Multi-Clique-Width[*]

Martin Fürer[†]

The Pennsylvania State University, University Park, USA
furer@cse.psu.edu

Abstract

Multi-clique-width is obtained by a simple modification in the definition of clique-width. It has the advantage of providing a natural extension of tree-width. Unlike clique-width, it does not explode exponentially compared to tree-width. Efficient algorithms based on multi-clique-width are still possible for interesting tasks like computing the independent set polynomial or testing c-colorability. In particular, c-colorability can be tested in time linear in n and singly exponential in c and the width k of a given multi-k-expression. For these tasks, the running time as a function of the multi-clique-width is the same as the running time of the fastest known algorithm as a function of the clique-width. This results in an exponential speed-up for some graphs, if the corresponding graph generating expressions are given. The reason is that the multi-clique-width is never bigger, but is exponentially smaller than the clique-width for many graphs. This gap shows up when the tree-width is basically equal to the multi-clique width as well as when the tree-width is not bounded by any function of the clique-width.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, D.2.8 Metrics

Keywords and phrases clique-width, parameterized complexity, tree-width, independent set polynomial, graph coloring

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.14

1 Introduction

Tree-width is the first and by far the most important width parameter. It is motivated by the fact that almost all interesting problems that are hard for general graphs allow efficient algorithms when restricted to trees. Furthermore such algorithms are often quite trivial. The promise of the notion of tree-width is to extend such efficient algorithms to much larger classes of tree-like graphs. Graphs of bounded tree-width have one shortcoming though. They are all sparse.

Clique-width [8] is the second most important width parameter. It has been defined by Courcelle and Olariu [12] based on previously used operations [8]. It is intended to compensate for the main shortcoming of the class of graphs of bounded tree-width. The idea is that many graphs are not sparse, but are still constructed in a somewhat simple and uniform way. One would expect to find efficient algorithms for such graphs too. The most extreme example is the clique. It's hard to find a natural problem that is difficult for a clique.

It turns out that graphs of bounded tree-width actually also have bounded clique-width [12, 6], and many efficient algorithms extend to the larger class. Indeed, every graph property expressible in $\mathcal{M}S_1$, the monadic second order logic with set quantifiers for vertices only,

[*] This work was partially supported by NSF Grant CCF-1320814.
[†] Part of this work has been done while visiting Theoretical Computer Science, ETH Zürich, Switzerland

is decidable in linear time (linear in $|V| + |E|$) for graphs of bounded clique-width [10]. On the other hand, graph properties expressible in $\mathcal{M}S_2$, the monadic second order logic with set quantifiers for vertices and edges, is decidable in linear time for graphs of bounded tree-width [7], but not for graphs of bounded clique-width [10] under suitable complexity assumptions.

Recently the split-matching-width (sm-width), a new width parameter has been defined [25] to handle some of the $\mathcal{M}S_2$ expressible problems for unbounded tree-width. The strength of sm-width is strictly between tree-width and clique-width. We will not discuss it, because we focus on parameters that are at least as strong as clique-width.

We are concerned with the fact that the containment of the bounded tree-width graphs in bounded clique-width graphs is not obvious. Furthermore, the generalization from bounded tree-width to bounded clique-width does not come cheap. The width can blow up exponentially, with a potential for a significant loss of efficiency for many algorithms.

This creates a cumbersome situation for the many problems that have efficient solutions in terms of tree-width as well as in terms of clique-width, assuming the corresponding decompositions are known. One would like to run the algorithm based on clique-width to cover a much larger class of graphs, but that would mean an exponential sacrifice in running time for some graphs with small tree-width. Arguably, there should be a notion of a width parameter that bridges this gap more graciously.

Naturally, this aesthetic goal is not the whole story. One could run a tree-width and a clique-width based algorithm and take the result from the one that finishes first. Maybe the following argument is more important. If every clique-width based algorithm is exponentially slower than a tree-width based algorithm for a typical graph of small tree-width, then this seems to indicate some deficiency of the notion of clique-width. Therefore, what we really want, and what we will achieve, is finding a natural notion of width, such that for many important computational problems, algorithms based on this width have all of the following desirable properties.

- The algorithms based on this width are at least as fast as algorithms based on tree-width.
- The algorithms based on this width are at least as fast as algorithms based on clique-width.
- The algorithms based on this width are exponentially faster than algorithms based on clique-width, not only for graphs of small tree-width, but for many graphs of arbitrary large tree-width.

The current author has searched for some time for a parameter naturally generalizing tree-width and clique-width. Ideally, there should be no exponential blow-up in the parameter value. The second objective has been obtained with the notion of fusion-width [18]. It has been shown before that the fusion operation does not produce unbounded clique-width graphs from bounded clique-width graphs [9]. Indeed, there is a much tighter relationship. Graphs of tree-width k have fusion-width at most $k + 2$ [18], while in the worst case, they have clique-width exponential in k [6].

This is a very desirable property of fusion width. The drawback is that attaching a fusion operation is somewhat unnatural. It is an artificial push of the tree-width concept into a clique-width-like environment. Here, we investigate a far more natural width parameter that achieves this goal in a more direct way. We call it *multi-clique-width*, it is obtained by a simple modification in the notion of clique-width, namely by allowing every vertex to have multiple labels. Furthermore, it seems that the multi-clique-width might often be exponentially smaller than the fusion-width. On the other hand, it can be shown that multi-clique-width is never more than twice the fusion-width.

In this paper, we propose multi-clique-width as a serious contender of clique-width. This powerful parameter has some very desirable properties. Its definition is equally simple and natural as that of clique-width. The multi-clique-width is never bigger than the clique-width, but often exponentially smaller [13]. And most importantly, there is no explosion of the width when moving from tree-width to multi-clique-width. Furthermore, there are interesting algorithms where the dependence of the running time on the (potentially much smaller) multi-clique-width is about the same as the dependence on the clique width for a similar algorithm working with clique-width. Thus, multi-clique-width allows some tasks to be solved much more efficiently than previously known.

There are other important width parameter, the rank-width [22] and the boolean-width [5], that share some significant properties with the multi-clique-width. They too are never bigger than the clique-width and can be exponentially smaller. So why do we want to investigate yet another similar parameter?

Rank-width, boolean-width, clique-width, and multi-clique-width are all equivalent in the sense that the exact same problems are solvable in polynomial time for bounded width. If one of these parameters is bounded (by a constant), then so are the others. Rank-width has been introduced with this equivalence in mind [22]. Before, the graphs of bounded clique-width could not be identified computationally. Therefore, graphs of bounded clique-width could only be handled efficiently, when a corresponding k-expression had been given. Now the rank-width $\mathrm{rw}(G)$ can be approximated by a constant factor, and a $2^{3\mathrm{rw}(G)+2} - 1$-expression can be computed in time $O(f(k)\, n^9 \log n)$ for some function f [22]. The later algorithm [20] to compute the rank-width exactly with a similar running time has not much effect on the approximability of clique-width. In theoretical investigations, the exponential bound on the clique-width has not often been viewed as a major concern, because the main goal has been to handle bounded clique-width graphs in polynomial time, not to speed them up, even when the rank-width is much smaller than the clique-width.

There are algorithms based directly on rank-width [19]. Still, for many application algorithms, clique-width based algorithms are more natural and easier to design.

We will show that for some computational tasks multi-clique-width can be used with the same ease as clique-width, but with an exponential speed-up for many graphs. These are exponential speed-ups in the parameter, meaning that the class of graphs with bounded parameter value would not change, just the computations get much faster.

Many questions are not yet answered for multi-clique-width. We conjecture multi-clique-width to be NP-hard. Nevertheless it might be that multi-clique-width could be approximated by a constant factor in polynomial time. We don't yet know. These are open problems for clique-width and boolean-width too.

In comparison with boolean-width or rank-width, when the corresponding decompositions are given, we notice that the known efficient algorithms for NP-complete problems are quadratic in n for bounded width, while for $\mathcal{M}S_1$ expressible graph properties, we have linear time algorithms [10] for bounded clique-width or multi-clique-width. Furthermore, we will illustrate that such algorithms can be quite simple and efficient as a function of the multi-clique-width.

Naturally, one could argue that this is an unfair comparison. The linear time is a result of the structural information that is given by the expression defining a graph with a bound on the clique-width or multi-clique-width. Boolean-width or rank-width on the other hand are defined with existential properties that constantly require some search in the graph. There are no concise boolean expressions or rank expressions defining these graphs. There is truth in this argument, but at the same time, one could claim that this is still an advantage of clique-width and multi-clique width. Maybe, some nice classes of graphs have nice expressions describing them. If such expressions are known or can easily be found, they certainly should be used for efficient algorithms.

2 Definitions and Preliminaries

We use the standard notions of tree decomposition, tree-width, k-expression, and clique-width.

▶ **Definition 1.** A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{B_i : i \in I\}, T)$, where $T = (I, F)$ is a tree and each node $i \in I$ has a subset $B_i \subseteq V$ of vertices (called the bag of i) associated to it with the following properties.

1. $\bigcup_{i \in I} B_i = V$, i.e., each vertex belongs to at least one bag.
2. For all edges $e = \{p, q\} \in E$, there is at least one $i \in I$ with $\{p, q\} \subseteq B_i$, i.e., each edge is represented by at least one bag.
3. For every vertex $v \in V$, the set of indices i of bags containing v induces a subtree of T (i.e., a connected subgraph of T).

▶ **Definition 2.** The *width of a tree decomposition* is 1 less than its largest bag size. The *tree-width* $\text{tw}(G)$ [24] of a graph G is the width of a minimal width tree decomposition of G.

It is NP-complete to decide whether the tree-width of a graph is at most k (if k is part of the input) [1]. For every fixed k, there is a linear time algorithm deciding whether the tree-width is at most k, and if that is the case, producing a corresponding tree decomposition [2]. For arbitrary k, this task can still be approximated. A tree decomposition of width $O(k \log n)$ can be found in polynomial time [4], and in time $O(c^k n)$ almost a 5-approximation [3] can be found (a tree of width at most $5k + 4$ to be precise). Furthermore, a tree decomposition of width $O(k^2)$ can be found in time $O(k^7 n \log n)$ [17].

It is often convenient to view T as a rooted tree, where an arbitrary fixed node has been chosen as the root.

▶ **Definition 3.** A *semi-smooth tree decomposition* of width k is a rooted tree decomposition where the bag B_i of every node i contains exactly 1 vertex that is not in the bag of the parent node $p(i)$. For rooted trees T with $v \in B_i \setminus B_{p(i)}$, we say that *node i is the* home *of vertex v.*

In other words, the home of a vertex v is the highest node whose bag contains v. The bag B_r of the root r of a semi-smooth tree decomposition contains just one vertex.

▶ **Proposition 4.** *Every graph $G = (V, E)$ has a semi-smooth tree decomposition of width $k = \text{tw}(G)$ with $|I| = |V|$. Any tree decomposition of bounded width can be transformed into a semi-smooth tree decomposition in linear time.*

Proof. Do a depth-first search of the tree and omit nodes whose bag is contained in the bag of the parent. Insert intermediate nodes if more than one vertex has the same home. ◀

We use the standard notation of k-expression to define clique-width.

▶ **Definition 5.** A *k-expression* is an expression formed from the atoms $i(v)$, the two unary operations $\eta_{i,j}$ and $\rho_{i \to j}$, and one binary operation $\oplus$ as follows.

- $i(v)$ creates a vertex v with label i, where i is from the set $\{1, \ldots, k\}$.
- $\eta_{i,j}$ creates an edge between every vertex with label i and every vertex with label j for $i \neq j$ (with $i, j \in \{1, \ldots, k\}$).
- $\rho_{i \to j}$ changes all labels i to j (with $i, j \in \{1, \ldots, k\}$).
- $\oplus$ (join-operation) does a disjoint union of the generated labeled graphs.

Finally, the *generated graph* is obtained by deleting the labels.

We also allow multi-way join-operations, as $\oplus$ is associative.

▶ **Definition 6.** The *clique-width* $\mathrm{cw}(G)$ of a graph is the smallest k such that the graph can be defined by a k-expression [12].

Computing the clique-width is NP-hard [15]. Thus, one usually assumes that a graph is given together with a k-expression.

Theoretically, this is not necessary, because for constant k, the clique-width can be approximated in polynomial time [22, 21]. But the approximation ratio is exponential in k.

Now we define multi-clique-width in a similar way as clique-width. The essential difference is that every vertex can have any set of labels (including singleton sets and the empty set). There is a new operation ϵ_i to delete a label. The creation of multiple vertices with the same set of labels by one command is an unessential convenience.

▶ **Definition 7.** A *multi-k-expression* is an expression formed from the atoms $m\langle i_1, \ldots, i_\ell \rangle$, the three unary operations $\eta_{i,j}$, $\rho_{i \to S}$, and ϵ_i, as well as the binary operation $\oplus$ as follows. Assume $i, j \in \{1, \ldots, k\}$, $\ell \in \{0, \ldots, k\}$ and $\emptyset \subseteq S, \{i_1, \ldots, i_\ell\} \subseteq \{1, \ldots, k\}$).

- $m\langle i_1, \ldots, i_\ell \rangle$ with m a positive integer and $i_1 < \cdots < i_\ell \le k$, creates m vertices, each with label set $\{i_1, \ldots, i_\ell\}$.
- $\eta_{i,j}$ creates an edge between every vertex u with label i and every vertex v with label j. This operation is only allowed when there are no vertices with label i and j simultaneously, in particular $i \neq j$.
- $\rho_{i \to S}$ replaces label i by the set of labels S, i.e., if a vertex v had label set S' with $i \in S'$ before this operation, then v has label set $(S' \setminus i) \cup S$ after the operation.
- ϵ_i deletes the label i from all vertices.
- $\oplus$ (join-operation) does a disjoint union of the generated labeled graphs.

Finally, the *generated graph* is obtained by deleting the labels.

S and $\{i_1, \ldots, i_\ell\}$ are allowed to be empty, even though the latter is not very interesting, as it only creates isolated vertices. Note that ϵ_i is just the special case of $\rho_{i \to S}$ with $S = \emptyset$. We list it separately, because one might want to consider the *strict multi-k-expressions* without $\rho_{i \to S}$. In Theorem 13 below, $\rho_{i \to S}$ is not used. Alternatively, one might restrict $\rho_{i \to S}$ to the classical case with S being a singleton. The relative power of these 3 versions might be worth studying.

▶ **Definition 8.** The *multi-clique-width* $\mathrm{mcw}(G)$ of a graph is the smallest k such that the graph can be defined by a multi-k-expression.

It turns out that multiply labeled graphs have been used before [12] in a more auxiliary role, and a variant of the multi-clique-width has actually appeared in the literature under the name m-clique-width [13] in the context of preprocessing for shortest path routing computations.

Here we list the standard definition of boolean-width in order to compare it with multi-clique-width.

▶ **Definition 9.** A *decomposition tree* of a graph $G = (V, E)$ is a tree T where V is the set of leaves and where all internal nodes have degree 3.
Every edge e of T defines a partition of V into X and $\overline{X}$ consisting of the leaves of the two trees obtained from T by removing e.
The *set of unions of neighborhoods* of X across the cut $\{X, \overline{X}\}$ is the set

$$U(X) = \{S' \subseteq \overline{X} \mid \exists S \subseteq X \;\; S' = N(S) \cap \overline{X}\}.$$

X defines bool-dim$(X) = \log_2 |U(X)|$.
The *boolean-width* of G is the minimum over all trees T of the maximum over all cuts $\{X, \overline{X}\}$ defined by an edge e of T of bool-dim$(X) = \log_2 |U(X)|$.

3 Relationship between Different Width Parameters

Multi-clique-width extends the notions of tree-width and of clique-width in a natural way.

▶ **Proposition 10.** *For every graph G, $\mathrm{mcw}(G) \le \mathrm{cw}(G) \le 2^{\mathrm{mcw}(G)}$.*

Proof. The first inequality directly follows from the definitions. For the second inequality, just use a label for every set of labels. ◀

▶ **Corollary 11.** *A class of graphs has bounded clique-width if and only if it has bounded multi-clique-width.*

▶ **Corollary 12.** *Properties of graphs expressible in $\mathcal{M}S_1$ (monadic second order logic without quantifiers over sets of edges) are linear time decidable for graphs of bounded multi-clique-width.*

Proof. This follows from Corollary 11 and the corresponding meta-theorem for clique-width [10]. ◀

▶ **Theorem 13.** *If a tree-decomposition of width k of a graph $G = (V, E)$ is given, then a multi-$(k+2)$-expression for G can be found in polynomial time.*

Proof. Assume, G is given with a tree decomposition of width $k = \mathrm{tw}(G)$. In linear time, the tree decomposition is transformed into a semi-smooth tree decomposition $(\{B_i : i \in I\}, T)$. Now we assign an identifier $\iota(v)$ from $\{1, 2, \ldots, k+1\}$ to each vertex v top-down, i.e., starting at the root of T. When identifiers have been assigned to the vertices whose homes are above the home of vertex v, we assign to vertex v the smallest identifier not assigned to the other vertices in the bag of the home of v.

Next, we define a multi-$(k+2)$-expression whose parse tree T' is basically isomorphic to the tree T of the tree decomposition. The difference it that in T' every internal node has an additional child that is a leaf. We call it an auxiliary leaf. Furthermore, above each internal node i, we introduce three auxiliary nodes obtained by subdividing the edge to the parent of i.

The main idea is that every vertex v is created at its home, or more precisely, in the auxiliary node below its home. Then the edges from v to neighbors of v with a home further down the tree are added. The upper neighbors of v, i.e., those that have their home higher up the tree, are not yet created. Vertex v remembers to attach to these neighbors later by taking the set of identifiers of these neighbors as its labels. All upper neighbors of v are together with v in the bag B_i of the home i of v in T. The vertex v needs at most k labels for this purpose. We give v an additional label, $k+2$, to allow the lower neighbors of v to connect to v. Node i of T' is a multi-way join operation of all its children, including the new auxiliary child. The purpose of the three nodes inserted above node i is to add the edges between v and its neighbors in the subtree of i, and to delete the two labels ($k+2$, and $\iota(v)$, the identifier of v) that have been used to create these new edges. The multi-$(k+2)$-expression is built bottom-up.

Now we define the multi-$(k+2)$-expression exactly by assigning atoms to the leaves and operations to the internal nodes as follows.

Regular leaf: Let the leaf i be the home of some vertex v. Let $v_1, \dots, v_\ell$ be the neighbors of v with identifiers $i_1, \dots, i_\ell$. Clearly, $\{v, v_1, \dots, v_\ell\} \subseteq B_i$. Then the expression $1\langle i_1, \dots, i_\ell \rangle$ creates v in leaf i.

Auxiliary leaf: Let the internal node i be the home of some vertex v. Let $v_1, \dots, v_\ell$ be the upper neighbors of v with identifiers $i_1, \dots, i_\ell$. Let $c_0(i)$ be the child of i which is an auxiliary leaf. Then the expression for $c_0(i)$ is $1\langle k+2, i_1, \dots, i_\ell \rangle$.

Internal node: Let i be the home of some vertex v, and let $c_1(i), \dots, c_q(i)$ be the children of i in T. Let $c_0(i)$ be the auxiliary leaf child of i in T'. Furthermore, let $\iota(v)$ be the identifier of v. Assume, for child $c_j(i)$ we already have the expression E_j. Then the multi-$(k+2)$-expression for node i, or more precisely of the third auxiliary node above it, is

$$\epsilon_{k+2}(\epsilon_{\iota(v)}(\eta_{\iota(v),k+2}(E_0 \oplus E_1 \oplus \cdots \oplus E_q))).$$

Now the following is easily proved by induction on the height of node i.

▶ **Claim 14.** *The multi-$(k+2)$-expression for node i generates the labeled graph $G_i = (V_i, E_i)$ induced by the vertices whose home is in the subtree of i. Furthermore, the set of labels of every vertex $v \in V_i$ is equal to the set of identifiers of the neighbors of v in $V \setminus V_i$.*

By the inductive hypothesis of the claim, all vertices V' in the subtree of node i that are adjacent to v in G have a label $\iota(v)$. The vertex v has a label $k+2$, but no label $\iota(v)$. Thus the operation $\eta_{\iota(v),k+2}$ creates exactly the edges between v and V'. Now, the labels $\iota(v)$ and $k+2$ can be deleted, because both have served their purpose. From every vertex labeled $\iota(v)$, the edge to v is now already constructed, and the label $k+2$ only had to mark the vertex v for the construction of these edges.

The claim for the root implies the theorem. ◀

A weaker form of Theorem 13 is the implied inequality between multi-clique-width and tree-width.

▶ **Corollary 15.** *For every graph G,* $\mathrm{mcw}(G) \le \mathrm{tw}(G) + 2$.

As an immediate corollary, we obtain $cw(G) \le 2^{\mathrm{tw}(G)+2}$. The tighter bound of $cw(G) \le 2^{\mathrm{tw}(G)+1} + 1$ [12] is obtained by noticing that one could use the label $k+2$ strictly as a singleton label. Instead of deleting it with an ϵ_{k+2} operation, one could change it to the set of other labels we wanted to assign to that vertex using a $\rho_{i \to S}$ operation. The even tighter bound $cw \le 1.5 \cdot 2^{\mathrm{tw}(G)}$ [6] is obtained by handling higher degree join nodes more efficiently. Following every binary join, the necessary edges could be inserted, allowing the number of labels to be decreased. This saves one fourth of the labels.

▶ **Corollary 16.** *There are graphs G with* $\mathrm{cw}(G) \ge 2^{\lfloor \mathrm{mcw}(G)/2 \rfloor - 2}$.

Proof. There are graphs G with $\mathrm{tw}(G) = k$ and clique-width $\mathrm{cw}(G) \ge 2^{\lfloor k/2 \rfloor - 1}$ [6]. Such graphs have multi-clique-width $\mathrm{mcw}(G) \le k + 2$ by Corollary 15. ◀

Naturally, it is easy to find graph classes with unbounded tree-width that still exhibit this exponential discrepancy between clique-width and multi-clique-width. One way is just to add a large clique, but there are many not so obvious ways.

We want to compare multi-clique-width with boolean-width.

▶ **Theorem 17.** *For every graph G,* $\mathrm{boolw}(G) \le \mathrm{mcw}(G) \le 2^{\mathrm{boolw}(G)}$.

Proof. $\operatorname{boolw}(G) \leq \operatorname{mcw}(G)$: Assuming $\operatorname{mcw}(G) = k$, we start with a multi-$k$-expression for G. W.l.o.g., assume that each vertex v is created as a single vertex by the operation $m\langle i_1, \ldots, i_j\rangle$ with $m = 1$. Then, there is a bijection between the vertices V and the leaves of the parse tree T. Viewed as a graph, the other nodes of T have degrees 2 or 3. We replace all maximal paths with internal nodes of degree 2 by single edges to obtain a tree T'.

Consider any edge $e = (u, v)$ of T', where v is a descendant of u in T. Let $X \subseteq V$ be the set of vertices of the subtree T_v. For every subset $S \subseteq X$, the set $N(S) \cap \overline{X}$ of neighbors of S outside of X only depends on the union of the set of labels of the vertices of S. There are at most 2^k such subsets of labels, and thus at most 2^k such neighborhoods. The binary logarithm of the largest such number of neighborhoods over all edges of T' is an upper bound on $\operatorname{boolw}(G)$, i.e., $\operatorname{boolw}(G) \leq k$.

$\operatorname{mcw}(G) \leq 2^{\operatorname{boolw}(G)}$: By Proposition 10, $\operatorname{mcw}(G) \leq \operatorname{cw}(G)$, and the inequality $\operatorname{cw}(G) \leq 2^{\operatorname{boolw}(G)}$ [5] is known. ◀

Even though, boolean-width has the desirable property $\operatorname{boolw}(G) \leq \operatorname{mcw}(G)$, sometimes more efficient algorithms are possible in terms of $\operatorname{mcw}(G)$ than in terms of $\operatorname{boolw}(G)$. Indeed, every graph property expressible in $\mathcal{M}S_1$, is decidable in linear time for graphs of bounded clique-width [10], while for arbitrary graphs of bounded boolean-width, at least quadratic time is required. Naturally, this can also be viewed as an indication of the strength of the boolean-width parameter. Even graphs without a simple structure can have small boolean-width. In the next section, we will see that for specific problems the (exponential) dependance of the running time on the multi-clique-width can be quite good.

4 Algorithms based on Multi-Clique-Width

The algorithmic purpose of clique-width and other width parameters is to put problems into FPT, i.e., making them fixed parameter tractable (see [14]). This means achieving a running time of $O(f(k)n^{O(1)})$ for an arbitrary computable function f. In reality things are not so bad. Algorithms based on clique-width often have a running time of $O(c^k n^e)$ or $O(c^{k \log k} n^e)$ with $k = \operatorname{cw}(G)$, $n = |V|$, and c and e being small constants.

Assume that we are given a multi-k-expression for G and we have an algorithm with similar running time when k is the multi-clique-width. Then we have an exponential time speed-up when choosing the multi-clique based algorithm with running time $2^{O(k)}n^e$, instead of the clique-width based algorithm with clique-width $2^{\Omega(k)}$ and running time $2^{2^{\Omega(k)}}n^e$ for infinitely many graphs.

Indeed, we want to illustrate here that this scenario is occurring quite frequently. Let us see that there are many graphs with unbounded tree-width, multi-clique-width k, and clique-width exponential in k. Without changing the notion of multi-clique-width, we could extend the multi-k-expressions to allow arbitrary $k/2$-labeled graphs of tree-width at most $k/2 - 2$ as atoms instead of just isolated vertices. It is not had to see that such graphs could be produced by proper multi-k-expressions. $k/2$ labels are sufficient to simulate the tree-width construction, and the remaining $k/2$ labels could be placed in any possible way.

But if at least one of the tree-width $k/2-2$ components are complicated, then the clique-width is exponential in k [6]. And if the graph contains a large clique of size ℓ (which could be $\Omega(n)$), then the tree-width is at least ℓ. Thus, we have a huge class of graphs where the multi-clique-width is exponentially better than the clique-width.

Now we exhibit the ease of use of the parameter $\operatorname{mcw}(G)$ for Independent Set. The running time as a function of the width is roughly the same for clique-width k as for multi-clique-width k. Hence, we gain an exponential speed-up in the width parameter for all the many instances were the clique-width is exponentially bigger than the multi-clique-width.

Instead of only finding a maximum independent set, or even just computing its size, we solve the more involved problem of computing the independent set polynomial, i.e., computing the numbers of independent sets of all sizes. This is not much more difficult, and one can easily simplify the algorithm if only a maximum independent set is needed. Then the dependence of the running time on the size n goes down to linear from polynomial, while the dependence on the width k stays singly exponential. In particular, we have an FPT algorithm to compute the independent set polynomial. We refer to [11, 16] for more discussions of the fixed parameter tractability of counting problems.

It is worth mentioning that the very same problem of computing the number of independent sets of all sizes has been studied by Bui-Xuan et al. [5]. They achieve a running time of $O(\alpha^2 n^2 k 2^{2k})$ for graphs with boolean-width k and maximum independent set size α. Note that α could be as large as $\Omega(n)$. We will not elaborate on our running time, but just remark that using FFTs, $O(\alpha \log \alpha \, nk2^k)$ is sufficient for multi-clique-width k.

▶ **Definition 18.** The independent set polynomial of a graph G is

$$I(x) = \sum_{i=1}^{n} a_i x^i$$

where a_i is the number of independent sets of size i in G.

The independent set polynomial is not strong enough to describe the involvement of the different labels in the independent sets. We need to do a more detailed counting to allow recurrence equations to govern the definition of the polynomials as the labeled graph is assembled by a multi-k-expression.

▶ **Definition 19.** Let $[k] = \{1, \ldots, k\}$ be the set of vertex labels. The $[k]$-labeled independent set polynomial of a $[k]$-labeled graph G (each vertex can have multiple labels from $[k]$) is

$$P(x, x_1, \ldots, x_k) = \sum_{i=1}^{n} \sum_{(n_1,\ldots,n_k)\in\{0,1\}^k} a_{i;n_1,\ldots,n_k} \, x^i \prod_{j=1}^{k} x_j^{n_j}$$

where $n_j \in \{0, 1\}$ and $a_{i;n_1,\ldots,n_k}$ is the number of independent sets of size i in G which contain some vertices with label j if and only if $n_j = 1$.

The $[k]$-labeled independent set polynomial is used in the inductive construction. At the end, the independent set polynomial $I(x)$ is easily obtained from $[k]$-labeled independent set polynomial.

▶ **Theorem 20.** *Given a graph G with n vertices, and a multi-k-expression generating G with multi-clique-width k, the independent set polynomial $I(x)$ of G can be computed in time $O(2^k(kn)^{O(1)})$.*

Proof. Using dynamic programming, we compute the $[k]$-labeled independent set polynomial of the $[k]$-labeled graphs generated by subexpressions of the given multi-k-expression. The computation is done bottom-up in the parse tree of the given multi-k-expression.

For any atomic expression $m\langle i_1, \ldots, i_j\rangle$ creating m vertices with labels $i_1, \ldots, i_j$, we have the $[k]$-labeled independent set polynomial

$$1 + \sum_{\ell=1}^{m} \binom{m}{\ell} x^\ell x_{i_1} \cdots x_{i_j} = 1 + ((1+x)^m - 1) x_{i_1} \cdots x_{i_j}.$$

In $O(m)$ arithmetic operations, we can compute all coefficients using the recurrence $\binom{m}{\ell+1} = \binom{m}{\ell}(m-\ell)/(\ell+1)$. Thus all atomic expressions for the $n = |V|$ vertices can be computed in time $O(n)$.

If the expression E has the polynomial $\tilde{P}(x, x_1, \ldots, x_k)$, then $\eta_{i,j}(E)$ has the polynomial

$$P(x, x_1, \ldots, x_k) = \tilde{P}(x, x_1, \ldots, x_k) \mod x_i x_j,$$

i.e., terms containing x_i and x_j are deleted. This is correct, because a set of vertices is independent after the introduction of the edges between labels i and j, if and only if it was an independent set before and does not contain both labels i and j.

If the expression E has the polynomial $\tilde{P}(x, x_1, \ldots, x_k)$, then $\rho_{i \to S}(E)$ has the polynomial

$$\begin{aligned} P(x, x_1, \ldots, x_k) = {} & \tilde{P}(x, x_1, \ldots, x_{i-1}, x_{i_1} \cdots x_{i_j}, x_{i+1}, \ldots, x_k) \\ & \mod (x_{i_1}^2 - x_{i_1}) \cdots \mod (x_{i_j}^2 - x_{i_j}) \end{aligned} \tag{1}$$

for $S = \{i_1, \ldots, i_j\}$, i.e., first x_i is replaced by the product $x_{i_1} \ldots x_{i_j}$. Then squares of indeterminates are replaced by their first powers. This is correct, because we still count all independent sets. They just occur in different categories as they involve different labels.

If the expression E has the polynomial $\tilde{P}(x, x_1, \ldots, x_k)$, then $\epsilon_i(E)$ has the polynomial

$$P(x, x_1, \ldots, x_k) = \tilde{P}(x, x_1, \ldots x_{i-1}, 1, x_{i+1}, \ldots, x_k),$$

i.e., the indeterminate x_i is replaced by 1. This is correct, because it is just a special case $\rho_{i \to S}(E)$.

If the expression E_ℓ ($\ell \in \{1, 2\}$) has the polynomial $\tilde{P}_\ell(x, x_1, \ldots, x_k)$, then the expression $E_1 \oplus E_2$ has the polynomial

$$\begin{aligned} P(x, x_1, \ldots, x_k) = {} & \tilde{P}_1(x, x_1, \ldots, x_k)\, \tilde{P}_2(x, x_1, \ldots, x_k) \\ & \mod (x_1^2 - x_1) \cdots \mod (x_k^2 - x_k), \end{aligned} \tag{2}$$

i.e., in the product of the polynomials, every x_i^2 is replaced by x_i, as we only care about the occurrence of a label and not about the multiplicity of such an occurrence. This is correct, because every independent set of G_1 can be combined with every independent set of G_2 to form an independent set of the join graph G, and every independent set of G can be formed in this way.

Finally,

$$I(x) = \sum_{i=1}^{n} a_i x^i = \sum_{i=1}^{n} \sum_{(n_1, \ldots, n_k) \in \{0,1\}^k} a_{i, n_1, \ldots, n_k} x^i = P(x, 1, \ldots, 1),$$

because
$a_i = \sum_{(n_1, \ldots, n_k) \in \{0,1\}^k} a_{i, n_1, \ldots, n_k}$.

To bound the running time, one should notice that the polynomial $P(x, x_1, \ldots, x_k)$ has $2^k(n+1)$ coefficients.

The polynomial $\tilde{P}(x, x_1, \ldots, x_{i-1}, x_{i_1} \cdots x_{i_j}, x_{i+1}, \ldots, x_k)$ in Eq. (1) has at most $2^k(n+1)$ non-zero coefficients, not $3^k(n+1)$, because the substitution does not increase the number of monomials. If the product in Eq. (2) is computed by school multiplication, then the running time is $O(4^k (kn)^{O(1)})$. But with a fast Fourier transform (evaluating the polynomial for $x_i = 0$ and $x_1 = 1$ for all i), the time is only $O(2^k (kn)^{O(1)})$. ◀

The easier problem of just finding the size of a maximum independent set (rather than computing the numbers of independent sets of all sizes) is now trivial. At each stage, for all exponents $n_1, \ldots, n_k$, the coefficient $a_{i;n_1,\ldots n_k}$ is only stored for the largest i with $a_{i;n_1,\ldots,n_k} \neq 0$.

▶ **Corollary 21.** *A maximum independent set can be found in time $O(2^k k^{O(1)} n)$ in graphs with multi-clique-width k.*

Proof. If during the dynamic programming algorithm to compute the size of a maximum independent set, one always stores where the largest exponent i came from, then at the end, one can easily backtrace to actually find a maximum independent set. ◀

As an additional example, we consider the NP-complete decision problem c-coloring, asking whether the input graph G can be colored with c colors for a constant integer $c \geq 3$, such that no adjacent vertices have the same color.

▶ **Theorem 22.** *For graphs G of multi-clique-width k with a given multi-k-expression for G, and any positive integer constant c, the c-coloring problem can be solved in time $2^{O(ck)}n$.*

Proof. We present a dynamic programming algorithm based on the parse tree structure of the multi-k-expression. We classify the colorings of the graphs generated by sub-expressions according to the labels used for the vertices of each color. Let Q with $|Q| = c$ be the set of colors and L with $|L| = k$ be the set of labels. Let $B_1, \ldots, B_r$ with $r = 2^{ck}$ be the sequence (say in lexicographic order) of all bipartite graphs with the left vertex set Q and the right vertex set L. Let E_p be the set of edges in B_p. For every subexpression F, we define $F(B_p)$ to be true, if and only if the following holds. The graph generated by F can be colored with Q such that some vertex colored with $q \in Q$ is labeled with a set of labels containing $i \in L$, if and only if (q, i) is an edge in B_p.

We now show that $F(B_p)$ can easily be computed from all the $F'(B_{p'})$ where F' is a subexpression of F and $p' \in \{1, \ldots, r\}$. We analyze according to the structure of F.

If F is an atomic expression $m\langle i_1, \ldots, i_j \rangle$ creating m vertices with labels $i_1, \ldots, i_j$, then $F(B_p)$ is true, if and only if $E_p = \{(q, i) \mid q \in Q' \text{ and } i \in \{i_1, \ldots, i_j\}\}$ for some Q' with $\emptyset \neq Q' \subseteq Q$.

If $F = \eta_{i,j}(F')$, then $F(B_p)$ is true, if and only if $F'(B_p)$ is true and for no color $q \in Q$ there are both edges (q, i) and (q, j) present in B_p. In other words, a previous coloring is still valid, if and only if no color appears at both endpoints of newly added edges.

If $F = \rho_{i \to S}(F')$, then $F(B_p)$ is true, if and only if $F'(B_{p'})$ is true for some p' with

$$E_p = \{(q, \ell) \mid (q, \ell) \in E_{p'} \text{ and } \ell \neq i\} \cup \{(q, j) \mid (q, i) \in E_{p'} \text{ and } j \in S\}.$$

If $F = F' \oplus F''$, then $F(B_p)$ is true, if and only if $F'(B_{p'})$ is true and $F''(B_{p''})$ is true for some p', p'' with $E_p = E_{p'} \cup E_{p''}$.

Given this simple characterization of $F(B_p)$ in terms of $F'(B_{p'})$ for some p' and the immediate sub-expressions F', it should be immediately clear how the value of $F(B_p)$ can be computed, when the values of the $F'(B_{p'})$ are known.

Furthermore, it is a simple proof by induction on the structure of an expression F that $F(B_p)$ is true, if and only if the graph generated by F can be colored with Q such that some vertex colored with $q \in Q$ is labeled with a set of labels containing $i \in L$, if and only if (q, i) is an edge in B_p.

Naturally, at the end, the graph generated by F is k-colorable, if and only if $F(B_p)$ is true for some B_p.

The running time is linear in n, because there are $O(n)$ nodes to process and the time spent in every node only depends on the number c of colors and the number k of labels. In every node, an array of 2^{ck} boolean values (one for each bipartite graph on the vertex sets Q and L) has to be processed in a simple fashion. The resulting running time is $2^{O(ck)}n$.

There is quite some waste of time involved in handling all the bipartite graphs on the vertex sets Q and L, because the truth value for a graph B_j does not change, when the set of colors Q and the set of labels L are permuted in an arbitrary way. This does not mean that the running time can be divided by $c!k!$, because typically many such permutations are automorphisms not creating new bipartite graphs. The exact number of isomorphism types of such bipartite graphs can be computed with the Redfield-Pólya enumeration theorem (see [23]), but that does not result in a nicer upper bound. Clearly, any practical implementation would do the computation for just one bipartite graph for every isomorphism type. ◀

5 Conclusions and Open Problems

We have proposed a powerful parameter multi-clique-width. It allows us to achieve faster running times for natural classes of graphs and interesting algorithmic tasks. Assume, we are given the input graph by a multi-k-expression. Then we have very efficient algorithms for this class of graphs, as illustrated by the independent set polynomial and the coloring problem. On the other hand, for any algorithm based on clique-width, we could only get exponentially slower (in k) algorithms for the same problems and the same collection of graphs. Also, equally efficient algorithms are not known based on rank-width or boolean-width, when the corresponding decompositions are given.

Most questions related to the new multi-clique-width are still open. Is it difficult to compute or approximate? We expect it to be NP-hard, like clique-width. We also conjecture it to be in FPT (fixed parameter tractable) and to be constant factor approximable in time singly exponential in the multi-clique-width and linear in the length like tree-width. But obviously this is very difficult, as it is also open for clique-width.

A main question is whether most algorithms for clique-width k, can be extended to work with similar efficiency for multi-clique-width k. We have illustrated that this is the case for some interesting counting and decision problems. On the other hand, there is the question of identifying the problems where this is not the case.

References

1 Stefan Arnborg, D. G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Alg. and Discrete Methods*, 8:277–284, 1987.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

3 Hans L. Bodlaender, Pål G. Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $O(c^k n)$ 5-approximation algorithm for treewidth. In *Proc. 54th FOCS 2013*, pages 499–508. IEEE, 2013.

4 Hans L. Bodlaender, John R. Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

5 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.

6 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005. doi:10.1137/S0097539701385351.

7 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput*, 85(1):12–75, March 1990. doi:10.1016/0890-5401(90)90043-H.

8 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. `doi:10.1016/0022-0000(93)90004-G`.

9 Bruno Courcelle and Johann A. Makowsky. Fusion in relational structures and the verification of monadic second-order properties. *Mathematical Structures in Computer Science*, 12(2):203–235, 2002.

10 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. `doi:10.1007/s002249910009`.

11 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001. `doi:10.1016/S0166-218X(00)00221-3`.

12 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. `doi:10.1016/S0166-218X(99)00184-5`.

13 Bruno Courcelle and Andrew Twigg. Constrained-path labellings on graphs of bounded clique-width. *Theory Comput. Syst.*, 47(2):531–567, 2010. URL: `http://springerlink.metapress.com/content/b3268gtk313180q0/`.

14 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

15 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width minimization is NP-hard. In Jon M. Kleinberg, editor, *STOC*, pages 354–362. ACM, 2006. `doi:10.1145/1132516.1132568`.

16 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008. `doi:10.1016/j.dam.2006.06.020`.

17 Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *CoRR*, abs/1511.01379, 2015. URL: `http://arxiv.org/abs/1511.01379`.

18 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. *Proceedings of LATIN 2014: Theoretical Informatics - 11th Latin American Symposium. Springer LNCS*, 8392:72–83, 2014. `doi:10.1007/978-3-642-54423-1`.

19 Robert Ganian and Petr Hlinený. On parse trees and myhill-nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics*, 158(7):851–867, 2010. `doi:10.1016/j.dam.2009.10.018`.

20 Petr Hlinený and Sang il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput*, 38(3):1012–1032, 2008. `doi:10.1137/070685920`.

21 Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):10:1–10:20, December 2008. `doi:10.1145/1435375.1435385`.

22 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. `doi:10.1016/j.jctb.2005.10.006`.

23 G. Pólya and R. C. Read. *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds*. Springer-Verlag, New York, 1987.

24 Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. `doi:10.1016/0095-8956(84)90013-3`.

25 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. In Dieter Kratsch and Ioan Todinca, editors, *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, volume 8747 of *Lecture Notes in Computer Science*, pages 396–407. Springer, 2014. `doi:10.1007/978-3-319-12340-0`.