upper bound provided by Lemma 3.7. In the appendix we provide k-derivations and linear k-derivations of minimal length for the named graphs.

We have also computed the linear clique-width of the named graphs of Table III. It turned out that for most graphs clique-width and linear clique-width coincide or are very close, see Table IV. The McGee graph is the only one among the considered graphs where the gap between clique-width and linear clique-width is 2. For five of the graphs, the gap is 1, and for all remaining graphs, clique-width and linear clique-width are the same. Computing linear clique-width turns out to be somewhat less expensive compared computing clique-width due to the additional clauses. However, the performance improvement is smaller for graphs whose linear clique-width is larger than the clique-width.

Table IV. Difference between clique-width and linear clique-width of named graphs.

0		1	2
Brinkmann	Paley-13	Chvátal	McGee
Clebsch	Paley-17	Flower Snark	
Desargues	Pappus	Folkman	
Dodecahedron	Poussin	Franklin	
Errera	Robertson	Hoffman	
Frucht	Shrikhande	Petersen	
Kittell	Sousselier		

8. CONCLUSION

We have presented a SAT approach to the exact computation of clique-width and linear clique-width, based on a reformulation of clique-width and several techniques to speed up the search. This new approach allowed us to systematically compute the exact clique-width and linear clique-width of various small graphs. We think that our results could be of relevance for theoretical investigations. For instance, knowing small vertex-minimal graphs of certain clique-width could be helpful for the design of discrete algorithms that recognize graphs of bounded clique-width. Such graphs can also be useful as gadgets for a reduction to show that the recognition of graphs of clique-width 4 is NP-hard, which is still a long-standing open problem [Fellows et al. 2009]; the question is also open for linear clique-width [Heggernes et al. 2012]. Furthermore, as discussed in Section 1, there are no heuristic algorithms to compute the clique-width directly, but heuristic algorithms for related parameters can be used to obtain upper bounds on the clique-width. Our SAT-based approach can be used to empirically evaluate how far heuristics are from the optimum, at least for small and medium-sized graphs.

So far we have focused in our experiments on the exact (linear) clique-width, but for various applications it is sufficient to have good upper bounds. Our results (see Table I) suggest that our approach can be scaled to medium-sized graphs for the computation of upper bounds. We also observed that for many graphs the upper bound of Lemma 3.7 is not tight. Thus, we expect that searching for shorter derivations, which is significantly faster, will yield optimal or close to optimal solutions in many cases.

Finally, we would like to mention that our SAT-based approach is very flexible and open. We think that similar to the adaption to linear clique-width, it can also be adapted to other variants of clique-width, such as m-clique-width [Courcelle and Twigg 2010] and NLC-width [Wanke 1994]. Our approach could then be used for an empirical comparison of these parameters.

Acknowledgement

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing grid resources that have contributed to the research results reported within this paper.

A. APPENDIX: K-DERIVATIONS OF NAMED GRAPHS

In this appendix we provide (linear) derivations for the named graphs of Table III. We will use the following *string notation*, which avoids the redundancy that comes along with the explicit statement of each template. Let $\mathcal{D} = (T_0, \dots, T_t)$ be a derivation over a universe U. For two elements $a, a' \in U$ let $c(a, a')$ be the smallest $i \in \{1, \dots, t\}$ such that a, a' belong to the same component in T_i. Similarly, if a, a' belong to the same group in T_t, let $c(a, a')$ be the smallest $i \in \{1, \dots, t\}$ such that a, a' belong to the same group in T_i; otherwise, if a, a' do not belong to the same group of T_t, then we let $g(a, a') = t + 1$. Let $a_1, \dots, a_n$ be the elements of U in such an order that for any three elements a_i, a_{i+1}, a_{i+2} we have $c(a_i, a_{i+1}), c(a_{i+1}, a_{i+2}) \leq c(a_i, a_{i+2})$. It is easy to see that such an order always exists, although it may not be unique. Now we can represent the sequence $\text{cmp}(T_1), \dots, \text{cmp}(T_t)$ by the *component string*

$$a_1^{c(a_1,a_2)} a_2^{c(a_2,a_3)} a_3^{c(a_3,a_4)} \dots a_{n-1}^{c(a_{n-1},a_n)} a_n$$

For instance, the component string for the derivation of Example 3.1 is $a^1 b^2 c^3 d$.

For the groups we proceed similarly. Let $b_1, \dots, b_n$ be the elements of U in such an order that for any three elements b_i, b_{i+1}, b_{i+2} we have $g(b_i, b_{i+1}), g(b_{i+1}, b_{i+2}) \leq g(b_i, b_{i+2})$.

Now we represent the sequence grp$(T_1), \ldots, grp(T_t)$ by the *group string*

$$b_1^{g(b_1,b_2)} b_2^{g(b_2,b_3)} b_3^{g(b_3,b_4)} \ldots b_{n-1}^{g(b_{n-1},b_n)} b_n$$

The corresponding group string for the derivation of Example 3.1 is $a^3b^4c^4d$. By joining the component string and the group string with a semicolon we get a concise representation of a derivation: $a^1b^2c^3d : a^3b^4c^4d$. It is straightforward to reconstruct the derivation from the string.

As an example, consider the Petersen graph as depicted in Figure 6, with

$$\begin{aligned} V &= \{a, b, c, d, e, f, g, h, i, j\}, \\ E &= \{ac, ad, af, bd, be, bg, ce, ch, di, ej, fg, fj, gh, hi, ij\}. \end{aligned}$$

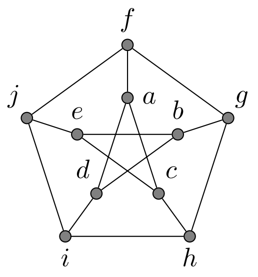

Fig. 6. The Petersen graph

With our SAT approach we found the 5-derivation $(T_0, \ldots, T_4)$ with

$$\begin{aligned} \textbf{cmp}(T_0) &= \{a, b, c, d, e, f, g, h, i, j\}, & \textbf{grp}(T_0) &= \{a, b, c, d, e, f, g, h, i, j\}, \\ \textbf{cmp}(T_1) &= \{ace, bfg, dhi, i\}, & \textbf{grp}(T_1) &= \{a, b, c, d, e, f, g, h, i, j\}, \\ \textbf{cmp}(T_2) &= \{abcefg, dhi, j\}, & \textbf{grp}(T_2) &= \{a, b, cg, d, e, f, h, i, j\}, \\ \textbf{cmp}(T_3) &= \{abcdefghi, j\}, & \textbf{grp}(T_3) &= \{ab, cg, d, efi, h, j\}, \\ \textbf{cmp}(T_4) &= \{abcdefghij\}, & \textbf{grp}(T_4) &= \{ab, cg, dh, efi, j\}; \end{aligned}$$

we omit curly brackets and commas when we state components and groups.

This 5-derivation is represented by the string

$$a^1c^1e^2b^1f^1g^3d^1h^1i^4j : a^3b^5c^2g^5d^4h^5e^3f^3i^5j.$$

The construction given in the proof of Lemma 3.6, after simplification to save some relabelings, yields the 5-expression

$$\eta_{3,5}(ABC \oplus 5(j))$$

where

$$\begin{aligned} ABC &= \rho_{5\to 6}(\eta_{1,4}(\eta_{2,3}(AB \oplus C))), \\ AB &= \rho_{4\to 1}(\rho_{5\to 3}(\eta_{1,5}(\eta_{3,4}(A \oplus B)))), \\ A &= \eta_{2,3}(\eta_{1,2}(1(a) \oplus 2(c) \oplus 3(e))), \\ B &= \eta_{2,5}(\eta_{4,5}(4(b) \oplus 5(f) \oplus 2(g))), \\ C &= \eta_{3,4}(\eta_{3,5}(4(d) \oplus 5(h) \oplus 3(i))). \end{aligned}$$

Next we we provide certificates for tight upper bounds for the clique-width and the linear clique-width of the famous named graphs mentioned in Section 7.3. We give the certificates in terms of k-derivations and linear k-derivations, stated in string notation. We state (linear) k-derivations of minimal length, thus providing certificates for the derivation lengths stated in Table III. Shortest k-derivations are often not linear, hence we provide both, a linear k-derivation and a k-derivation, even for graphs whose clique-width and linear clique-width coincide.

Brinkmann Graph

$$\begin{aligned}
V &= \{a, \ldots, u\} \\
E &= \{ad, ae, aj, as, bc, bd, bh, bt, ce, cl, cu, dg, di, ef, \\
&\quad ek, fh, fo, fp, gl, gm, gp, hj, hm, ik, in, io, jn, ju, \\
&\quad km, kq, lo, ls, mr, np, nt, or, pq, qs, qu, rt, ru, st\} \\
\text{10-der} &= a^1b^1c^1e^1k^1u^2d^1g^1m^1n^1p^1q^3j^4h^5s^5t^6f^5i^5o^5r^6l : \\
&\quad a^4q^6b^6j^7c^2g^6o^7d^3k^6n^7e^2p^5h^7f^7i^7l^7m^5u^6t^7r^7s \\
\text{lin 10-der} &= a^3d^1e^1f^1g^1h^1i^1k^1l^1m^1o^2p^2q^3j^4c^5b^5n^6r^6s^6t^6u : \\
&\quad a^5l^7b^6n^7c^6j^7d^4h^6e^5f^2g^3k^6i^3p^7m^2o^7q^7r^7s^7t^7u
\end{aligned}$$

Chvátal Graph

$$\begin{aligned}
V &= \{a, \ldots, l\} \\
E &= \{ag, aj, ak, al, bc, bf, bh, bk, cg, ci, cl, dh, \\
&\quad dj, dk, dl, ef, ei, ek, el, fg, fj, gh, hi, ij\} \\
\text{5-der} &= a^1b^1f^1g^1k^2c^1d^1h^1i^1l^3j^4e : a^2d^4b^2g^3c^2h^5e^5f^2i^5j^5k^2l \\
\text{lin 6-der} &= a^1b^1c^1g^1k^1l^2e^2f^3d^3i^4h^4j : a^3f^5b^2g^5c^3e^4k^2l^5d^4i^5h^5j
\end{aligned}$$

Clebsch Graph

$$\begin{aligned}
V &= \{a, \ldots, p\} \\
E &= \{ac, ae, ag, aj, am, be, bf, bh, bj, bn, cd, cf, cn, co, \\
&\quad de, dh, di, dp, ek, el, fg, fi, fl, gh, gk, gp, hm, ho, \\
&\quad ij, ik, im, jo, jp, kn, ko, lm, lo, lp, mn, np\} \\
\text{8-der} &= a^1d^1g^1h^1i^1j^1m^1p^2b^1c^1e^1f^1k^1l^1n^1o : \\
&\quad a^2d^3b^2o^3c^2e^3f^2k^3g^2i^3h^2j^3l^2n^3m^2p, \\
\text{lin 8-der} &= a^1c^1i^1j^1k^1m^1n^1o^2b^2f^2g^2h^3d^3e^3l^3p : \\
&\quad a^2k^3b^4c^2i^3h^4d^4e^4f^3m^2o^4g^3j^2n^4l^4p
\end{aligned}$$

Desargues Graph

$$\begin{aligned}
V &= \{a, \ldots, t\} \\
E &= \{ab, af, at, bc, bq, cd, cl, de, do, ef, ej, fg, gh, gp, hi, \\
&\quad hs, ij, in, jk, kl, kt, lm, mn, mr, no, op, pq, qr, rs, st\}
\end{aligned}$$

$$\begin{array}{rl} \text{8-der} & a^1f^1g^1p^1q^1r^1s^1t^2c^1d^1i^1j^1k^1l^1n^1o^3b^3h^4e^4m : \\ & a^2c^2q^4g^2i^2s^5b^4k^3o^3p^4t^5d^2f^2j^5e^5h^5l^2n^2r^5m \end{array}$$

$$\begin{array}{rl} \text{lin 8-der} & a^1c^1d^1f^1l^1m^1s^1t^2b^2q^3r^4p^5o^6g^7e^7k^8h^8i^8j^8n : \\ & a^2c^3b^4r^5q^7p^8d^8f^8l^2t^9e^8k^9g^7s^9h^9i^9j^9m^6o^9n \end{array}$$

Dodecahedral Graph (Dodecahedron)

$$\begin{aligned} V &= \{a, \dots, t\} \\ E &= \{an, ao, ap, be, bf, bm, cg, cn, cs, dh, do, dt, ek, es, \\ &\quad fl, ft, gk, gp, hl, hp, ij, in, iq, jo, jr, kl, mq, mr, qs, rt\} \\ \text{8-der} &= a^1e^1h^1j^1k^1l^1o^1p^2c^1g^1s^3n^4i^5b^4m^4q^6d^2f^5r^3t : \\ &\quad a^2c^4g^3k^2p^5n^6e^7b^6l^7d^7f^7h^2o^6t^7i^5s^6q^7j^5m^7r \\ \text{lin 8-der} &= a^3b^1d^1e^1f^1l^1m^1q^1t^2j^2r^3o^4k^5i^6h^6p^7c^7g^7n^7s : \\ &\quad a^7i^8b^2f^3m^2t^3r^4o^6j^7d^5l^7h^8c^8e^6q^8g^8k^7p^8n^8s \end{aligned}$$

Errera Graph

$$\begin{aligned} V &= \{a, \dots, q\} \\ E &= \{ac, ae, aj, al, aq, bh, bi, bm, bn, bo, ce, cj, ck, co, df \\ &\quad dh, dk, dm, dp, ek, el, ep, fg, fm, fn, fp, gi, gl, gn, \\ &\quad gp, gq, hk, hm, ho, ij, in, io, iq, jo, jq, ko, kp, lp, lq, mn\} \\ \text{8-der} &= a^1c^1e^1j^1k^1l^1q^2b^1d^1f^1h^1m^1n^2p^3g^3i^2o : \\ &\quad a^2m^3d^2e^4b^2j^4c^2h^3k^4f^2l^3p^4g^4i^4n^2q^4o \\ \text{lin 8-der} &= a^5b^1d^1f^1i^1k^1n^1o^1p^2h^2m^3e^3j^4l^6c^6g^6q : \\ &\quad a^6j^7b^2d^3h^3m^7c^7e^6k^4o^7f^2n^5p^7g^7i^6l^7q \end{aligned}$$

Flower Snark

$$\begin{aligned} V &= \{a, \dots, t\} \\ E &= \{ab, ae, aq, bc, bd, ch, ct, dg, ds, ef, ei, fg, fh, gl, hk, \\ &\quad ij, im, jk, jl, kp, lo, mn, mq, no, np, ot, ps, qr, rs, rt\} \\ \text{7-der} &= a^1e^1i^1j^1q^2m^2n^3c^1h^1k^1o^1t^2d^1g^1l^1p^1s^4b^4f^4r : \\ &\quad a^3c^2d^5b^5e^3g^2h^5f^5i^3k^3l^4j^4m^3o^3p^4n^5q^3s^2t^5r \\ \text{lin 8-der} &= a^6b^6c^3d^3f^2g^1h^1k^1l^1o^1p^1s^1t^2r^4i^4j^5m^5n^6e^6q : \\ &\quad a^7b^7c^4d^7e^7f^6i^7g^3s^4h^3t^5j^5k^2l^6n^6o^2p^7m^6r^7q \end{aligned}$$

Folkman Graph

$$\begin{aligned} V &= \{a, \dots, t\} \\ E &= \{ab, ad, aj, al, bc, bm, bs, cd, cf, ch, dm, ds, ef, eh, en, ep, fg, fq, gh, \\ &\quad gj, gl, hq, ij, il, ir, it, jk, kl, kn, kp, mn, mp, no, op, or, ot, qr, qt, rs, st\} \end{aligned}$$

$$\begin{aligned}
\text{5-der} &= a^1i^1j^1k^1l^2c^1e^1f^1g^1h^1q^3b^2d^1s^2m^4n^3o^1p^4r^3t : \\
&\quad a^2c^4b^3d^4f^1h^2j^1l^3g^5e^2k^3m^5i^2q^3s^4o^5n^3p^5r^3t \\
\text{lin 6-der} &= a^5b^5c^2f^1g^1h^1k^1n^1o^1p^1q^2m^3e^3r^3t^4i^4s^5d^5j^5l : \\
&\quad a^6b^5d^6c^3m^5s^6e^4f^1h^3n^1p^4o^2q^5r^3t^6g^2k^5i^6j^5l
\end{aligned}$$

Franklin Graph

$$\begin{aligned}
V &= \{a, \ldots, l\} \\
E &= \{ab, ah, al, bc, bk, cd, cf, de, di, \\
&\quad ef, el, fg, gh, gj, hi, ij, jk, kl\} \\
\text{4-der} &= a^1h^2b^2j^1k^3d^1i^2e^2f^1g^4c^5l : a^3e^3k^6b^3d^2f^6c^5g^2i^4h^2j^6l \\
\text{lin 5-der} &= a^1g^1h^1j^1k^2d^2i^3f^4b^4e^5c^5l : a^2k^5e^6b^5d^4f^6c^6g^4h^2j^3i^6l
\end{aligned}$$

Frucht Graph

$$\begin{aligned}
V &= \{a, \ldots, l\} \\
E &= \{ab, ag, ak, bc, bh, cd, ci, de, di, \\
&\quad ef, ej, fg, fj, gk, hk, hl, il, jl\} \\
\text{5-der} &= a^1b^1g^1h^1k^2d^1e^1i^1j^1l^3c^3f : a^2k^3h^3l^4b^2d^2i^4c^4e^2g^2j^4f \\
\text{lin 5-der} &= a^5b^3c^2d^1e^1i^1j^1l^2h^4k^6f^6g : a^6k^7b^6c^4d^2i^3l^5h^7e^2j^7f^7g
\end{aligned}$$

Hoffman Graph

$$\begin{aligned}
V &= \{a, \ldots, p\} \\
E &= \{ai, aj, ak, al, bi, bj, bk, bm, ci, cl, cn, co, dj, dl, dn, dp, \\
&\quad ek, el, eo, ep, fi, fm, fn, fo, gj, gm, gn, gp, hk, hm, ho, hp\} \\
\text{6-der} &= a^1c^1d^1e^1l^2b^1f^1g^1h^1m^2p^3o^4k^5n^6i^5j : \\
&\quad a^2b^7c^2f^7d^2g^7e^2h^5k^6l^2m^3p^4o^6n^7i^7j \\
\text{lin 7-der} &= a^4c^1d^1f^1g^1i^1j^2o^2p^3l^3m^3n^4h^5b^5e^5k : \\
&\quad a^5h^6b^6c^3d^4f^3g^4n^6e^6i^2j^5m^6k^6l^5o^3p
\end{aligned}$$

Kittell Graph

$$\begin{aligned}
V &= \{a, \ldots, w\} \\
E &= \{af, ak, am, aq, aw, bc, bd, bk, br, bt, cl, co, cp, cr, ct, dh, dk, dn, dt, eg, ej, eq, \\
&\quad eu, ew, fk, fl, fm, fn, fs, fv, gp, gq, gr, gu, hn, ho, ht, hv, ij, il, ip, is, iu, \\
&\quad jm, js, ju, jw, kn, kq, kr, lo, lp, ls, lv, ms, mw, nv, ot, ov, pr, pu, qr, qw\} \\
\text{8-der} &= a^7b^1c^1f^1h^1n^1o^1t^1v^2d^1r^3l^3p^4k^5e^3j^3u^4m^3w^6g^6q^7i^5s : \\
&\quad a^8b^3d^3n^5c^4h^2t^4o^2v^7e^5r^7g^8f^6m^8i^8j^5l^8k^5w^7q^8p^6u^8s \\
\text{lin 8-der} &= a^1e^1i^1l^1m^1s^1u^1w^2g^3j^3q^4p^5r^6k^7b^7c^8d^8n^9h^{10}v^{11}f^{11}o^{11}t :
\end{aligned}$$

$$a^7m^2s^9k^{11}n^{12}b^{10}d^{12}c^{11}h^{12}e^3w^4j^5i^3u^6g^7q^8p^8r^{12}f^{12}l^{11}v^{12}o^{12}t$$

McGee Graph

$$\begin{aligned}
V &= \{a,\dots,x\}\\
E &= \{ab, ah, ax, bc, bs, cd, co, de, dk, ef, ev, fg, fr, gh, gn, hi, ij, iu,\\
&\quad jk, jq, kl, lm, lx, mn, mt, no, op, pq, pw, qr, rs, st, tu, uv, vw, wx\}\\
\text{8-der} &= a^1b^1c^1o^1p^1w^1x^2i^1j^1k^1l^1m^1t^1u^3q^4s^5d^3f^1h^4e^4g^6n^6r^6v :\\
&\quad a^2i^6b^2t^5j^2p^4l^3x^6c^2k^6d^7e^5u^2w^7f^5q^5s^7g^5m^2o^7h^7n^7r^7v\\
\text{lin 10-der} &= a^1b^1e^1f^1g^1m^1n^1t^1w^1x^2s^2v^3c^3d^4k^4l^5h^5i^5u^6j^6o^6p^6q^6r :\\
&\quad a^2g^6b^4e^5d^5l^5m^2x^6h^6t^3v^6u^7c^4n^7f^3s^7i^6k^7j^7o^7p^7q^7r^7w
\end{aligned}$$

Paley Graph with 13 vertices (Paley-13)

$$\begin{aligned}
V &= \{a,\dots,m\}\\
E &= \{ab, ad, ae, aj, ak, am, bc, be, bf, bk, bl, cd, cf, cg, cl, cm, de, dg, dh, dm,\\
&\quad ef, eh, ei, fg, fi, fj, gh, gj, gk, hi, hk, hl, ij, il, im, jk, jm, kl, lm\}\\
\text{9-der} &= a^1d^1e^1f^1h^1j^1k^1l^1m^2g^3b^1c^3i : a^3k^4b^4c^4d^3g^4e^4f^3l^4h^2j^4i^4m\\
\text{lin 9-der} &= a^2b^1c^1d^1e^1g^1h^1i^1j^2f^3k^3l^3m : a^3j^4b^3h^4c^2i^4d^4e^3f^4g^4k^4l^4m
\end{aligned}$$

Paley Graph with 17 vertices (Paley-17)

$$\begin{aligned}
V &= \{a,\dots,q\}\\
E &= \{ab, ac, ae, ai, aj, an, ap, aq, bc, bd, bf, bj, bk, bo, bq, cd, ce, cg\\
&\quad ck, cl, cp, de, df, dh, dl, dm, dq, ef, eg, ei, em, en, fg, fh, fj,\\
&\quad fn, fo, gh, gi, gk, go, gp, hi, hj, hl, hp, hq, ij, ik, im, iq, jk,\\
&\quad jl, jn, kl, km, ko, lm, ln, lp, mn, mo, mq, no, np, op, oq, pq\}\\
\text{11-der} &= a^3b^1c^1e^1f^1g^1j^1k^1l^1o^1p^1q^2m^4d^5h^3i^5n :\\
&\quad a^5e^4m^6b^2c^6d^6f^3l^5p^6g^5q^6h^6i^6j^6k^6n^6o\\
\text{lin 11-der} &= a^1c^1d^1e^1g^1h^1i^1k^1m^1n^1p^2o^3j^4b^5f^5l^5q :\\
&\quad a^5i^6b^5o^6c^4k^6d^6e^3g^6f^6h^6j^5n^6l^6m^2p^6q
\end{aligned}$$

Pappus Graph

$$\begin{aligned}
V &= \{a,\dots,r\}\\
E &= \{ad, ao, ap, be, bp, bq, cf, cg, cr, dg, dh, ef, eh,\\
&\quad fi, gj, hk, il, im, jm, jn, kl, kn, lo, mp, nq, or, qr\}\\
\text{8-der} &= a^1b^1e^1h^1o^1p^2c^1g^1j^1k^1n^1r^3i^1l^2m^4d^4f^4q :\\
&\quad a^2g^3h^5b^2n^4r^5c^2e^3i^5d^5f^5j^2p^4l^4m^5k^3o^5q\\
\text{lin 8-der} &= a^1b^1i^1j^1l^1n^1o^1q^2d^2e^3m^4h^5k^5p^6c^6f^6g^6r :
\end{aligned}$$

$$a^3b^4m^6h^5l^2n^6k^6p^7c^7d^5j^7e^6i^7f^7g^7o^2q^7r$$

Petersen Graph

$$\begin{aligned}
V &= \{a,\dots,j\}\\
E &= \{ac, ad, af, bd, be, bg, ce, ch, di, ej, fg, fj, gh, hi, ij\}\\
\textbf{5-der} &= a^1c^1e^2b^1f^1g^3d^1i^2h^4j : a^3b^5c^2g^5d^4h^5e^3f^3i^5j\\
\textbf{lin 6-der} &= a^1d^1f^1g^1h^1i^2b^2c^2e^3j : a^2h^3b^3c^4d^2g^4e^4f^2i^4j
\end{aligned}$$

Poussin Graph

$$\begin{aligned}
V &= \{a,\dots,o\}\\
E &= \{ac, af, ak, al, am, bc, bf, bi, bj, bn, cf, ci, cl, dg, dh, dj, dm, dn, do, eg,\\
&\quad ek, el, eo, fj, fm, gi, gl, gn, go, hk, hm, ho, il, in, jm, jn, kl, km, ko\}\\
\textbf{7-der} &= a^3c^1e^1g^1i^1l^2d^1f^1h^1j^1m^3o^4b^2n^4k : a^4e^2h^4l^2m^4o^5b^5c^3f^5d^3g^5i^2j^5k^5n\\
\textbf{lin 7-der} &= a^2c^1d^1f^1i^1j^1l^1n^3m^4h^5o^6b^7e^7g^7k : a^5m^6h^8b^7c^4f^3j^8d^7i^2n^8e^8g^8k^8l^7o
\end{aligned}$$

Robertson Graph

$$\begin{aligned}
V &= \{a,\dots,s\}\\
E &= \{ab, ae, ap, as, bc, bi, bm, cd, cg, cr, de, dl, do, ef, ej, fg, fm, fq, gh,\\
&\quad gk, hi, ho, hs, ij, iq, jk, jn, kl, kp, lm, ls, mn, no, nr, op, pq, qr, rs\}\\
\textbf{9-der} &= a^5l^3r^2s^6b^1c^1f^1g^1i^1j^1k^1m^1q^2n^3p^4o^5d^1e^6h :\\
&\quad a^7b^5p^6e^7c^6n^5q^7d^6k^4m^7f^3j^7g^2i^6o^6s^7h^7l^7r\\
\textbf{lin 9-der} &\quad a^1b^1c^1d^1e^1f^1g^1i^1q^2h^3p^4o^5m^5s^6l^7j^7k^7n^7r :\\
&\quad a^5h^6b^2f^7d^8c^4q^7s^8e^3i^8g^5p^7l^8j^8k^8m^7o^8n^8r
\end{aligned}$$

Shrikhande Graph

$$\begin{aligned}
V &= \{a,\dots,p\}\\
E &= \{ab, ac, ad, ae, af, ag, bc, bg, bh, bi, bj, cd, ch, ck, cl, df,\\
&\quad dk, dm, dn, ef, eg, el, eo, ep, fi, fm, fo, gj, gn, gp, hi, hl,\\
&\quad hm, hp, ij, im, io, jk, jn, jo, kl, kn, ko, lo, lp, mn, mp, np\}\\
\textbf{9-der} &= a^1b^1c^1d^1g^1i^1j^1k^1l^2h^2n^3o^4e^3f^3m^1p :\\
&\quad a^4o^5b^2c^4j^2k^5d^4i^5e^5f^5g^4l^5h^3n^5m^5p\\
\textbf{lin 9-der} &= a^1b^1d^1e^1f^1h^1i^1j^2g^2m^3c^3o^4k^4l^4n^4p :\\
&\quad a^3b^4f^2i^5c^4o^5d^4j^5e^4h^5g^3m^5k^5l^5n^5p
\end{aligned}$$

Sousselier Graph

$$\begin{aligned}
V &= \{a, \dots, p\} \\
E &= \{ab, ao, ap, bc, bf, bl, cd, cn, de, dp, ef, ei, eo, \\
&\quad fg, gh, gp, hi, hl, ij, jk, jp, kl, ko, lm, mn, mp, no\} \\
\text{6-der} &= a^4 b^3 f^1 g^1 h^2 i^1 j^1 k^2 o^5 n^6 c^7 e^8 p^9 d^{10} l^3 m : \\
&\quad a^5 g^2 j^9 f^4 i^6 o^{10} c^8 e^{10} d^{11} b^7 h^3 k^{11} l^{11} m^{11} n^{11} p \\
\text{lin 6-der} &= a^4 b^1 f^1 g^1 h^1 i^1 j^2 k^3 o^5 c^6 n^7 e^8 p^9 m^{10} d^{10} l : \\
&\quad a^5 g^3 j^9 f^2 i^7 o^{10} n^{11} b^6 h^4 k^{11} c^8 e^{10} p^{11} d^{11} l^{11} m
\end{aligned}$$

REFERENCES

Isolde Adler and Mamadou Moustapha Kanté. 2013. Linear Rank-Width and Linear Clique-Width of Trees. In *Graph-Theoretic Concepts in Computer Science - 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers (Lecture Notes in Computer Science)*, Andreas Brandstädt, Klaus Jansen, and Rüdiger Reischuk (Eds.), Vol. 8165. Springer Verlag, 12–25.

Gilles Audemard and Laurent Simon. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 399–404. http://dl.acm.org/citation.cfm?id=1661445.1661509

Martin Beyß. 2013. Fast Algorithm for Rank-Width. In *Mathematical and Engineering Methods in Computer Science, 8th International Doctoral Workshop, MEMICS 2012, Znojmo, Czech Republic, October 25-28, 2012, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 7721. Springer Verlag, 82–93.

Armin Biere. 2012. Lingeling and friends entering the SAT Challenge 2012. In *Solver and Benchmark Descriptions (Department of Computer Science Series of Publications B.)*, A. Balint, A. Belov, A. Diepold, S. Gerber, M. Järvisalo, and C. Sinz (Eds.), Vol. B-2012-2. University of Helsinki, 33–34.

Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press.

Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. 2011. Boolean-width of graphs. *Theoretical Computer Science* 412, 39 (2011), 5187–5204.

Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. 2012. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discr. Appl. Math.* 160, 6 (2012), 834–865.

Derek G. Corneil and Udi Rotics. 2005. On the relationship between clique-width and treewidth. *SIAM J. Comput.* 34, 4 (2005), 825–847.

Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. 1991. Context-free Handle-rewriting Hypergraph Grammars. In *Graph-Grammars and their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5–9, 1990, Proceedings (Lecture Notes in Computer Science)*, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg (Eds.), Vol. 532. 253–268.

Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. 1993. Handle-rewriting hypergraph grammars. *J. of Computer and System Sciences* 46, 2 (1993), 218–270.

Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. 2000. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.* 33, 2 (2000), 125–150.

Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. 2001. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.* 108, 1-2 (2001), 23–52.

Bruno Courcelle and Stephan Olariu. 2000. Upper bounds to the clique-width of graphs. *Discr. Appl. Math.* 101, 1-3 (2000), 77–114.

Bruno Courcelle and Andrew Twigg. 2010. Constrained-path labellings on graphs of bounded clique-width. *Theory Comput. Syst.* 47, 2 (2010), 531–567.

Reinhard Diestel. 2000. *Graph Theory* (2nd ed.). Graduate Texts in Mathematics, Vol. 173. Springer Verlag, New York.

P. Alex Dow and Richard E. Korf. 2007. Best-First Search for Treewidth. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 1146–1151.

Iréne Durand and Bruno Courcelle. 2013. Infinite Transducers on Terms Denoting Graphs. In *Proceedings of ELS 2013, the 6th European Lisp Symposium June 3–4, 2013, Madrid, Spain*. 47–58. http://www.nicklevine.org/els2013/proceedings.pdf.

Niklas Eén and Niklas Sörensson. 2004. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, Enrico Giunchiglia and Armando Tacchella (Eds.). Lecture Notes in Computer Science, Vol. 2919. Springer Verlag, 502–518.

Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. 2009. Clique-width is NP-complete. *SIAM J. Discrete Math.* 23, 2 (2009), 909–939.

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. 2007. *clasp* : A Conflict-Driven Answer Set Solver. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings (Lecture Notes in Computer Science)*, Chitta Baral, Gerhard Brewka, and John S. Schlipf (Eds.), Vol. 4483. Springer Verlag, 260–265.

Ian P. Gent. 2002. Arc Consistency in SAT. In *15th European Conference on Artificial Intelligence (ECAI 2002)*, F. van Harmelen (Ed.). IOS Press, 121–125.

Vibhav Gogate and Rina Dechter. 2004. A Complete Anytime Algorithm for Treewidth. In *Proceedings of the Proceedings of the Twentieth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*. AUAI Press, Arlington, Virginia, 201–208.

Martin Charles Golumbic and Udi Rotics. 2000. On the clique-width of some perfect graph classes. *Internat. J. Found. Comput. Sci.* 11, 3 (2000), 423–443. Selected papers from the Workshop on Graph-Theoretical Aspects of Computer Science (WG 99), Part 1 (Ascona).

Carla P. Gomes, Henry Kautz, Ashish Sabharwal, and Bart Selman. 2008. Satisfiability Solvers. In *Handbook of Knowledge Representation*. Foundations of Artificial Intelligence, Vol. 3. Elsevier, 89–134.

Frank Gurski and Egon Wanke. 2005. On the relationship between NLC-width and linear NLC-width. *Theoretical Computer Science* 347, 1-2 (2005), 76–89.

Michel Habib and Christophe Paul. 2010. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review* 4, 1 (2010), 41–59.

Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. 2011. Graphs of linear clique-width at most 3. *Theoretical Computer Science* 412, 39 (2011), 5466–5486.

Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. 2012. Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs. *Discr. Appl. Math.* 160, 6 (2012), 888–901.

Pinar Heggernes, Daniel Meister, and Udi Rotics. 2011. Computing the Clique-Width of Large Path Powers in Linear Time via a New Characterisation of Clique-Width. In *Computer Science - Theory and Applications - 6th International Computer Science Symposium in Russia, CSR 2011, St. Petersburg, Russia, June 14-18, 2011. Proceedings (Lecture Notes in Computer Science)*, Alexander S. Kulikov and Nikolay K. Vereshchagin (Eds.), Vol. 6651. Springer Verlag, 233–246.

Eivind Magnus Hvidevold, Sadia Sharmin, Jan Arne Telle, and Martin Vatshelle. 2012. Finding Good Decompositions for Dynamic Programming on Dense Graphs. In *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers (Lecture Notes in Computer Science)*, Dániel Marx and Peter Rossmanith (Eds.), Vol. 7112. Springer Verlag, 219–231.

Hadi Katebi, Karem A. Sakallah, and Igor L. Markov. 2012. Conflict Anticipation in the Search for Graph Automorphisms. In *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings (Lecture Notes in Computer Science)*, Nikolaj Bjørner and Andrei Voronkov (Eds.), Vol. 7180. Springer Verlag, 243–257.

Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. 2001. Treewidth: Computational Experiments. *Electronic Notes in Discrete Mathematics* 8 (2001), 54–57.

Choongbum Lee, Joonkyung Lee, and Sang-il Oum. 2012. Rank-width of random graphs. *J. Graph Theory* 70, 3 (2012), 339–347.

Vadim Lozin and Dieter Rautenbach. 2007. The relative clique-width of a graph. *J. Combin. Theory Ser. B* 97, 5 (2007), 846–858.

Sharad Malik and Lintao Zhang. 2009. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM* 52, 8 (2009), 76–82.

Brendan D. McKay. 1981. Practical graph isomorphism, In Proceedings of the Tenth Manitoba Conference on Numerical Mathematics and Computing, Vol. I (Winnipeg, Man., 1980). *Congr. Numer.* 30 (1981), 45–87.

Sang-il Oum and P. Seymour. 2006. Approximating Clique-width and Branch-width. *J. Combin. Theory Ser. B* 96, 4 (2006), 514–528.

Sang-Il Oum. 2008. Approximating Rank-width and Clique-width Quickly. *ACM Trans. Algorithms* 5, 1, Article 10 (Dec. 2008), 20 pages. DOI:http://dx.doi.org/10.1145/1435375.1435385

Karem A. Sakallah and João Marques-Silva. 2011. Anatomy and Empirical Evaluation of Modern SAT Solvers. 103 (2011), 96–121.

Marko Samer and Helmut Veith. 2009. Encoding Treewidth into SAT. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings (Lecture Notes in Computer Science)*, Vol. 5584. Springer Verlag, 45–50.

Carsten Sinz. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings (Lecture Notes in Computer Science)*, Peter van Beek (Ed.), Vol. 3709. Springer Verlag, 827–831.

J. Cole Smith, Elif Ulusal, and Illya V. Hicks. 2012. A combinatorial optimization algorithm for solving the branchwidth problem. *Comput. Optim. Appl.* 51, 3 (2012), 1211–1229.

Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. 2009. Compiling finite linear CSP into SAT. *Constraints* 14, 2 (2009), 254–272.

Magnus Wahlström. 2011. New plain-exponential time classes for graph homomorphism. *Theory Comput. Syst.* 49, 2 (2011).

Toby Walsh. 2000. SAT v CSP. In *6th International Conferenc on Principles and Practice of Constraint Programming (CP 2000) (Lecture Notes in Computer Science)*, R. Dechter (Ed.), Vol. 1894. Springer Verlag, 441–456.

Egon Wanke. 1994. k-NLC graphs and polynomial algorithms. *Discr. Appl. Math.* 54, 2-3 (1994), 251–266. Efficient algorithms and partial k-trees.

Eric Weisstein. 2015. MathWorld online Mathematics resource. (2015). Last accessed on February 13, 2015.

Hantao Zhang. 2009. Combinatorial Designs by SAT Solvers. In *Handbook of Satisfiability*, Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 533–568.

Received January 2014; revised ; accepted February 2015