Finer Tight Bounds for Coloring on Clique-Width

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France
michail.lampis@dauphine.fr

Abstract

We revisit the complexity of the classical k-Coloring problem parameterized by clique-width. This is a very well-studied problem that becomes highly intractable when the number of colors k is large. However, much less is known on its complexity for small, concrete values of k. In this paper, we completely determine the complexity of k-Coloring parameterized by clique-width for any fixed k, under the SETH. Specifically, we show that for all $k \geq 3, \epsilon > 0$, k-Coloring cannot be solved in time $O^*\left((2^k - 2 - \epsilon)^{\text{cw}}\right)$, and give an algorithm running in time $O^*\left((2^k - 2)^{\text{cw}}\right)$. Thus, if the SETH is true, $2^k - 2$ is the "correct" base of the exponent for every k.

Along the way, we also consider the complexity of k-Coloring parameterized by the related parameter modular treewidth (mtw). In this case we show that the "correct" running time, under the SETH, is $O^*\left(\binom{k}{\lfloor k/2 \rfloor}^{\text{mtw}}\right)$. If we base our results on a weaker assumption (the ETH), they imply that k-Coloring cannot be solved in time $n^{o(\text{cw})}$, even on instances with $O(\log n)$ colors.

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

Keywords and phrases Clique-width, SETH, Coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.86

1 Introduction

Graph Coloring (from now on simply Coloring) is one of the most intensely studied problems in theoretical computer science. In this classical problem we are given a graph $G = (V, E)$ and an integer k and are asked if we can partition V into k sets inducing edge-less graphs. Coloring is a notoriously hard problem as it remains NP-hard even in very restricted cases (e.g. 4-regular planar graphs [10]) and is essentially completely inapproximable in general [11, 30]. This intractability has motivated the study of the problem in the framework of parameterized complexity, especially with respect to structural graph parameters.[1]

Treewidth is by far the most widely studied among such graph parameters, and Coloring has long been known to be FPT by treewidth. This can be seen by either invoking Courcelle's theorem [4], or by applying a straightforward dynamic programming technique which, for each bag of a tree decomposition of width tw considers all possible k^{tw} colorings. Remarkably, thanks to the work of Lokshtanov, Marx, and Saurabh [23], we know that this basic algorithm is likely to be optimal, or at least that improving it would require a major breakthrough on SAT-solving, as, for any $k \geq 3, \epsilon > 0$, the existence of a $(k-\epsilon)^{\text{tw}}$ algorithm would contradict the Strong Exponential Time Hypothesis of Impagliazzo and Paturi [18, 19]. More recently, these lower bounds were strengthened, as Jaffke and Jansen showed that a $(k-\epsilon)^w$ algorithm would contradict the SETH when w is the graph's vertex edit distance from being a path

[1] In the remainder, we assume that the reader is familiar with the basics of parameterized complexity, such as the class FPT, as given in relevant textbooks [8, 13]

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 86; pp. 86:1–86:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

[20]. The same paper showed that the trivial algorithm can, however, be improved when one considers more restrictive parameters, such as vertex cover, but still not to the point that the base of the exponent becomes a constant. These results thus paint a very clear picture of the complexity of Coloring with respect to treewidth and its restrictions.

One of the drawbacks of treewidth is that it does not cover dense graphs, even if they have a very simple structure. This has led to the introduction of clique-width, which is by now (arguably) the second most well-studied parameter. The complexity of Coloring parameterized by clique-width has also been investigated. Even though Coloring is polynomial-time solvable when clique-width is constant, the best known algorithm for this case runs in time $n^{2^{O(\mathrm{cw})}}$ [21]. Fomin et al. [14] showed that Coloring is not FPT for clique-width (under standard assumptions), and this was recently followed up by Golovach et al. [17] who showed, somewhat devastatingly, that the aforementioned algorithm is likely to be optimal, as an algorithm running in $n^{2^{o(\mathrm{cw})}}$ would contradict the ETH. The problem thus seems to become significantly harder for clique-width, and this has, in part, motivated the study of alternative dense graph parameters, such as split-matching width [28], modular-width [15], and twin cover [16], all of which make Coloring FPT.

Contribution. Although the results mentioned above demonstrate a clear jump in the complexity of Coloring when moving from treewidth to clique-width, we observe that they leave open a significant hole: all the aforementioned hardness results for clique-width ([14, 17]) only apply to the case where k is large (polynomially related to the size of the graph). It is not hard to see that the problem becomes significantly easier if both cw and k are assumed to have moderate values; indeed Coloring is FPT when parameterized by $\mathrm{cw}+k$ [21]. Since the case where k is relatively small is arguably the most interesting scenario for most applications, we are strongly motivated to take a closer look at the complexity of Coloring parameterized by clique-width, in order to obtain a more fine-grained and quantitative estimate of the "price of generality" for this problem *for each fixed value of* k. Our aim is to reach tight bounds that paint a crisper picture of the complexity of the problem than what can be inferred by lower bounds parameterized only by clique-width, in the same way that the results of [23] do for k-Coloring on treewidth.

The main result of this paper is a lower bound which states that for all $k \geq 3, \epsilon > 0$, k-Coloring cannot be solved in time $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$, unless the SETH is false. This result gives a concrete, detailed answer to the question of how much the complexity of 3-Coloring, 4-Coloring, and generally k-Coloring, increases as one moves from treewidth to clique-width. As in the lower bound of [23], this result is established through a reduction from SAT. The main challenge here is that we need to pack a much larger amount of information per unit of width, and in particular that the graph induced by most label sets must be edge-less (otherwise many of the $2^k - 2$ choices we need to encode would be invalid). We work around this difficulty by a delicate use of the rename operation used in clique-width expressions.

Though having $2^k - 2$ in the base of the running time may seem somewhat curious (and certainly less natural than the k^{tw} bounds of [23]), we then go on to prove that this is the "correct" bound by giving a matching algorithm. The algorithm is based on standard DP techniques (including subset convolution [1, 29]), but requires a non-standard trick that "looks ahead" in the decomposition to lower the table size from $(2^k - 1)^{\mathrm{cw}}$ to $(2^k - 2)^{\mathrm{cw}}$. This improves the previously known DP algorithm of [21], which runs in $O^*\left(4^{k \cdot \mathrm{cw}}\right)$.

Beyond these results for clique-width we also consider the closely related parameters modular treewidth and modular pathwidth, which have more recently been considered as

more restricted versions of clique-width [25, 27]. The modular treewidth of a graph G is defined as the treewidth of the graph obtained from G if one collapses each twin class into a single vertex, where two vertices are twins if they have the same neighbors. By slightly altering our results for clique-width we tightly characterize the complexity of k-Coloring for these parameters: the problem is solvable in time $O^*\left(\binom{k}{\lfloor k/2 \rfloor}^{\mathrm{mtw}}\right)$, but not solvable in $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor}-\epsilon\right)^{\mathrm{mpw}}\right)$ under the SETH. Using the same reduction but relaxing the hypothesis to the ETH, we show that k-Coloring cannot be solved in time $n^{o(\mathrm{mpw})}$, and hence neither in time $n^{o(\mathrm{cw})}$ even on instances where $k = O(\log n)$. This can be seen as a strengthening of the lower bound of [14], which applies only to clique-width and uses $\Omega(n)$ colors. Our result is incomparable to the more recent double-exponential bound of [17] as it applies to the more restricted case where the number of colors is logarithmic, and is tight for this case. Indeed, any reduction giving a double-exponential bound, such as the one in [17], must inevitably use more than $\log n$ colors, otherwise it would contradict the aforementioned algorithms.

Non-binary CSPs. We mention as a secondary contribution of this paper a proof that, under the SETH, n-variable CSPs over an alphabet of size B cannot be solved in time $(B-\epsilon)^n$, for any B, ϵ. The interest of such a result is not so much technical (its proof is implicit in previous SETH-based bounds, going back to [23]), as conceptual. Such CSPs provide a convenient starting point for a SETH-based lower bound for *any base* of the exponential and hence allow us to isolate a technical part of such proofs from the main part of the reduction. This explicit statement on the hardness of CPSs has allowed the proofs of this paper to be significantly shortened, and may facilitate the design of other SETH-based hardness proofs.

2 Definitions and Preliminaries

We use standard graph-theoretic notation and assume that the reader is familiar with the basics of parameterized complexity, as well as standard notions such as treewidth [8, 13]. Let us recall the definition of clique-width (see [6, 5] for more details). A labeled graph G has clique-width w if it can be constructed using w labels and the following four basic operations: Introduce(i), for $i \in \{1, \ldots, w\}$, which constructs a single-vertex graph whose vertex has label i; Union(G_1, G_2), which constructs the disjoint union of two labeled graphs of clique-width w; Rename(i, j) which changes the label of all vertices labeled i to j; and Join(i, j) which adds all possible edges between vertices labeled i and vertices labeled j. Computing a graph's clique-width is NP-hard [12], and the best currently known approximation is exponential in clique-width [26]. In this paper, we will often assume that we are given together with a graph G, a clique-width expression constructing G. Since most of our results are negative, this only makes them stronger, as it shows that our lower bounds do not rely on the hardness of computing clique-width. We view a clique-width expression as a rooted binary tree, where the sub-tree rooted in each internal node represents the corresponding sub-graph of G. We use $\mathrm{cw}(G)$ to denote the minimum number of labels needed to construct a clique-width expression of G, and $\mathrm{tw}(G), \mathrm{pw}(G)$ to denote the treewidth and pathwidth of G respectively.

In a graph $G = (V, E)$ we say that $u, v \in V$ are false twins if $N(u) = N(v)$ and true twins if $N[u] = N[v]$, where $N[u] = N(u) \cup \{u\}$ denotes the closed neighborhood of u. We say that u, v are twins if they are true or false twins. We note that in any graph G the partition of vertices into twin classes is always unique, as the property of being twins is an equivalence relation [22]. Let G^t be the graph obtained from G by deleting from each twin class all but a single vertex. We define (following [25]) the modular treewidth of G, denoted $\mathrm{mtw}(G)$, as $\mathrm{tw}(G^t)$, and similarly the modular pathwidth $\mathrm{mpw}(G)$ as $\mathrm{pw}(G^t)$.

▶ **Lemma 1.** *For all* G, $\mathrm{pw}(G) \geq \mathrm{mpw}(G) \geq \mathrm{cw}(G) - 2$ *and* $\mathrm{pw}(G) \geq \mathrm{mpw}(G) \geq \mathrm{mtw}(G)$.

The Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [19] states that there exists a constant $c_3 > 1$ such that 3-SAT on instances with n variables cannot be solved in time c_3^n. If the ETH is true then we can define, for all $q \geq 3$ a constant $c_q > 1$ such that q-SAT, that is, SAT on instances where clauses have maximum size q, cannot be solved in time c_q^n. The Strong Exponential Time Hypothesis (SETH) [18] states that $\lim_{q\to\infty} c_q = 2$, or equivalently that, for each $\epsilon > 0$ there exists a q such that q-SAT cannot be solved in $(2-\epsilon)^n$. We note that sometimes a slightly weaker form of the SETH is used, which states simply that SAT cannot be solved in $(2-\epsilon)^n$ for any $\epsilon > 0$. The two formulations are not currently known to be equivalent. In this paper we use the original, stronger formulation of [18] (see also e.g. [7]) which assumes that c_q tends to 2.

For any $q, B \geq 2$ we define the q-CSP-B problem as follows: we are given a set X of n variables which take values in $\{1, \ldots, B\}$, and a set $\mathcal{C}$ of q-constraints on X. A q-constraint c is defined by an ordered tuple $V(c)$ of q variables of X, and a set $S(c) \subseteq \{1, \ldots, B\}^q$ of satisfying assignments for c. The question is whether there exists an assignment $\sigma : X \to \{1, \ldots, B\}$ which satisfies all constraints $c \in \mathcal{C}$. We say that a constraint $c \in \mathcal{C}$ is satisfied if applying σ to $V(c)$ produces a tuple of assignments that appears in $S(c)$. To simplify presentation, we will assume that in the input the list $S(c)$ of the at most B^q satisfying assignments of each constraint is given explicitly, and that a q-CSP-B instance is allowed to contain constraints on fewer than q variables (as we can add dummy variables to a constraint).

3 SETH and Non-binary CSPs

The SETH states, informally, that as SAT clauses become larger, eventually the best algorithm for SAT is simply to try out all possible assignments to all variables. In this section we show that the same is essentially true for CSP with a larger, non-binary alphabet. The interest in presenting such a result is that very often we seek to prove a SETH-based lower bound showing that a problem does not admit an algorithm running in c^w, for some constant c and width parameter w (such as treewidth, or in our case clique-width). This becomes complicated when we reduce directly from SAT if c is not a power of 2 as one cannot make a one-to-one correspondence between binary SAT variables and "units of width" (in our case labels) in the new instance, which are intended to encode c choices. As a result, essentially all known SETH lower bounds of this form include as part of their construction a group gadget, which maps every t variables of the original SAT instance to p elements of the new problem, for appropriately chosen integers p, t (see e.g. [3, 9, 20, 23]). Such gadgets are, however, often cumbersome to design, because they require a problem-specific trick that expresses a mapping of assignments from a binary to a non-binary domain. We therefore prefer to construct a custom-made CSP with a convenient running time bound, which will later allow us to reduce directly to the problem we are interested in (Coloring on clique-width), in a way that maps exactly one variable to one clique-width label. This will allow our SETH-based bounds to be significantly simplified, as we will no longer have to worry about a discrepancy between the bases of the exponentials.

▶ **Theorem 2.** *For any* $B \geq 2$, $\epsilon > 0$ *we have the following: if the SETH is true, then there exists a* q *such that* n*-variable* q-CSP-B *cannot be solved in time* $O^*\left((B-\epsilon)^n\right)$.

4 SETH-based Lower Bound for Clique-width

In this section we present our main lower bound result stating that k-Coloring cannot be solved in time $O^*\left((2^k-2-\epsilon)^{\text{cw}}\right)$, for any $k \geq 3, \epsilon > 0$, under the SETH. In Section 4.1 we present some basic gadgets that will also be of use in our lower bound for modular pathwidth (Section 5). We then present the main part of the proof in Section 4.2.

4.1 List Coloring and Basic Gadgets

The high-level machinery that we will make use of in our reduction consists of two major points: first, we would like to be able to express implication constraints, that is, constraints of the form "if vertex u received color c_1, then vertex v must receive color c_2"; second, we would like to express disjunction constraints of the form "at least one of the vertices of the set $S \subseteq V$ must take a special color 1". We build this machinery in the following steps: first, we show that we can (almost) equivalently produce an instance of the more general List Coloring problem; then we use the ability to construct lists to make *weak edge gadgets*, which for a given pair of vertices (u_1, u_2) rule out a specific pair of assigned colors; using these weak edges we construct the aforementioned implication gadgets; and finally we are able to implement OR constraints using paths on vertices with appropriate lists.

We give all details for these constructions below. We remark however, for the convenience of the reader, that a high-level understanding of the informal meaning of implication gadgets and OR gadgets (precisely stated in Lemmata 7, 10) is already sufficient to follow the description of the main part of the reduction, given in Section 4.2. See also Figure 1.

List Coloring. To simplify the presentation of our reduction it will be convenient to use a slightly more general problem. In List Coloring, we are given a graph G and a list of possible colors associated with each vertex and are asked if there is a proper coloring such that each vertex uses a color from its list. This problem clearly generalizes k-Coloring, as all lists may be $\{1, \ldots, k\}$. We will make use of a reduction in the opposite direction.

▶ **Lemma 3.** *There is a polynomial-time algorithm which, given an instance of* List Coloring *on a graph G where all lists are subsets of $\{1, \ldots, k\}$, transforms it into an equivalent instance of k-*Coloring *on a graph G'. Furthermore, the algorithm transforms a clique-width expression of G with* cw *labels, to a clique-width expression of G' with* $\text{cw} + k$ *labels. If all twins of G share the same list, the algorithm transforms a modular path decomposition of width* mpw *for G, to a modular path decomposition of G' of width* $\text{mpw} + k$.

Weak Edges and Implications. Normally, the existence of an edge (u, v) in an instance of Coloring forbids the vertices u, v from obtaining the same color, whatever that color may be. We will find it convenient to construct edges that forbid only a specific pair of colors from appearing on u, v, while allowing any other combination of colors to be used on these two vertices. Similar versions of this gadget have appeared before, for example [23, 24].

▶ **Definition 4.** For two vertices u_1, u_2 of a graph G and two colors c_1, c_2 a (c_1, c_2)-*weak edge* from u_1 to u_2 consists of the following:

1. Three new vertices v_1, v_2, v_3 such that $\{u_1, v_1, v_2, v_3, u_2\}$ induces a path in this order, with endpoints u_1, u_2, and v_1, v_2, v_3 having no edges to the rest of G.
2. If $c_1 \neq c_2$ let c' be a color distinct from c_1, c_2. We assign to v_1, v_2, v_3 the lists $\{c_1, c_2\}, \{c_2, c'\}, \{c_2, c'\}$. If $c_1 = c_2$, we assign lists $\{c_1, c'\}, \{c', c''\}, \{c_1, c''\}$ to v_1, v_2, v_3 respectively, where c', c'' are two distinct colors, different from c_1.

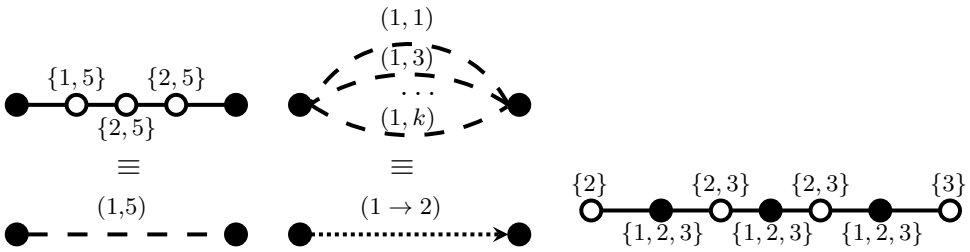

Figure 1 Basic gadgets, where empty vertices are internal and solid vertices are endpoints that will be connected to the rest of the graph. On the left, a weak edge that forbids the combination $(1, 5)$ on its endpoints. In the middle, an implication that forces color 2 on the right if color 1 is used on the left. On the right an OR gadget: one of the solid vertices must take color 1.

▶ **Lemma 5.** *Let G be an instance of* List Coloring *that contains a (c_1, c_2)-weak edge from u_1 to u_2. Then G does not admit a valid coloring that assigns colors (c_1, c_2) to (u_1, u_2). Furthermore, if G' denotes the graph obtained by deleting the internal vertices of the weak edge, any proper list coloring of G' that does not assign c_1 to u_1 or does not assign c_2 to u_2 can be extended to a proper list coloring of G.*

Let us now use the weak edges we have defined above to construct an implication gadget. The intuitive meaning of placing an implication gadget from a vertex u_1 to a vertex u_2 is to impose the constraint that if u_1 is assigned color c_1, then u_2 must be assigned color c_2.

▶ **Definition 6.** For two vertices u_1, u_2 and two colors (c_1, c_2) a $(c_1 \to c_2)$-implication from vertex u_1 to vertex u_2 is constructed as follows: for each color $c' \neq c_2$, we add a (c_1, c')-weak edge from u_1 to u_2.

▶ **Lemma 7.** *Let G be an instance of* List Coloring *that contains a $(c_1 \to c_2)$-implication from u_1 to u_2. Then G does not admit a list coloring that gives color c_1 to u_1 and a color $c' \neq c_2$ to u_2. Furthermore, if G' is the graph obtained from G by deleting the internal vertices of the implication gadget, any coloring of G' that either does not assign c_1 to u_1, or assigns c_2 to u_2 can be extended to a coloring of G.*

▶ **Lemma 8.** *Let G be an instance of* List Coloring*, and G' be the graph obtained from G by replacing every (c_1, c_2)-weak edge or $(c_1 \to c_2)$-implication gadget with endpoints u_1, u_2 with an edge (u_1, u_2) (or simply deleting the internal vertices of the weak edge if (u_1, u_2) already exists). Then* $\mathrm{pw}(G) \leq \mathrm{pw}(G') + 3$.

OR gadgets. We will also make use of a gadget that forces any valid list coloring of a graph to assign a special color 1 to one vertex out of a set of vertices. Invariably, the idea will be that this will be a color that activates some implications, allowing us to propagate information about the coloring between parts of the graph. We recall that a similar version of an OR gadget was also used in [23].

▶ **Definition 9.** An OR gadget on an independent set of vertices S, denoted $\mathrm{OR}(S)$, is constructed as follows: we assign list $\{1, 2, 3\}$ to all vertices of S; we construct a new set S' of internal vertices (that will not be connected to the rest of G), such that $|S'| = |S| + 1$ and $S \cup S'$ induces a path alternating between vertices of S and S'; we assign list $\{2, 3\}$ to all vertices of S', except the two endpoints of the path, which receive lists $\{2\}, \{3\}$, respectively.

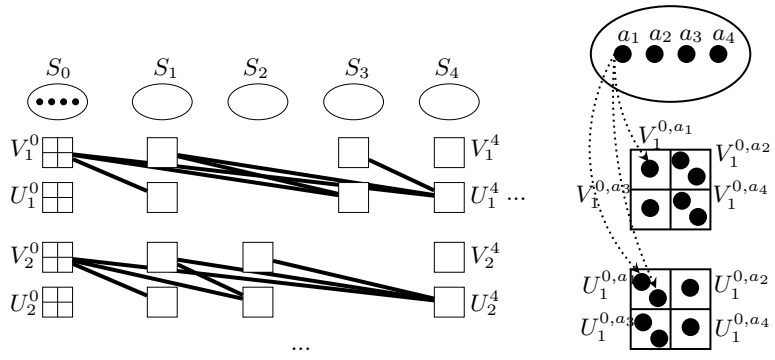

Figure 2 Left: high-level view of the reduction. Rows correspond to variables, columns to constraints. Here, variable x_1 appears in constraints c_0, c_1, c_3, c_4. Right: connections between the OR gadgets $\mathrm{OR}(S_j)$ and the V_i^j, U_i^j sets. Giving color 1 to a_1 represents selecting this assignment. This forces the use of some colors in V_1^{0,a_1} and the complementary set in U_1^{0,a_1}.

▶ **Lemma 10.** *If G is a* List Coloring *instance that contains an OR(S) gadget then G does not admit a list coloring that does not use color* 1 *in any vertex of S. Furthermore, for any vertex $u \in S$, there exists a proper list coloring of the graph induced by the gadget that assigns color* 1 *only to u.*

4.2 Reduction for Clique-width

▶ **Theorem 11.** *For any $k \geq 3, \epsilon > 0$, if there exists an algorithm solving k-coloring in time $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$, where* cw *is the input graph's clique-width, then the SETH is false.*

The proof of Theorem 11 consists of a reduction from a CSP produced by Theorem 2. Before giving details, let us give some intuition. Our new instance will be a graph and a clique-width expression with, roughly, n labels, where n is the number of variables of the CSP instance. The set of colors used in each label will encode the value given to a variable in a satisfying assignment. As a result, with k colors, we will have $2^k - 2$ encodings available, as every label set uses at least one color, but will never use all k colors. To verify that these assignments are correct, we will construct for each constraint an OR gadget which forces the use of color 1 on a vertex representing a particular assignment. This assignment dictates the value of each variable of the constraint, and therefore the set of colors used in some of our label sets. To verify that the assignment is consistent we use implication gadgets that force some auxilliary vertices to receive the complement of the colors dictated by the constraint assignment, and then connect these with the vertices encoding the true assignment. If the assignment used is truly consistent, these edges will end up being properly colored.

Construction. We are given $k \geq 3, \epsilon > 0$. Let $B = 2^k - 2$. Let q be the smallest integer such that n-variable q-CSP-B does not admit an $O^*\left((B - \epsilon)^n\right)$ algorithm. According to Theorem 2, q exists if the SETH is true, and it depends only on B, ϵ. Consider an arbitrary n-variable instance of q-CSP-B, call it ϕ. We use the existence of the supposed $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$ algorithm to obtain an $O^*\left((B - \epsilon)^n\right)$ algorithm that decides ϕ, contradicting the SETH.

We define in some arbitrary way a translation function T which, given a value $v \in \{1, \ldots, B\}$ returns a non-empty proper subset of $\{1, \ldots, k\}$. We make sure that T is defined in such a way that it is one-to-one; this is possible since the number of non-empty proper subsets of $\{1, \ldots, k\}$ is exactly $B = 2^k - 2$.

Let $X = \{x_1, \ldots, x_n\}$ be the set of the n variables of the q-CSP-B instance and $C = \{c_0, \ldots, c_{m-1}\}$ the set of its m constraints. Let $L = 3m(nk+1)$. We now construct our graph, where if we don't specify the list of a vertex it can be assumed to be $\{1, \ldots, k\}$. For each $j \in \{0, \ldots, L-1\}$ we do the following:

1. Let $j' = j \bmod m$ and let S be the set of satisfying assignments of the constraint $c_{j'}$. We construct an independent set of vertices S_j that contains a vertex for every assignment of S. We construct an $\mathrm{OR}(S_j)$ gadget on these vertices.
2. For each x_i which appears in $c_{j'}$ and for each assignment $a \in S$ we do the following:
   a. Let $v \in \{1, \ldots, B\}$ be the value given to x_i by the assignment a. Construct an independent set $V_i^{j,a}$ of $|T(v)|$ vertices and an independent set $U_i^{j,a}$ of $k - |T(v)|$ vertices. Recall that $T(v)$, the translation function, returns a set of size between 1 and $k-1$, so both these sets are non-empty.
   b. For each color $c \in T(v)$ select a distinct vertex in $V_i^{j,a}$ and add a $(1 \to c)$-implication gadget from the vertex that represents the assignment a in S_j to this vertex of $V_i^{j,a}$.
   c. For each color $c \in \{1, \ldots, k\} \setminus T(v)$ select a distinct vertex in $U_i^{j,a}$ and add a $(1 \to c)$-implication gadget from the vertex that represents the assignment a in S_j to this vertex of $U_i^{j,a}$.
   d. Connect all vertices of $U_i^{j,a}$ with all vertices of previously constructed sets $V_i^{l,a'}$, for all $l < j$ and all assignments a'.

This completes the construction, and we call the constructed List Coloring instance $G(\phi)$. The intended meaning is that the sets $V_i^{j,a}$ will use a set of colors that encodes the value of the variable x_i, while the sets $U_i^{j,a}$ will use colors from the complement of this set.

▶ **Lemma 12.** *If ϕ is a satisfiable q-CSP-B instance, then $G(\phi)$ admits a proper list coloring.*

Proof. Suppose that we have a satisfying assignment for ϕ which gives value v_i to variable x_i. The invariant we will maintain is that for all j, a, all vertices of sets $V_i^{j,a}$ will use only colors from $T(v_i)$, while all vertices of sets $U_i^{j,a}$ will use only colors from $\{1, \ldots, k\} \setminus T(v_i)$. As a result, all edges added in step 2d will be properly colored. The rest of the graph will be easy to color if we respect the informal meaning of OR and implication gadgets.

More specifically, for each $\mathrm{OR}(S_j)$ gadget we let $j' = j \bmod m$ and consider the constraint $c_{j'}$. The supposed assignment to ϕ assigns to the variables of the constraint values consistent with a satisfying assignment a of $c_{j'}$. We give color 1 to the corresponding vertex of S_j. We use colors $\{2, 3\}$ to color all remaining vertices of the OR gadget. Note that the OR gadget is connected to the rest of the graph only through implication gadgets activated by color 1. Hence, by Lemma 7 we can remove all non-activated implication gadgets. For the remaining, activated implication gadgets we color their other endpoints, which are found in the sets $V_i^{j,a}$ and $U_i^{j,a}$ with the unique viable color. For every other assignment $a' \neq a$ we color all vertices of $V_i^{j,a'}$ using a color we used in $V_i^{j,a}$, and the vertices of $U_i^{j,a'}$ using a color we used in $U_i^{j,a}$.

The promised invariant is maintained, as the vertices of $V_i^{j,a}$ are forced to receive colors from $T(v_i)$, while vertices of $U_i^{j,a}$ are forced to receive colors from the complementary set. Thus, all edges of step 2d are properly colored, and since we also properly colored the OR gadgets and implication gadgets, we have a proper coloring of the whole graph. ◀

▶ **Lemma 13.** *If $G(\phi)$ admits a proper list coloring, then ϕ is a satisfiable q-CSP-B instance.*

Proof. Suppose we have a list coloring of $G(\phi)$ given by the function $\mathbf{c} : V \to \{1, \ldots, k\}$. For a set $V' \subseteq V$ we will write $\mathbf{c}(V')$ to denote the set of colors used by $\mathbf{c}$ for vertices of V', that is, $\mathbf{c}(V') = \{c \mid \exists u \in V',\ \mathbf{c}(u) = c\ \}$. Let $j \in \{0, \ldots, L-1\}$, $j' = j \bmod m$, and S be the set

of satisfying assignment of the constraint $c_{j'}$, which contains a variable x_i. Consider the set $V_i^j = \cup_{a \in S} V_i^{j,a}$. We define the *candidate assignment* of x_i at index j as $v_i^j := T^{-1}(\mathbf{c}(V_i^j))$. In other words, to obtain the candidate assignment for x_i at index j, we take the union of all colors used in $V_i^{j,a}$, and then translate this set back into a value in $\{1, \ldots, B\}$.

We observe that for all i, j, such that x_i appears in $c_{j'}$, where $j' = j \bmod m$, there exists an assignment a such that $\mathbf{c}(V_i^{j,a}) = \{1, \ldots, k\} \setminus \mathbf{c}(U_i^{j,a})$. To see this, note that by Lemma 10, one of the vertices of the $\mathrm{OR}(S_j)$ gadget must have received color 1, say the vertex that corresponds to assignment a. All the implications incident on this vertex are therefore activated, which means that, if a gives value $v \in \{1, \ldots, B\}$ to x_i, then $\mathbf{c}(V_i^{j,a}) = T(v)$ and $\mathbf{c}(U_i^{j,a}) = \{1, \ldots, k\} \setminus T(v)$ (because of the implications of steps 2b,2c and Lemma 7).

A key observation now is the following: for all $j_2 > j_1$ and for all i such that variable x_i appears in constraints $c_{j_1'}, c_{j_2'}$ with $j_1' = j_1 \bmod m$, $j_2' = j_2 \bmod m$, we have $\mathbf{c}(V_i^{j_1}) \subseteq \mathbf{c}(V_i^{j_2})$. In other words, the set of colors used in V_i^j can only increase as j increases. To see this, suppose that there exists $c \in \mathbf{c}(V_i^{j_1}) \setminus \mathbf{c}(V_i^{j_2})$. As argued in the previous paragraph, there exists an assignment a_2 such that $\mathbf{c}(V_i^{j_2}) \supseteq \mathbf{c}(V_i^{j_2,a_2}) = \{1, \ldots, k\} \setminus \mathbf{c}(U_i^{j_2,a_2})$. Because $c \notin \mathbf{c}(V_i^{j_2,a_2})$ we must have $c \in \mathbf{c}(U_i^{j_2,a_2})$, but because of step 2d, all vertices of $U_i^{j_2,a_2}$ are connected to all of $V_i^{j_1}$. Since $c \in \mathbf{c}(V_i^{j_1})$, this contradicts the correctness of the coloring.

The property established in the previous paragraph implies that for each i there exist at most k distinct candidate assignments v_i^j we can obtain for different values of j, as each assignment is obtained by translating the set of colors used in V_i^j, this set only increases, it always contains at least one color and at most k colors. Let us say that an index j_1 is problematic if, for some $i \in \{1, \ldots, n\}$ we have the following: x_i appears in constraint $c_{j_1'}$, where $j_1' = j_1 \bmod m$; and if j_2 is the minimum index such that $j_2 > j_1$ and x_i appears in constraint $c_{j_2'}$, where $j_2' = j_2 \bmod m$, then $v_i^{j_1} \neq v_i^{j_2}$. In other words, an index is problematic if the candidate assignment it produces for a variable disagrees with the candidate assignment produced for the same variable in the next index that involves this variable. It is not hard to see that there are at most kn problematic indices, because for each variable there are at most k problematic indices. Therefore, since $L = 3m(nk + 1)$, by pigeonhole principle, there exists an interval L' of at least $3m$ consecutive non-problematic indices.

We now obtain an assignment for the original instance as follows: for each variable i we take an index $j \in L'$ such that x_i appears in constraint $c_{j'}$, where $j' = j \bmod m$, and give x_i the candidate value v_i^j from that index. Observe that, by the definition of L' the index we select is irrelevant, as all candidate values are constant throughout the interval L'.

We claim that this is a satisfying assignment. Suppose not, so there exists an unsatisfied constraint $c_{j'}$. Because L' contains $3m$ consecutive indices, there exists three indices $j_1 < j_2 < j_3 \in L'$ such that $j' = j_1 \bmod m = j_2 \bmod m = j_3 \bmod m$. We observe that for all variables x_i appearing in $c_{j'}$ we have given value $v_i^{j_2}$, that is the candidate value obtained at index j_2, since all indices in L' give the same candidate values to all variables.

Now, there exists a vertex in S_{j_2} that received color 1, representing an assignment a. If the assignment we produced is not consistent with a, there exists a variable x_i such that we have given x_i value $v = v_i^{j_2}$, while a gives it value $v' \neq v$. Consider now the set $V_i^{j_2,a}$. Because of the implication gadgets, it uses the colors $T(v') \neq T(v)$. If there exists $c \in T(v) \setminus T(v')$ then $c \in \mathbf{c}(U_i^{j_2,a})$. But $U_i^{j_2,a}$ is connected to all vertices of $V_i^{j_1}$, which, we assumed use all colors of $T(v)$, therefore also color c, contradicting the correctness of the coloring. If on the other hand there exists $c \in T(v') \setminus T(v)$, then since $\mathbf{c}(V_i^{j_3}) = T(v)$, there exists an a' such that $\mathbf{c}(U_i^{j_3,a'}) = \{1, \ldots, k\} \setminus T(v)$. Therefore, $c \in \mathbf{c}(U_i^{j_3,a'})$, while $c \in \mathbf{c}(V_i^{j_2,a})$, and by step 2d these sets are connected, again obtaining a contradiction. We therefore conclude that we must have a consistent satisfying assignment. ◀

▶ **Lemma 14.** *$G(\phi)$ can be constructed in time polynomial in $|\phi|$, and we have* $\mathrm{cw}(G(\phi)) \leq n + O(qk^2B^q) = n + f(\epsilon, k)$ *for some function f.*

Proof. For fixed $k \geq 3, \epsilon > 0$, we have that $B = 2^k - 2$ and q is a constant that only depends on B, ϵ (that is, on k, ϵ). Each constraint of the q-CSP-B instance has at most B^q satisfying assignments. Therefore, it is not hard to see that the whole construction can be performed in polynomial time, if k, ϵ, B, q are constants. For clique-width we use the following labels:

1. n main labels, representing the variables of ϕ.
2. A single junk label. Its informal meaning is that a vertex that receives this label will not be connected to anything else not yet introduced in the graph.
3. $O(B^q)$ constraint work labels.
4. $O(qk^2B^q)$ variable-constraint incidence work labels.

To give a clique-width expression we will describe how to build the graph, following essentially the steps given in the description of the construction by maintaining the following invariant: before starting iteration j, all vertices of the set $\bigcup_{j'<j} V_i^{j',a}$ (where we take the union over all assignments a), have label i, and all other vertices have the junk label.

This invariant is vacuously satisfied before the first iteration, since the graph is empty. Suppose that for some $j \in \{0, \ldots, L-1\}$ the invariant is true. We use the $O(B^q)$ constraint work labels to introduce the vertices of the $\mathrm{OR}(S_j)$ gadget of step 1, giving each vertex a distinct label. We use join operations to construct the internal edges of the OR gadget.

Then, for each variable x_i that appears in the current constraint we do the following: we use $O(k^2B^q)$ of the variable-constraint incidence work labels to introduce the vertices of $V_i^{j,a}$, $U_i^{j,a}$ as well as the implication gadgets connecting these to S_j. Again we use a distinct label for each vertex, but the number of vertices (including internal vertices of the implication gadgets) is $O(k^2B^q)$, so we have sufficiently many labels to use distinct labels for each of the q variables of the constraint. We use join operations to add the edges inside all implication gadgets. Then we use join operations to connect $U_i^{j,a}$ to all vertices $\bigcup_{j'<j} V_i^{j',a}$, for $j' < j$. This is possible, since the invariant states that all the vertices of $\bigcup_{j'<j} V_i^{j',a}$ have the same label. We then rename all the vertices of $U_i^{j,a}$, for all a to the junk label, and do the same also for internal vertices of all implication gadgets. We proceed to the next variable of the same constraint and handle it using its own $O(k^2B^q)$ labels. Once we have handled all variables of the current constraint, we rename all vertices of each $V_i^{j,a}$ to label i for all a. We then rename all vertices of the $\mathrm{OR}(S_j)$ gadget to the junk label and increase j by 1. It is not hard to see that we have maintained the invariant and constructed all edges induced by the vertices introduced in steps up to j, so repeating this process constructs the graph. ◀

5 Modular Pathwidth and ETH

We present a lower bound on the complexity of k-Coloring parameterized by modular pathwidth. Specifically, we show that, under the SETH, no algorithm can solve k-Coloring in $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor} - \epsilon\right)^{\mathrm{mpw}}\right)$. This bound is somewhat weaker than the one in Theorem 11, which is natural since modular pathwidth is more restrictive than clique-width. As we see in Section 6, however, this bound is tight. We complete this section by performing the same reduction with the ETH, rather than the SETH, as a starting point. Under this weaker assumption we prove that k-Coloring does not admit an algorithm running in $n^{o(\mathrm{mpw})}$, even when $k = O(\log n)$, which implies that the problem does not admit an algorithm running in $2^{o(k \cdot \mathrm{mpw})}$. This is tight, and also applies to clique-width (Lemma 1). In this way, our reduction gives an alternative proof that k-Coloring is unlikely to be FPT parameterized by clique-width, even in instances with a logarithmic number of colors.

5.1 SETH-based Lower Bound

▶ **Theorem 15.** *For any $k \geq 3, \epsilon > 0$, if there exists an algorithm solving k-coloring in time $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor} - \epsilon\right)^{\text{mpw}}\right)$, where* mpw *is the graph's modular pathwidth, then the SETH is false.*

As in Theorem 11, we begin our reduction from a q-CSP-B instance, where the alphabet size B is equal to the base of the exponential in our lower bound. The intuition will be that the "important" vertices of the bags in a modular tree decomposition of our graph will correspond to classes of $\lfloor k/2 \rfloor$ true twin vertices. The set of $\lfloor k/2 \rfloor$ colors used to color them will encode the value of one variable of the original instance.

Construction. We are given some $k \geq 3, \epsilon > 0$. Let $B = \binom{k}{\lfloor k/2 \rfloor}$. Let q be the smallest integer such that n-variable q-CSP-B does not admit an $O^*\left((B-\epsilon)^n\right)$ algorithm. Consider an arbitrary n-variable instance of q-CSP-B, call it ϕ. We use the existence of the supposed algorithm to obtain an $O^*\left((B-\epsilon)^n\right)$ algorithm that decides ϕ, contradicting the SETH.

As in Theorem 11 we define a one-to-one translation function T. This time, when T is given as input a value $v \in \{1, \ldots, B\}$, it returns a subset of $\{1, \ldots, k\}$ of cardinality $\lfloor k/2 \rfloor$. Let $X = \{x_1, \ldots, x_n\}$ be the set of the n variables of the q-CSP-B instance and $C = \{c_0, \ldots, c_{m-1}\}$ the set of its m constraints. We construct our graph $G(\phi)$ as follows, where if we don't specify a list for a vertex its list is $\{1, \ldots, k\}$:

1. For each variable $i \in \{1, \ldots, n\}$ we construct a clique V_i on $\lfloor k/2 \rfloor$ vertices.
2. For each $j \in \{0, \ldots, m-1\}$, let S be the set of satisfying assignments of the constraint c_j. We construct an independent set S_j with one vertex for each element of S. We construct an $\text{OR}(S_j)$ gadget on the set S_j.
3. For each $j \in \{0, \ldots, m-1\}$, each satisfying assignment a for the constraint c_j, and each variable x_i appearing in c_j we do the following:
   a. We construct a set $U_i^{j,a}$ of $\lceil k/2 \rceil$ vertices.
   b. Suppose a assigns value v to x_i. For each color $c \in \{1, \ldots, k\} \setminus T(v)$ we select a vertex of $U_i^{j,a}$. We construct a $(1 \to c)$-implication gadget from the vertex representing a in S_j to this vertex of $U_i^{j,a}$.
   c. We connect all vertices of $U_i^{j,a}$ with all vertices of V_i.

▶ **Lemma 16.** *If ϕ is a satisfiable q-CSP-B instance, then $G(\phi)$ admits a proper list coloring.*

▶ **Lemma 17.** *If $G(\phi)$ admits a proper list coloring, then ϕ is a satisfiable q-CSP-B instance.*

▶ **Lemma 18.** *$G(\phi)$ can be constructed in time polynomial in $|\phi|$, and $\text{mpw}(G) \leq n + O(1)$.*

5.2 ETH-based Lower Bound

▶ **Theorem 19.** *If there exists an algorithm that solves k-Coloring on instances with n vertices and $k = O(\log n)$ in time $n^{o(\text{mpw})}$, then the ETH is false. As a result, if there is an algorithm solving k-Coloring in time $2^{o(k \cdot \text{mpw})}$, then the ETH is false.*

6 Algorithms

We present two algorithms establishing that the lower bounds of Sections 4,5 are essentially tight. Though both algorithms are based on standard techniques, we remark that the algorithm for clique-width requires some extra effort to obtain a DP of the promised size.

6.1 Clique-width algorithm

Our algorithm is based on standard DP. Its basic idea is that a partial solution is characterized by the set of colors it uses to color a set of vertices that share the same label. This leads to a DP table of size $(2^k-1)^{\mathrm{cw}}$, by observing that for any non-empty label set, any viable partial solution will use at least one color, hence there are 2^k-1 possible subsets of $\{1,\ldots,k\}$ to consider. To improve this to $(2^k-2)^{\mathrm{cw}}$, which would match the lower bound of Theorem 11 we need a further idea which will allow us to also rule out the set that uses all k colors.

Let t be a node of the binary tree representing the clique-width expression of G, and let V_t^i be the set of vertices that have label $i \in \{1,\ldots,\mathrm{cw}\}$ in the labeled graph G_t produced by the sub-expression rooted at t. We will say that V_t^i is a *live* label set if there exists an edge in G that is incident on a vertex of V_t^i and does not appear in G_t. In other words, a label set is live if there is a join operation that involves its vertices which has not yet appeared in t. The main observation is that live label sets cannot use all k colors in a valid partial solution, since then the subsequent join operation will fail. Non-live label sets, on the other hand, are irrelevant, since if the coloring is already valid for such a set it is guaranteed to remain valid. Our DP algorithm will therefore keep track of the partial colorings only of live label sets, and thus produce a DP table of size $(2^k-2)^{\mathrm{cw}}$. In this sense, our DP algorithm is slightly non-standard, as part of its procedure involves "looking ahead" in the graph to determine if a label set is live or not. What remains is the problem of implementing the DP so that it takes time linear in the table size; this is handled using the techniques introduced in [1, 29].

▶ **Theorem 20.** *There is an algorithm which, given a graph G, an integer k, and a clique-width expression for G with* cw *labels decides if G is k-colorable in time $O^*\left((2^k-2)^{\mathrm{cw}}\right)$.*

6.2 Modular Treewidth Algorithm

For modular treewidth, we remark that k-Coloring for this parameter can be seen as an equivalent version of Multi-Coloring parameterized by treewidth. In Multi-Coloring, each vertex v has a demand $b(v)$, and we are asked to assign $b(v)$ distinct colors to each vertex so that neighboring vertices have disjoint colors (see e.g. [2]). In our context, the vertex representing a class of b true twins corresponds to a vertex with demand b.

▶ **Theorem 21.** *There is an algorithm which, given a graph G, an integer k, and a modular tree decomposition of G of width* mtw*, decides if G is k-colorable in time $O^*\left(\binom{k}{\lfloor k/2\rfloor}^{\mathrm{mtw}}\right)$.*

7 Conclusions – Open Problems

We have given tight bounds for k-Coloring parameterized by clique-width, complementing previously known bounds for treewidth. A natural question is now how robust these bounds are, especially in the context of approximation. Specifically, does there exist a constant factor approximation algorithm for k-Coloring running in $O^*\left((k-\epsilon)^{\mathrm{tw}}\right)$ or $O^*\left((2^k-2-\epsilon)^{\mathrm{cw}}\right)$? Current knowledge cannot even rule out the existence of such algorithms with a small *additive* approximation error and this area is still largely unexplored.

References

1 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2010.

2 Marthe Bonamy, Lukasz Kowalik, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. Tight lower bounds for the complexity of multicoloring. In *ESA*, volume 87 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

3 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *IPEC*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

4 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

5 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.

6 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

7 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 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. In *FOCS*, pages 150–159. IEEE Computer Society, 2011.

10 David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30(3):289–293, 1980.

11 Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998.

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

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

14 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

15 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

16 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011.

17 Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Cliquewidth III: the odd case of graph coloring parameterized by cliquewidth. In *SODA*, pages 262–273. SIAM, 2018.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

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

20 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *CIAC*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017.

21 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003.

22 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

23 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789. SIAM, 2011.

24 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *ICALP*, volume 55 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

25 Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016.

26 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

27 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

28 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. *Algorithmica*, 75(1):218–253, 2016.

29 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

30 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.