Tight Algorithmic Applications of Clique-Width Generalizations

Vera Chekan
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

In this work, we study two natural generalizations of clique-width introduced by Martin Fürer. Multi-clique-width (mcw) allows every vertex to hold multiple labels [ITCS 2017], while for fusion-width (fw) we have a possibility to merge all vertices of a certain label [LATIN 2014]. Fürer has shown that both parameters are upper-bounded by treewidth thus making them more appealing from an algorithmic perspective than clique-width and asked for applications of these parameters for problem solving. First, we determine the relation between these two parameters by showing that $\mathrm{mcw} \leq \mathrm{fw} + 1$. Then we show that when parameterized by multi-clique-width, many problems (e.g., Connected Dominating Set) admit algorithms with the same running time as for clique-width despite the exponential gap between these two parameters. For some problems (e.g., Hamiltonian Cycle) we show an analogous result for fusion-width: For this we present an alternative view on fusion-width by introducing so-called glue-expressions which might be interesting on their own. All algorithms obtained in this work are tight up to (Strong) Exponential Time Hypothesis.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized complexity, connectivity problems, clique-width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.35

Related Version *Full Version*: https://arxiv.org/abs/2307.04628 [8]

Funding *Vera Chekan*: Supported by DFG Research Training Group 2434 "Facets of Complexity".

1 Introduction

In parameterized complexity apart from the input size we consider a so-called parameter and study the complexity of problems depending on both the input size and the parameter where the allowed dependency on the input size is polynomial. In a more fine-grained setting one is interested in the best possible dependency on the parameter under reasonable conjectures. A broad line of research is devoted to so-called *structural parameters* measuring how simple the graph structure is: different parameters quantify various notions of possibly useful input structure. Probably the most prominent structural parameter is treewidth, which reflects how well a graph can be decomposed using small vertex separators. For a variety of problems, the tight complexity parameterized by treewidth (or its path-like analogue pathwidth) has been determined under the so-called Strong Exponential Time Hypothesis (e.g., [33, 24, 32, 11, 28, 12, 16]). However, the main drawback of treewidth is that it is only bounded in sparse graphs: a graph on n vertices of treewidth k has no more than nk edges.

To capture the structure of dense graphs, several parameters have been introduced and considered. One of the most studied is clique-width. The clique-width of a graph is at most k if it can be constructed using the following four operations on k-labeled graphs: create a vertex with some label from $1, \dots, k$; form a disjoint union of two already constructed graphs; give all vertices with label i label j instead; or create all edges between vertices

© Vera Chekan and Stefan Kratsch;
licensed under Creative Commons License CC-BY 4.0

with labels i and j. It is known that if a graph has treewidth k, then it has clique-width at most $3 \cdot 2^{k-1}$ and it is also known that an exponential dependence in this bound is necessary [9]. Conversely, cliques have clique-width at most 2 and unbounded treewidth. So on the one hand, clique-width is strictly more expressive than treewidth in the sense that if we can solve a problem efficiently on classes of graphs of bounded clique-width, then this is also true for classes of graphs of bounded treewidth. On the other hand, the exponential gap has the effect that as the price of solving the problem for larger graph classes we potentially obtain worse running times for some graph families.

Fürer introduced and studied two natural generalizations of clique-width, namely fusion-width (fw) [19] and multi-clique-width (mcw) [20]. For fusion-width, additionally to the clique-width operations, he allows an operator that fuses (i.e., merges) all vertices of label i. Originally, fusion-width (under a different name) was introduced by Courcelle and Makowsky [10]. However, they did not suggested studying it as a new width parameter since it is parametrically (i.e., up to some function) equivalent to clique-width. For multi-clique-width, the operations remain roughly the same as for clique-width but now every vertex is allowed to have multiple labels. For these parameters, Fürer showed the following relations to clique-width (cw) and treewidth (tw):

$$\mathrm{fw} \leq \mathrm{cw} \leq \mathrm{fw} \cdot 2^{\mathrm{fw}} \qquad \mathrm{mcw} \leq \mathrm{cw} \leq 2^{\mathrm{mcw}} \qquad \mathrm{fw} \leq \mathrm{tw} + 2 \qquad \mathrm{mcw} \leq \mathrm{tw} + 2 \tag{1}$$

Fürer also observed that the exponential gaps between clique-width and both fusion- and multi-clique-width are necessary. As our first result, we determine the relation between fusion-width and multi-clique-width:

▶ **Theorem 1.** *For every graph G, it holds that* $\mathrm{mcw}(G) \leq \mathrm{fw}(G) + 1$*. Moreover, given a fuse-k-expression ϕ of G, a multi-clique-width-$(k+1)$-expression of G can be created in time polynomial in $|\phi|$ and k.*

The relations in (1) imply that a problem is FPT parameterized by fusion-width resp. multi-clique-width if and only if this is the case for clique-width. However, the running times of such algorithms might strongly differ. Fürer initiated a fine-grained study of problem complexities relative to multi-clique-width, starting with the Independent Set problem. He showed that this problem can be solved in $\mathcal{O}^*(2^{\mathrm{mcw}})$ where $\mathcal{O}^*$ hides factors polynomial in the input size. On the other hand, Lokshtanov et al. proved that under SETH no algorithm can solve this problem in $\mathcal{O}^*((2-\varepsilon)^{\mathrm{pw}})$ where pw denotes the parameter called pathwidth [28]. Clique-width of a graph is at most its pathwidth plus two [15] so the same lower bound holds for clique-width and hence, multi-clique-width as well. Therefore, the tight dependence on both clique-width and multi-clique-width is the same, namely $\mathcal{O}^*(2^k)$. We show that this is the case for many further problems.

▶ **Theorem 2.** *Let G be a graph given together with a multi-k-expression of G. Then:*

- *Dominating Set can be solved in time $\mathcal{O}^*(4^k)$;*
- *q-Coloring can be solved in time $\mathcal{O}^*((2^q-2)^k)$;*
- *Connected Vertex Cover can be solved in time $\mathcal{O}^*(6^k)$;*
- *Connected Dominating Set can be solved in time $\mathcal{O}^*(5^k)$.*

And these results are tight under SETH.

Further, Chromatic Number can be solved in time $f(k) \cdot n^{2^{\mathcal{O}(k)}}$ and this is tight under ETH.

We prove this by providing algorithms for multi-clique-width with the same running time as the known tight algorithms for clique-width. The lower bounds for clique-width known from the literature then apply to multi-clique-width as well proving the tightness of our results.

By Theorem 1, these results also apply to fusion-width. For the following three problems we obtain similar tight bounds relative to fusion-width as for clique-width, but it remains open whether the same is true relative to multi-clique-width:

▶ **Theorem 3.** *Let G be a graph given together with a fuse-k-expression of G. Then:*

- *Max Cut can be solved in time $f(k) \cdot n^{\mathcal{O}(k)}$;*
- *Edge Dominating Set can be solved in time $f(k) \cdot n^{\mathcal{O}(k)}$;*
- *Hamiltonian Cycle can be solved in time $f(k) \cdot n^{\mathcal{O}(k)}$.*

And these results are tight under ETH.

To prove these upper bounds, we provide an alternative view on fuse-expressions, called *glue-expressions*, interesting on its own. We show that a fuse-k-expression can be transformed into a glue-k-expression in polynomial time and then present dynamic-programming algorithms on glue-expressions. Due to the exponential gap between clique-width and both fusion- and multi-clique-width, our results provide exponentially faster algorithms on graphs witnessing these gaps.

Related Work. Two parameters related to both treewidth and clique-width are modular treewidth (mtw) [3, 22] and twinclass-treewidth [29, 31, 27] (unfortunately, sometimes also referred to as modular treewidth). It is known that $\text{mcw} \leq \text{mtw} + 3$ (personal communication with Falko Hegerfeld). Further dense parameters have been widely studied in the literature. Rank-width (rw) was introduced by Oum and Seymour and it reflects the $\mathbb{F}_2$-rank of the adjacency matrices in the so-called branch decompositions. Originally, it was defined to obtain a fixed-parameter approximation of clique-width [30] by showing that $\text{rw} \leq \text{cw} \leq 2^{\text{rw}+1} - 1$. Later, Bui-Xuan et al. started the study of algorithmic properties of rank-width [5]. Recently, Bergougnoux et al. proved the tightness of first ETH-tight lower bounds for this parameterization [2]. Another parameter defined via branch-decompositions and reflecting the number of different neighborhoods across certain cuts is boolean-width (boolw), introduced by Bui-Xuan et al. [6, 7]. Fürer [20] showed that $\text{boolw} \leq \text{mcw} \leq 2^{\text{boolw}}$. Recently, Eiben et al. presented a framework unifying the definitions and algorithms for computation of many graph parameters [13].

Organization. We start with some required definitions and notations in Section 2. In Section 3 we prove the relation between fusion-width and multi-clique-width from Theorem 1. After that, in Section 4 we introduce glue-k-expressions and show how to obtain such an expression given a fuse-k-expression of a graph. Then in Section 5 we employ these expressions to obtain algorithms parameterized by fusion-width. In Section 6 we present algorithms parameterized by multi-clique-width. We conclude with some open questions in Section 7. In this work some technical details have been omitted due to space constraints. We refer to the full version of the paper for all proofs [8].

2 Preliminaries

For $k \in \mathbb{N}_0$, we denote by $[k]$ the set $\{1, \dots, k\}$ and we denote by $[k]_0$ the set $[k] \cup \{0\}$.

We use standard graph-theoretic notation. Our graphs are simple and undirected if not explicitly stated otherwise. For a graph H and a partition (V_1, V_2) of $V(H)$, by $E_H(V_1, V_2) = \{\{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2\}$ we denote the set of edges between V_1 and V_2. For a set S of edges in a graph H, by $V(S)$ we denote the set of vertices incident with the edges in S.

A k-labeled graph is a pair (H, lab_H) where $\mathrm{lab}_H \colon V(H) \to [k]$ is a *labeling function* of H. Sometimes to simplify the notation in our proofs we will allow the labeling function to map to some set of cardinality k instead of the set $[k]$. In the following, if the number k of labels does not matter, or it is clear from the context, we omit k from the notions (e.g., a labeled graph instead of a k-labeled graph). Also, if the labeling function is clear from the context, then we simply call H a labeled graph as well. Also we sometimes omit the subscript H of the labeling function lab_H for simplicity. For $i \in [k]$, by $U_i^H = \mathrm{lab}_H^{-1}(i)$ we denote the set of vertices of H with label i. We consider the following four operations on k-labeled graphs.

1. *Introduce*: For $i \in [k]$, the operator $v\langle i \rangle$ creates a graph with a single vertex v that has label i. We call v the *title* of the vertex.
2. *Union*: The operator $\oplus$ takes two vertex-disjoint k-labeled graphs and creates their disjoint union. The labels are preserved.
3. *Join*: For $i \neq j \in [k]$, the operator $\eta_{i,j}$ takes a k-labeled graph H and creates the supergraph H' on the same vertex set with $E(H') = E(H) \cup \{\{u, v\} \mid \mathrm{lab}_H(u) = i, \mathrm{lab}_H(v) = j\}$. The labels are preserved.
4. *Relabel*: For $i \neq j$, the operator $\rho_{i \to j}$ takes a k-labeled graph H and creates the same k-labeled graph H' apart from the fact that every vertex that with label i in H instead has label j in H'.

A well-formed sequence of such operations is called a *k-expression* or a *clique-expression*. With a k-expression ϕ one can associate a rooted tree such that every node corresponds to an operator, this tree is called a *parse tree* of ϕ. With a slight abuse of notation, we denote it by ϕ as well. By G^ϕ we denote the labeled graph arising in ϕ. And for a node t of ϕ by G_t^ϕ we denote the labeled graph arising in the subtree (sometimes also called a *sub-expression*) rooted at t, this subtree is denoted by ϕ_t. The graph G_t^ϕ is then a subgraph of G^ϕ. A graph H has *clique-width* of at most k if there is a labeling function lab_H of H and a k-expression ϕ such that G^ϕ is equal to (H, lab_H). By $\mathrm{cw}(H)$ we denote the smallest integer k such that H has clique-width at most k. Fürer has studied two generalizations of k-expressions [19, 20].

Fuse: For $i \in [k]$, the operator θ_i takes a k-labeled graph H with $\mathrm{lab}_H^{-1}(i) \neq \emptyset$ and *fuses* the vertices with label i, i.e., the arising graph H' has vertex set $(V(H) - \mathrm{lab}_H^{-1}(i)) \dot\cup \{v\}$, the edge relation in $V(H) - \mathrm{lab}_H^{-1}(i)$ is preserved, and $N_{H'}(v) = N_H(\mathrm{lab}_H^{-1}(i))$. The labels of vertices in $V(H') - v$ are preserved, and vertex v has label i. A *fuse-k-expression* is a well-formed expression that additionally to the above four operations is allowed to use fuses. We adopt the above notations from k-expressions to fuse-k-expressions. Let us only remark that for a node t of a fuse-k-expression ϕ, the graph G_t^ϕ is not necessarily a subgraph of G^ϕ since some vertices of G_t^ϕ might be fused later in ϕ.

▶ Remark 4. Originally, Fürer allows that a single introduce-node creates multiple, say q, vertices with the same label. However, we can eliminate such operations from a fuse-expression ϕ as follows. If the vertices introduced at some node participate in some fuse later in the expression, then it suffices to introduce only one of them. Otherwise, we can replace this introduce-node by q nodes introducing single vertices combined using union-nodes. These vertices are then also the vertices of G^ϕ. So in total, replacing all such introduce-nodes would increase the number of nodes of the parse tree by at most $\mathcal{O}(|V(G^\phi)|)$, which is not a problem for our algorithmic applications.

Another generalization of clique-width introduced by Fürer [20] is multi-clique-width (mcw). A multi-k-labeled graph is a pair (H, lab_H) where $\mathrm{lab}_H \colon V(H) \to 2^{[k]}$ is a multi-labeling function. We consider the following four operations of multi-k-labeled graphs.

1. *Introduce*: For $q \in [k]$ and $i_1, \dots i_q \in [k]$, the operator $v\langle i_1, \dots, i_q \rangle$ creates a multi-k-labeled graph with a single vertex that has label set $\{i_1, \dots, i_q\}$.
2. *Union*: The operator $\oplus$ takes two vertex-disjoint multi-k-labeled graphs and creates their disjoint union. The labels are preserved.
3. *Join*: For $i \neq j \in [k]$, the operator $\eta_{i,j}$ takes a multi-k-labeled graph H and creates its supergraph H' on the same vertex set with $E(H') = E(H) \cup \{\{u, v\} \mid i \in \text{lab}_H(u), j \in \text{lab}_H(v)\}$. This operation is only allowed when there is no vertex in H with labels i and j simultaneously, i.e., for every vertex v of H we have $\{i, j\} \not\subseteq \text{lab}_H(v)$. The labels are preserved.
4. *Relabel*: For $i \in [k]$ and $S \subseteq [k]$, the operator $\rho_{i \to S}$ takes a multi-k-labeled graph H and creates the same multi-labeled graph apart from the fact that every vertex with label set $L \subseteq [k]$ such that $i \in L$ in H instead has label set $(L \setminus \{i\}) \cup S$ in H'. Note that $S = \emptyset$ is allowed.

A well-formed sequence of these four operations is called a *multi-k-expression*. As for fuse-expressions, Fürer allows introduce-nodes to create multiple vertices but we can eliminate this by increasing the number of nodes in the expression by at most $\mathcal{O}(|V(G^\phi)|)$. We adopt the analogous notations from k-expressions to multi-k-expressions.

Complexity. To the best of our knowledge, the only known way to approximate multi-clique-width and fusion-width is via clique-width, i.e., to employ the relation (1). The only known way to approximate clique-width is, in turn, via rank-width. This way we obtain a 2^{2^k}-approximation of multi-clique-width and fusion-width running in FPT time. For this reason, to obtain tight running times in our algorithms we always assume that a fuse- or multi-k-expression is provided. Let us emphasize that this is also the case for all tight results for clique-width in the literature (see e.g., [1, 27]). In this work, we will show that if a graph admits a multi-k-expression resp. a fuse-k-expression, then it also admits one whose size is polynomial in the size of the graph. Moreover, such a "compression" can be carried out in time polynomial in the size of the original expression. Therefore, we delegate this compression to a black-box algorithm computing or approximating multi-clique-width or fusion-width and assume that provided expressions have size polynomial in the graph size.

(Strong) Exponential Time Hypothesis. The algorithms in this work are tight under one of the following conjectures formulated by Impagliazzo et al. [23]. The Exponential Time Hypothesis (ETH) states that there is $0 < \varepsilon < 1$ such that 3-SAT with n variables and m clauses cannot be solved in time $\mathcal{O}^*(2^{\varepsilon n})$. The Strong Exponential Time Hypothesis (SETH) states that for every $0 < \varepsilon < 1$ there is an integer q such that q-SAT cannot be solved in time $\mathcal{O}^*(2^{\varepsilon n})$. In this work, $\mathcal{O}^*$ hides factors polynomial in the input size.

Simplifications. If the graph is clear from the context, by n we denote the number of its vertices. If not stated otherwise, the number of labels is denoted by k and a label is a number from $[k]$.

3 Relation Between Fusion-Width and Multi-Clique-Width

In this section, we show that the multi-clique-width of a graph is at most as large as its fusion-width plus one. Fürer [20] has proven the following relation:

▶ **Theorem 5** ([20]). *For every graph H, it holds that $\text{cw}(H) \leq \text{fw}(H) \cdot 2^{\text{fw}(H)}$.*

The proof is constructive: given a fuse-k-expression it creates a $k \cdot 2^k$-expression of the same graph. We use ideas from this proof to prove our result.

▶ **Theorem 6.** *For every graph H, it holds that* $\mathrm{mcw}(H) \leq \mathrm{fw}(H) + 1$*. Moreover, given a fuse-k-expression ϕ of H, a multi-$(k+1)$-expression of H can be created in time polynomial in $|\phi|$ and k.*

Proof. Here we sketch the intuition behind the proof and for all formalities we refer to the full version. We start by showing that $\mathrm{mcw}(H) \leq 2 \cdot \mathrm{fw}(H)$ holds. To prove this, we will consider a fuse-k-expression of H and from it, we will construct a multi-$2k$-expression of H using labels $\{1, \dots, k, \widehat{1}, \dots, \widehat{k}\}$. For simplicity, let $\widehat{[k]} = \{\widehat{1}, \dots, \widehat{k}\}$. For this first step, we follow the construction of Fürer in his proof of Theorem 5. There he uses $k \cdot 2^k$ labels from the set $[k] \times 2^{[k]}$ so the second component of such a label is a subset of $[k]$. Multi-expressions allow vertices to hold multiple labels and we model the second component of a label via subsets of $\widehat{[k]}$. After that, we show that the labels i and $\widehat{i}$ can be almost unified for every $i \in [k]$. Using just one additional label $\star$, we then obtain a multi-$(k+1)$-expression of H.

For simplicity of representation, in this proof sketch we assume that our fuse-expression contains no relabel-nodes (we refer to the full version for the complete proof). We may assume that our fuse-expression does not contain nodes that do not change the arising labeled graph. Also if a vertex arising in some fuse-operation participates in some later fuse-operation, then the earlier fuse can be removed. Now we assume that any vertex arising in a fuse-operation does not participate in later fuses. We say that v is a *fuse-vertex* at a node x of the expression if v *participates* in some fuse-operation above x. For the label i of v we then also say that i is a *fuse-label* at x. Instead of first creating the fuse-vertices via introduce-nodes and then fusing them, we will introduce only one vertex representing the result of the fusion. The creation of the edges incident with such a *new* vertex (originally incident with fuse-vertices) then needs to be *postponed* until the moment where this vertex is introduced. To remember that some vertex v is missing an edge to a new vertex with label i, we will add a label $\widehat{i}$ to the label set of v. After the creation of this vertex, the edge will be reconstructed using a corresponding join.

Now we provide more details. First, we cut off every leaf of the expression introducing a fuse-vertex. Second, we replace every fuse-node θ_i by a *new* introduce-node $1\langle i \rangle$. Since for every fuse-vertex we have kept only the latest fuse-node it participates in, the vertex set of the graph arising in the current expression is $V(H)$, and the edges incident with new vertices need to be created. For this, let x be a $\eta_{i,j}$-node. If both i and j are not fuse-labels at x, no additional work needs to be done. Next, assume that exactly one of the labels i and j, say i, is a fuse-label at x. The information about the created edges is stored in vertices of label j: for this, we replace this join with a relabel $\rho_{j \to \{j, \widehat{i}\}}$. Now assume that both i and j are fuse-labels at x. Then x creates only one edge of H since all vertices of label i (resp. j) are fused into one vertex later. We may assume that in the original expression, the fuse of vertices with label j happens before the fuse of label i. We store the information about the postponed edge in j as follows. Let x_j be the new introduce-node that replaced the fuse-node for label j. And let S denote the set of labels of the new vertex created in this node: in the beginning, S consists of j only but after processing some join-nodes, it might contain further labels from $\widehat{[k]}$. Then we replace x_j with a $1\langle S \cup \{\widehat{i}\} \rangle$-node. Finally, we create the remembered edges as follows. Let x be a new $1\langle S \rangle$-node for some set S of labels. By construction, there exists a unique $i \in S \cap [k]$: this is the label of vertices originally fused at this node while $S \setminus i \subseteq \widehat{[k]}$ remembers the edges to be created later in the expression. So right after x, we first add a $\eta_{i,\widehat{i}}$-node and second, a $\rho_{\widehat{i} \to \emptyset}$-node to reflect that the missing edges have been created.

This concludes the construction of a multi-$2k$-expression of H. The crucial observation is that the labels i and $\widehat{i}$ are almost never used at the same time. Indeed, if label $\widehat{i}$ occurs in some vertex created by some sub-expression, then all leaves introducing the vertices of label i have been cut off from this sub-expression. So there are no vertices with label i now. The only moment to which both labels i and $\widehat{i}$ occur simultaneously is right after a new $1\langle S\rangle$-node with $i \in S$: now the label $\widehat{i}$ is used to create the postponed edges incident with the new vertex and then $\widehat{i}$ is removed. So apart from these two operations, we can unify i and $\widehat{i}$: one additional label $\star$ suffices to distinguish them during these operations. This results in a multi-$(k+1)$-expression of H. ◀

4 Reduced Glue-Expressions

▶ **Definition 7.** *A* glue-k-expression *is a well-formed expression constructed from introduce-, join-, relabel-, and glue-operations on k-labeled graphs. A glue-operation takes as input two k-labeled graphs (H_1, lab_1) and (H_2, lab_2) satisfying the following two properties:*

- *For every $v \in V(H_1) \cap V(H_2)$, the vertex v has the same label in H_1 and H_2, i.e., we have $\mathrm{lab}_1(v) = \mathrm{lab}_2(v)$.*
- *For every $v \in V(H_1) \cap V(H_2)$ and every $j \in [2]$, the vertex v is the unique vertex with its label in H_j, i.e., we have $|\mathrm{lab}_1^{-1}(\mathrm{lab}_1(v))| = |\mathrm{lab}_2^{-1}(\mathrm{lab}_2(v))| = 1$.*

In this case, we call the graphs H_1 and H_2 glueable. *The output of this operation is then the graph $H_1 \sqcup H_2$ with $V(H_1 \sqcup H_2) = V(H_1) \cup V(H_2)$ and $E(H_1 \sqcup H_2) = E(H_1) \cup E(H_2)$ where the labels are preserved. The vertices in $V(H_1) \cap V(H_2)$ are called* glue-vertices.

Note that if H_1 and H_2 satisfy these properties, then the gluing is equivalent to a union followed by a sequence of fuses θ_i where i is a label of a vertex shared by H_1 and H_2.

▶ **Definition 8.** *A glue-k-expression ξ is called* reduced *if the following properties hold:*

1. *Let $i, j \in [k]$, let t be a $\eta_{i,j}$-node in ξ, and let t' be the child of t in ξ. Then $G^\xi_{t'}$ contains no edge $\{v, w\}$ with $\mathrm{lab}^\xi_{t'}(v) = i$ and $\mathrm{lab}^\xi_{t'}(w) = j$.*
2. *Let t be a glue-node in ξ and let t_1 and t_2 be its children. Then the graphs $G^\xi_{t_1}$ and $G^\xi_{t_2}$ are edge-disjoint.*
3. *Let t be a glue-node in ξ, let t_1 and t_2 be its children, and let v be a glue-vertex. Then for every $j \in [2]$, the vertex v has an incident edge in $G^\xi_{t_j}$.*

As we will see in Section 5, such expressions are very useful for algorithmic applications.

In this section we sketch how to transform a fuse-k-expression into a reduced glue-k-expression of the same graph in polynomial time. Although the idea behind the construction of such expressions is quite natural, the realization is rather technical. For this reason, here we only provide a high-level idea of the required steps and refer to the full version for the extensive description.

In the first phase, we transform a fuse-k-expression into a (not necessarily reduced) glue-k-expression as follows. As the first step, we want to achieve that every fuse-node t is right above some union-node, i.e., there are only fuse-nodes on the path between t and the topmost union-node below t. For this, we shift every fuse-node down to the closest union-node (see Figure 1 (a)). We emphasize that this cannot be achieved simply by repetitive swapping of the fuse-node with its child: e.g., swapping θ_i with $\rho_{j \to i}$ would change the arising graph. In the second step, we want to achieve that every fuse-node t fuses exactly two vertices and these vertices come from different sides of the union right below it. In other words, we want that any two vertices, that are ever fused, are fused as early as possible. So if there is a node t fusing at least two vertices coming from the same side of the topmost union-node $t_\oplus$

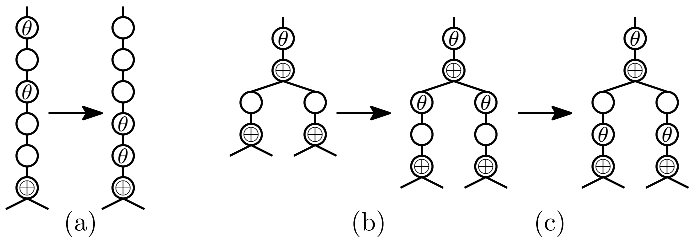

Figure 1 The steps to transform a fuse-expression into a glue-expression. (a) Shifting fuse-nodes to union-nodes. (b) Creating copies of fuse-nodes to fuse vertices as early as possible. (c) Shifting the copied fuse-nodes to union-nodes.

below it, these vertices can actually be fused before $t_\oplus$ already. To accomplish this, we add a new fuse-node t' below $t_\oplus$ (see Figure 1 (b)). After that, t' is shifted down to the next union-node as in the first step (see Figure 1 (c)). In this second step, one needs to be careful in order to ensure that the process terminates at all and that it takes only polynomial time. For this, we choose an appropriate order for processing the fuse-nodes. After these two steps, we replace every sequence consisting of a union-node and following fuse-nodes with a glue-node to obtain a glue-expression.

In the second phase, we make our glue-expression reduced as follows (see Definition 8). First, given two join-nodes creating the same edge, we can show that at least of them can be safely removed. Second, if a glue-vertex of two glued graphs has no incident edge in one of them, then we can remove this vertex from that graph without changing the result of the gluing. We refer to the full version for details. Altogether, we prove the following:

▶ **Theorem 9.** *Let ϕ be a fuse-k-expression of a graph H on n vertices and m edges. Then in time polynomial in $|\phi|$ and k we can compute a reduced glue-k-expression ζ of H such that the parse tree of ζ contains $\mathcal{O}(k^2(m+n))$ nodes.*

Let us remark, that unlike clique-expressions (whose leaves are in bijection with the vertices of the arising graph), the number of leaves in a fuse-expression can be unbounded in general. This is due to fuse-nodes reducing the number of vertices in the constructed graph. So Theorem 9 in particular shows that a polynomial number of introduce-nodes suffices.

5 Algorithms Parameterized by Fusion-Width

In this section we show how to employ reduced glue-expressions to obtain algorithms parameterized by fusion-width (given a fuse-expression of corresponding width). In the previous section, we showed that every fuse-k-expression can be transformed into a reduced glue-k-expression of small size in polynomial time. So for the remainder of this section we assume that a glue-k-expression ϕ of an input graph G is given. Recall that in particular, two graphs glued at any glue-node of ϕ are edge-disjoint.

All algorithms in this section have running time $f(\mathrm{fw})n^{\mathcal{O}(\mathrm{fw})}$. For each of the problems considered here, in the literature there is a lower bound stating that under ETH, the problem cannot be solved in time $f(\mathrm{cw})n^{o(\mathrm{cw})}$ even if a clique-expression of corresponding width is provided [17, 18]. Since fusion-width is at most as large as clique-width, these lower bounds hold for fusion-width as well implying the tightness of our results. Fürer observed that there exist graphs whose clique-width is exponential in their fusion-width [19]. Therefore, in addition to solving these problems for a larger class of graphs, we obtain an exponential improvement in the running time for some families of graphs. The only difference between glue- and clique-expressions is the possibility of using glue-nodes so it will suffice to extend existing dynamic-programming algorithms from the literature to glue-nodes.

5.1 Max Cut

In this problem, given a graph $G = (V, E)$ we are asked about the maximum cardinality of $E_G(V_1, V_2)$ over all partitions (V_1, V_2) of V. Fomin et al. have developed an $n^{\mathcal{O}(\text{cw})}$ algorithm [17], which we now extend to reduced glue-expressions by showing how to deal with glue-nodes. Before this, a small remark: the correctness of their algorithm for a join-node t requires that none of edges created by t is already present in the graph. A reduced glue-expression satisfies this property so the procedures for all node types other than glue-nodes can indeed be adopted.

Their dynamic-programming tables are defined as follows. For a graph H, the table T_H contains all vectors $h = (s_1, \ldots, s_k, r)$ with $0 \leq s_i \leq |U_i^H|$ for every $i \in [k]$ and $0 \leq r \leq |E(H)|$ for which there exists a partition (V_1, V_2) of $V(H)$ such that $|V_1 \cap U_i^H| = s_i$ for every $i \in [k]$ and there are at least r edges between V_1 and V_2 in H. We say that the partition (V_1, V_2) *witnesses* the vector h. Then the output of the algorithm is the largest integer r such that T_G contains an entry $(s_1, \ldots, s_k, r)$ for some $s_1, \ldots, s_k \in \mathbb{N}_0$. Processing a fuse-operation θ_i applied to some subgraph H seems to be problematic in this setting for the following reason. Some vertices of label i might have common neighbors so after the application of the fuse-operator, multiple edges fall together. Given the table T_H only we cannot deduce how many of those edges were running across the partition and it is unclear how to update the table correctly. To avoid such issues, we replace fuse-nodes with glue-nodes.

Now we provide a way to compute the table T_H if $H = H^1 \sqcup H^2$ for two glue-able edge-disjoint k-labeled graphs H^1 and H^2 if the tables T_{H^1} and T_{H^2} are provided. Let $\{v_1, \ldots, v_q\} = V(H^1) \cap V(H^2)$ for some $q \in \mathbb{N}_0$ and let $i_1, \ldots, i_q$ be the labels of $v_1, \ldots, v_q$ in H^1, respectively. Glueability implies that for every $j \in [q]$, it holds that $|U_{i_j}^{H^1}| = |U_{i_j}^{H^2}| = 1$. Hence, for every entry $(s_1, \ldots, s_k, r)$ of T_{H^1} and every $j \in [q]$, it also holds that $s_{i_j} \in \{0, 1\}$ with $s_{i_j} = 1$ if and only if v_j is put into V_1 in the partition witnessing this entry. The same holds for the entries in T_{H^2}. This gives the following way to compute the table T_H. We initialize this table to be empty. Then we iterate through all pairs of vectors $h^1 = (s_1^1, \ldots, s_k^1, r^1)$ from T_{H^1} and $h^2 = (s_1^2, \ldots, s_k^2, r^2)$ from T_{H^2}. If there is an index $j \in [q]$ such that $s_{i_j}^1 \neq s_{i_j}^2$, then we skip this pair. Otherwise, for every $0 \leq r \leq r^1 + r^2$, we add to T_H the vector $h = (s_1, \ldots, s_k, r)$ where for all $i \in [k]$

$$s_i = \begin{cases} s_i^1 + s_i^2 & i \notin \{i_1, \ldots, i_q\} \\ s_i^1 \cdot s_i^2 & i \in \{i_1, \ldots, i_q\} \end{cases}.$$

Note that for $i \in \{i_1, \ldots, i_q\}$, the above definition simply states that we have $s_i = 1$ iff both s_i^1 and s_i^2 are equal to 1, and $s_i = 0$ iff both s_i^1 and s_i^2 are equal to 0. It is not difficult to verify the correctness of this procedure. If there are no glue-vertices, then our approach coincides with the one for union-nodes by Fomin et al. If there is a glue-vertex, say v, then we consider partitions (V_1^1, V_2^1) and (V_1^2, V_2^2) of H^1 and H^2, respectively, putting v on the same side. Then a partition of H naturally arises as a union $(V_1^1 \cup V_1^2, V_1^2 \cup V_2^2)$. Since the graphs are edge-disjoint, every edge crossing this partition crosses exactly one of the partitions (V_1^1, V_2^1) and (V_1^2, V_2^2) implying that the choice of r is correct. We refer to the full version for a formal proof.

Recall that every graph constructed from any sub-expression of ϕ is a subgraph of G so the tables for graphs H^1 and H^2 contain $n^{\mathcal{O}(k)}$ entries. Thus, the procedure for glue-nodes runs in time $n^{\mathcal{O}(k)}$. By Theorem 9 we may assume that ϕ contains a polynomial number of nodes. Altogether, we obtain an algorithm solving Max Cut in time $n^{\mathcal{O}(\text{fw})}$. Fomin et al. have also proven that under ETH, it cannot be solved in time $f(\text{cw})n^{o(\text{cw})}$ for any computable function f (see [17] Theorem 4.1) so our result is tight.

5.2 Edge Dominating Set

In this problem, given a graph $G = (V, E)$ we are asked about the cardinality of a minimum set $X \subseteq E$ such that every edge in E either belongs to X itself or it has an incident edge in X. Fomin et al. have developed an $n^{\mathcal{O}(\mathrm{cw})}$ algorithm solving this problem [17], which we now extend to reduced glue-expressions. As for Max Cut, the algorithm requires that for any join-node, the edges created by this node are not yet present in the graph. A reduced glue-expression satisfies this property so their procedures for introduce-, join-, and relabel-nodes can be adopted.

For a k-labeled graph H, the table T_H contains all vectors $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$ of non-negative integers such that there exists a set $S \subseteq E(H)$ and a set $R \subseteq V(H) \setminus V(S)$ with the following properties:

- $|S| \leq \ell \leq |E(H)|$;
- for every $i \in [k]$, exactly s_i vertices of U_i^H are incident with edges in S;
- for every $i \in [k]$, we have $|R \cap U_i^H| = r_i$;
- every edge of H undominated by S has an end-vertex in R.

We say that the pair (S, R) witnesses the vector $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$ in H. The last property reflects that it is possible to attach a *pendant* edge to every vertex in R so that the set S together with these pendant edges dominates all edges of H. The size of the minimum edge dominating set of G is then the smallest integer ℓ such that the table T_G contains an entry $(s_1, \dots, s_k, 0, \dots, 0, \ell)$ for some $s_1, \dots, s_k \in \mathbb{N}_0$.

To complete the algorithm for the fusion-width parameterization, we provide a way to compute the table T_H if $H = H^1 \sqcup H^2$ for two glueable edge-disjoint k-labeled graphs H^1 and H^2 if the tables T_{H^1} and T_{H^2} are provided. Let $\{v_1, \dots, v_q\} = V(H^1) \cap V(H^2)$ for some $q \in \mathbb{N}_0$ and let $i_1, \dots, i_q$ be the labels of $v_1, \dots, v_q$ in H^1, respectively. Then for every $j \in [q]$, it holds that $|U_{i_j}^{H^1}| = |U_{i_j}^{H^2}| = 1$. Hence, for every entry $(s_1, \dots, s_k, r_1, \dots, r_k, \ell)$ of T_{H^1} and every $j \in [q]$, it holds that $s_{i_j} + r_{i_j} \leq 1$. The same holds for the entries in T_{H^2}. This motivates the following way to compute the table T_H. We initialize this table to be empty. Then we iterate through all pairs of vectors $h^1 = (s_1^1, \dots, s_k^1, r_1^1, \dots, r_k^1, \ell^1)$ from T_{H^1} and $h^2 = (s_1^2, \dots, s_k^2, r_1^2, \dots, r_k^2, \ell^2)$ from T_{H^2} and for every $\ell^1 + \ell^2 \leq \ell \leq |E(H)|$, we add to T_H the vector $h = (s_1, \dots, s_k, r_1, \dots, r_k, \ell)$ defined as follows. For every $i \in [k] \setminus \{i_1, \dots, i_q\}$, it holds that $s_i = s_i^1 + s_i^2$ and $r_i = r_i^1 + r_i^2$. And for every $i \in \{i_1, \dots, i_q\}$, it holds that $s_i = s_i^1 \vee s_i^2$ and $r_i = \neg s_i^1 \wedge \neg s_i^2 \wedge (r_i^1 \vee r_i^2)$.

To argue the correctness, we sketch only one direction here. The other is similar and for all details refer to the full version. We now show that if h_1 and h_2 belong to T_{H^1} and T_{H^2}, respectively, then the vector h indeed belongs to T_H. So let (S^1, R^1) witness h^1 in H^1 and let (S^2, R^2) witness h^2 in H^2. Then we set $S = S^1 \cup S^2$ and construct R from $R^1 \cup R^2$ by removing all vertices incident with S. Recall that every vertex in R^1 has no incident edge in S^1. So a vertex $v \in R^1 \setminus R$ must have an incident edge in S^2 by construction and therefore, the vertex v must be a glue-vertex. The analogous is true for $R^2 \setminus R$. With this, one can verify that R complies with $r_1, \dots, r_k$. Also it is straight-forward to verify that this is true for S and $s_1, \dots, s_k$. The bound $\ell^1 + \ell^2 \leq \ell$ implies that the size of S is at most ℓ (when proving the other direction, we use that H^1 and H^2 are edge-disjoint). So it remains to show that every edge of H undominated by S has an end-point in R. Recall that $E(H) = E(H^1) \cup E(H^2)$. Let e be an edge of $E(H^1)$ undominated by $S = S^1 \cup S^2$. Since it is not dominated by S^1, it has an end-point, say v, in R^1. By construction of R, either v still belongs to R or it has an incident edge in S^2. We have assumed that e is not dominated by $S^1 \cup S^2$ so v belongs to R as desired. A symmetric argument applies to edges of H^2. This concludes the proof that h is an entry of T_H.

As in the previous subsection, for any subgraph H of G constructed in some sub-expression of ϕ, the table T_H contains $n^{\mathcal{O}(k)}$ entries and we obtain an algorithm solving Edge Dominating Set in time $n^{\mathcal{O}(\text{fw})}$. Fomin et al. have also proven that under ETH, it cannot be solved in time $f(\text{cw})n^{o(\text{cw})}$ for any computable function f (see [17] Theorem 5.1) so our result is tight.

5.3 Hamiltonian Cycle

In this problem, given a graph $G = (V, E)$ we are asked about the existence of a cycle visiting each vertex exactly once. Our algorithm relies on the algorithm by Bergougnoux et al. [1] running in time $f(\text{cw})n^{\mathcal{O}(\text{cw})}$. A partial solution for this problem is usually a path packing, that is, a set of paths containing every vertex of the graph (constructed by the current sub-expression) exactly once. The earlier $f(\text{cw})n^{\mathcal{O}(\text{cw}^2)}$ algorithm by Espelage et al. stores for every pair of labels i and j, the number of paths in the path packing between a vertex with label i and a vertex with label j [14]. This naturally defines a graph on k vertices in which there is an edge for every path in the path packing. Bergougnoux et al. [1] show that instead of keeping track of all edges, it suffices to remember the degree sequence and the set of connected components of this graph. To obtain an algorithm relying on this idea they employ the technique of so-called representative sets: they define what does it mean for a set of partial solutions to be representative and then show that their procedures for nodes of a clique-expression maintain representativity.

To extend this algorithm to the parameterization by fusion-width, we describe how to handle glue-nodes. Let H_1 and H_2 be two edge-disjoint glueable graphs and let $\mathcal{P}_1$ and $\mathcal{P}_2$ be path packings of H_1 and H_2, respectively. Then a natural combination of these partial solutions is the gluing $\mathcal{P} := \mathcal{P}_1 \sqcup \mathcal{P}_2$ (a subgraph of $H_1 \sqcup H_2$). Unlike a union-node (as handled in [1]), the subgraph $\mathcal{P}$ is not necessarily a path packing: first, glue-vertices might have degree larger than 2 in $\mathcal{P}$ and second, cycles might occur (e.g., if two paths with the same end-vertices are glued). We can show that if we iterate over all $\mathcal{P}_1$ and $\mathcal{P}_2$ and filter out combinations $\mathcal{P}_1 \sqcup \mathcal{P}_2$ that are not path packings, then we obtain exactly the set of all partial solutions of $H_1 \sqcup H_2$. After that, we show that this approach maintains representativity. This part is very technical and we refer to the full version for all details. Let us remark that although we follow the idea by Bergougnoux et al. [1], our proof of correctness gets more involved than the one for union-nodes from their work: when gluing two graphs, the partial solutions are combined in a less trivial way than when forming a disjoint union of two graphs. The tightness is implied by [18].

6 Algorithms Parameterized by Multi-Clique-Width

In this section, we consider algorithms for problems parameterized by multi-clique-width. For all of these problems, SETH-tight (and for Chromatic Number even an ETH-tight) algorithms for clique-width are known, we refer to [18, 25, 27, 21] for the corresponding lower bounds. We show that algorithms with the same running time exist relative to multi-clique-width. The clique-width lower bounds then transfer and imply the tightness of our results. As for fusion-width, Fürer observed that there exist graphs whose clique-width is exponential in their multi-clique-width [20]. So we obtain exponentially faster algorithms for some graph classes.

For our results, we rely on existing algorithms for the parameterization by clique-width and show that these (almost) do not use the fact that every vertex holds exactly one label: Some of the algorithms use clique-expressions with certain properties (e.g., so-called *irredundancy*) though, so we need to either show that such a property holds for multi-clique-width expressions or provide an alternative way to keep the algorithm correct. Now we will sketch for each of the problems what changes need to be carried out and refer to the full version of the paper for all details. Using simple transformations we may assume that every introduce-node has form $1\langle i \rangle$ and every relabel-node has form $\rho_{i \to \emptyset}$ or $\rho_{i \to \{i,j\}}$ for some $i \neq j \in [k]$. Also we may assume that a multi-k-expression contains at most $\mathcal{O}(k^2 n)$ nodes.

For Dominating Set, the algorithm by Bodlaender et al. runs in time $\mathcal{O}^*(4^{\mathrm{cw}})$ [4] and never uses the fact that every vertex holds exactly one label. So using a straight-forward procedure to handle new relabel-nodes (of form $\rho_{i \to \emptyset}$ and $\rho_{i \to \{i,j\}}$) we obtain a $\mathcal{O}^*(4^{\mathrm{mcw}})$ algorithm for this problem.

The situation is similar for the Chromatic Number algorithm by Kobler and Rotics [26]. The only minor thing one needs to handle is that the algorithm assumes that every color used by a graph coloring appears on some label: in a multi-expression, it might occur that all labels are removed from a vertex. To avoid this issue, we increase the number of labels in the expression by 1 and ensure that every vertex holds the new label at all times. Also we provide a procedure for the new relabel-nodes. Apart from that, the algorithm remains the same.

In his work, Fürer presents an algorithm for q-Coloring parameterized by multi-clique-width but it is not tight yet. For the q-Coloring problem, a naive $\mathcal{O}^*((2^q)^{\mathrm{cw}})$ algorithm tracks for every label the set of colors used on this label. Lampis improves this running time to $\mathcal{O}^*((2^q - 2)^{\mathrm{cw}})$ by observing that the empty set of colors can only be used by an empty label, while all colors can only occur if the label does not participate in any join later (such a label is called *dead*): otherwise, a monochromatic edge would occur [27]. Hegerfeld and Kratsch employ the same idea to obtain an $\mathcal{O}^*(6^{\mathrm{cw}})$ algorithm for Connected Vertex Cover [21]. To define *dead* labels, they rely on the existence of so-called irredundant clique-expression whose existence is unknown for multi-clique-width. We show that one can still make these two algorithms work for multi-clique-width by using a slightly different definition of dead and their counterpart, namely *active*, labels. Finally, for Connected Dominating Set, to extend the $\mathcal{O}^*(5^{\mathrm{cw}})$ algorithm by Hegerfeld and Kratsch [21], there slightly more work is needed to adapt their inclusion-exclusion technique to a multi-label setting.

7 Conclusion

In this work, we studied two generalizations of clique-width, namely fusion-width and multi-clique-width, both introduced by Fürer [19, 20]. First, we showed that the fusion-width of a graph is an upper bound for its multi-clique-width. For the other direction, the best upper bound we are aware of is $\mathrm{fw} \leq 2^{\mathrm{mcw}}$ and we leave open whether this is tight. By extending existing algorithms for clique-width, we have obtained tight algorithms parameterized by multi-clique-width for Dominating Set, Chromatic Number, q-Coloring, Connected Vertex Cover, and Connected Dominating Set. The running times are the same as for (S)ETH-optimal algorithms parameterized by clique-width.

For Hamiltonian Cycle, MaxCut, and Edge Dominating Set, we were not able to achieve analogous results and these complexities remain open. Instead, we have introduced glue-expressions equivalent to fuse-expressions and then we employed them for these three problems to obtain tight algorithms parameterized by fusion-width with the same running times as ETH-optimal algorithms for clique-width.

Finally, in all algorithms we assume that a multi-k-expression / fuse-k-expression is provided. However, the complexity of computing these parameters is unknown. To the best of our knowledge, the best approximation would proceed via clique-width, have FPT running time, and a double-exponential approximation ratio.

References

1 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An optimal XP algorithm for hamiltonian cycle on graphs of bounded clique-width. *Algorithmica*, 82(6):1654–1674, 2020. doi:10.1007/s00453-019-00663-9.

2 Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof. Tight lower bounds for problems parameterized by rank-width. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.11.

3 Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nord. J. Comput.*, 7(1):14–31, 2000.

4 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2010. doi:10.1007/978-3-642-15155-2_17.

5 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discret. Appl. Math.*, 158(7):809–819, 2010. doi:10.1016/j.dam.2009.09.009.

6 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.

7 Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Fast dynamic programming for locally checkable vertex subset and vertex partitioning problems. *Theor. Comput. Sci.*, 511:66–76, 2013. doi:10.1016/j.tcs.2013.01.009.

8 Vera Chekan and Stefan Kratsch. Tight algorithmic applications of clique-width generalizations, 2023. Technical report. doi:10.48550/arXiv.2307.04628.

9 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.

10 Bruno Courcelle and Johann A. Makowsky. Fusion in relational structures and the verification of monadic second-order properties. *Math. Struct. Comput. Sci.*, 12(2):203–235, 2002. doi:10.1017/S0960129501003565.

11 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

13 Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A unifying framework for characterizing and computing width measures. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 63:1–63:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.63.

14 Wolfgang Espelage, Frank Gurski, and Egon Wanke. How to solve np-hard graph problems on clique-width bounded graphs in polynomial time. In Andreas Brandstädt and Van Bang Le, editors, *Graph-Theoretic Concepts in Computer Science, 27th International Workshop, WG 2001, Boltenhagen, Germany, June 14-16, 2001, Proceedings*, volume 2204 of *Lecture Notes in Computer Science*, pages 117–128. Springer, 2001. doi:10.1007/3-540-45477-2_12.

15 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.

16 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

18 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

19 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics – 11th Latin American Symposium, Montevideo, Uruguay, March 31 – April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2014. doi:10.1007/978-3-642-54423-1_7.

20 Martin Fürer. Multi-clique-width. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 14:1–14:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.14.

21 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *CoRR*, abs/2302.03627, 2023. doi:10.48550/arXiv.2302.03627.

22 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. *CoRR*, abs/2302.14128, 2023. doi:10.48550/arXiv.2302.14128.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

24 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

25 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

26 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discret. Appl. Math.*, 126(2-3):197–221, 2003. doi:10.1016/S0166-218X(02)00198-1.

27 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

28 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

29 Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016 – 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016. doi:10.1007/978-3-319-40970-2_1.

30 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.

31 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016. doi:10.1007/s00453-015-0030-x.

32 Marcin Pilipczuk. A tight lower bound for vertex planarization on graphs of bounded treewidth. *Discret. Appl. Math.*, 231:211–216, 2017. doi:10.1016/j.dam.2016.05.019.

33 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.