Structural Parameterizations of Clique Coloring*

Lars Jaffke[1], Paloma T. Lima[1], and Geevarghese Philip[2]

[1]University of Bergen, Bergen, Norway
{lars.jaffke,paloma.lima}@uib.no
[2]Chennai Mathematical Institute, Chennai, India and UMI ReLaX
gphilip@cmi.ac.in

October 25, 2021

Abstract

A clique coloring of a graph is an assignment of colors to its vertices such that no maximal clique is monochromatic. We initiate the study of structural parameterizations of the Clique Coloring problem which asks whether a given graph has a clique coloring with q colors. For fixed $q \geq 2$, we give an $\mathcal{O}^\star(q^{\mathsf{tw}})$-time algorithm when the input graph is given together with one of its tree decompositions of width tw. We complement this result with a matching lower bound under the Strong Exponential Time Hypothesis. We furthermore show that (when the number of colors is unbounded) Clique Coloring is XP parameterized by clique-width.

1 Introduction

Vertex coloring problems are central in algorithmic graph theory, and appear in many variants. One of these is Clique Coloring, which given a graph G and an integer k asks whether G has a clique coloring with k colors, i.e. whether each vertex of G can be assigned one of k colors such that there is no monochromatic maximal clique. The notion of a clique coloring of a graph was introduced in 1991 by Duffus et al. [17], and it behaves quite differently from the classical notion of a proper coloring, which forbids monochromatic edges. Any proper coloring is a clique coloring, but not vice versa. For instance, a complete graph on n vertices only has a proper coloring with n colors, while it has a clique coloring with two colors. Moreover, proper colorings are closed under taking subgraphs. On the other hand, removing vertices or edges from a graph may introduce new maximal cliques, therefore a clique coloring of a graph is not always a clique coloring of its subgraphs, not even of its induced subgraphs.

Also from a complexity-theoretic perspective, Clique Coloring behaves very differently from Graph Coloring. Most notably, while it is easy to decide whether a graph has a proper coloring with two colors, Bacsó et al. [2] showed that it is already coNP-hard to decide if a given coloring with two colors is a clique coloring. Marx [29] later proved Clique Coloring to be Σ_2^p-complete for every fixed number of (at least two) colors.

*L. J. is supported by the Trond Mohn Foundation (TMS). The work was partially done while L. J. and P. T. L. were visiting Chennai Mathematical Institute.

On the positive side, Cochefert and Kratsch [10] showed that the Clique Coloring problem can be solved in $\mathcal{O}^\star(2^n)$ time,[1] and the problem has been shown to be polynomial-time solvable on several graph classes. Mohar and Škrekovski [30] showed that all planar graphs are 3-clique colorable, and Kratochvíl and Tuza gave an algorithm that decides whether a given planar graph is 2-clique colorable [26]. For several graph classes it has been shown that all their members except odd cycles on at least five vertices (which require three colors) are 2-clique colorable [2, 3, 6, 7, 15, 24, 31, 34]. Therefore, on these classes Clique Coloring is polynomial-time solvable. Duffus et al. [17] even conjectured in 1991 that perfect graphs are 3-clique colorable, which was supported by many subclasses of perfect graphs being shown to be 2- or 3-clique colorable [1, 2, 9, 15, 17, 30, 31]. However, in 2016, Charbit et al. [8] showed that there are perfect graphs whose clique colorings require an unbounded number of colors.

In this work, we consider Clique Coloring from the viewpoint of parameterized algorithms and complexity [14, 16]. In particular, we consider structural parameterizations of Clique Coloring by two of the most commonly used decomposition-based width measures of graphs, namely *treewidth* and *clique-width*. Informally speaking, the treewidth of a graph G measures how close G is to being a forest. On dense graphs, the treewidth is unbounded, and clique-width can be viewed as an extension of treewidth that remains bounded on several simply structured classes of dense graphs.

Our first main result is that q-Clique Coloring parameterized by treewidth is fixed-parameter tractable. More precisely: we show that for any fixed $q \geq 2$, q-Clique Coloring (asking for a clique coloring with q colors) can be solved in time $\mathcal{O}^\star(q^{\mathsf{tw}})$, where tw denotes the width of a given tree decomposition of the input graph. We also show that this running time is likely the best possible in this parameterization; we prove that under the Strong Exponential Time Hypothesis (SETH), for any $q \geq 2$, there is no $\epsilon > 0$ such that q-Clique Coloring can be solved in time $\mathcal{O}^\star((q-\epsilon)^{\mathsf{tw}})$. In fact, we rule out $\mathcal{O}^\star((q-\epsilon)^t)$-time algorithms for a much smaller class of graphs than those of treewidth t, namely: graphs that have both *pathwidth* and *feedback vertex set number* simultaneously bounded by t.

Our second main result is an XP algorithm for Clique Coloring with clique-width as the parameter. The algorithm runs in time $n^{f(w)}$, when the input n-vertex graph is given with a clique-width w-expression and $f(w) = 2^{2^{\mathcal{O}(w)}}$. The double-exponential dependence on w in the degree of the polynomial stems from the notorious property of clique colorings which we mentioned above; namely, that taking induced subgraphs does not necessarily preserve clique colorings. This results in a large amount of information that needs to be carried along as the algorithm progresses.

This algorithm raises two questions. First, if Clique Coloring is FPT parameterized by clique-width even if k is a priori unbounded. Second, if the triple exponential dependence on w can be avoided under for instance the Exponential Time Hypothesis (ETH), also in the case when k is fixed. Intuitively, a positive answer to the first question only seems feasible via a proof that all graphs of clique-width w can be clique colored with at most some $g(w)$ colors, for some function g. However, the current literature appears to be far from providing such a result. On the other hand, hardness proofs for Graph Coloring parameterized by clique-width [18, 19] rely on the fact that cliques require many colors while keeping the clique-width small; since cliques can be clique colored with two colors, these tricks are of no use in the setting of Clique Coloring. For the second (possibly more tangible) question, one could search for an algorithm for 2-Clique Coloring running in time $2^{2^{2^{o(w)}}} \cdot n^{\mathcal{O}(1)}$, or rule out the existence of such an algorithm under ETH.

The paper is organized as follows. In Section 2, we give an introduction to the basic concepts

[1] The $\mathcal{O}^\star$-notation suppresses polynomial factors in the input size, i.e. for inputs of size n, we have that $\mathcal{O}^\star(f(n)) = \mathcal{O}(f(n) \cdot n^{\mathcal{O}(1)})$.

that are important in this work; in Section 3 we give the results on q-Clique Coloring parameterized by treewidth, and in Section 4 we give the algorithm for Clique Coloring parameterized by clique-width.

2 Preliminaries

Graphs. All graphs considered here are simple and finite. For a graph G we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G, respectively. For an edge $e = uv \in E(G)$, we call u and v the *endpoints* of e and we write $u \in e$ and $v \in e$.

For two graphs G and H, we say that G is a *subgraph* of H, written $G \subseteq H$, if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$. For a set of vertices $S \subseteq V(G)$, the *subgraph of G induced by S* is $G[S] := (S, \{uv \in E(G) \mid u, v \in S\})$; and we let $G - S := G[V(G) \setminus S]$.

For a graph H, we say that a graph G is *H-free* if G does not contain H as an induced subgraph. For a set of graphs $\mathcal{H}$, we say that G is *$\mathcal{H}$-free* if G is H-free for all $H \in \mathcal{H}$.

For a graph G and a vertex $v \in V(G)$, the set of its *neighbors* is $N_G(v) := \{u \in V(G) \mid uv \in E(G)\}$. Two vertices $u, v \in V(G)$ are called *false twins* if $N_G(u) = N_G(v)$. We say that a vertex v is *complete* to a set $X \subseteq V(G)$ if $X \subseteq N_G(v)$. The *degree* of v is $\deg_G(v) := |N_G(v)|$. The *closed neighborhood* of v is $N_G[v] := \{v\} \cup N_G(v)$. For a set $X \subseteq V(G)$, we let $N_G(X) := \bigcup_{v \in X} N_G(v) \setminus X$ and $N_G[X] := X \cup N_G(X)$. In all these cases, we may drop G as a subscript if it is clear from the context. A graph is called *subcubic* if all its vertices have degree at most three.

A graph G is *connected* if for all 2-partitions (X, Y) of $V(G)$ with $X \neq \emptyset$ and $Y \neq \emptyset$, there is a pair $x \in X$, $y \in Y$ such that $xy \in E(G)$. A *connected component* of a graph is a maximal connected subgraph. A connected graph is called a *cycle* if all its vertices have degree two. A graph that does not contain a cycle as a subgraph is called a *forest* and a connected forest is a *tree*. In a tree T, the vertices of degree one are called the *leaves* of T, denoted by $\mathrm{L}(T)$, and the vertices in $V(T) \setminus \mathrm{L}(T)$ are the *internal vertices* of T. A tree of maximum degree two is a *path* and the leaves of a path are called its *endpoints*. A tree T is called a *caterpillar* if it contains a path $P \subseteq T$ such that all vertices in $V(T) \setminus V(P)$ are adjacent to a vertex in P. A forest is called a *linear forest* if all its components are paths and a *caterpillar forest* if all its components are caterpillars.

A tree T is called *rooted*, if there is a distinguished vertex $r \in V(T)$, called the *root* of T, inducing an ancestral relation on $V(T)$: for a vertex $v \in V(T)$, if $v \neq r$, the neighbor of v on the path from v to r is called the *parent* of v, and all other neighbors of v are called its *children*. For a vertex $v \in V(T) \setminus \{r\}$ with parent p, the *subtree rooted at v*, denoted by T_v, is the subgraph of T induced by all vertices that are in the same connected component of $(V(T), E(T) \setminus \{vp\})$ as v. We define $T_r := T$.

A set of vertices $S \subseteq V(G)$ of a graph G is called an *independent set* if $E(G[S]) = \emptyset$. A set of vertices $S \subseteq V(G)$ is a *vertex cover* in G if $V(G) \setminus S$ is an independent set in G. A graph G is called *complete* if $E(G) = \{uv \mid u, v \in V(G)\}$. A set of vertices $S \subseteq V(G)$ is a *clique* in $G[S]$ is complete. A complete graph on three vertices is called a *triangle*.

A graph G is called *bipartite* if its vertex set can be partitioned into two nonempty independent sets, which we will refer to as a *bipartition* of G.

Notation for Equivalence Relations. Let Ω be a set and $\sim$ an equivalence relation over Ω. For an element $x \in \Omega$ the *equivalence class of x*, denoted by $[x]$, is the set $\{y \in \Omega \mid x \sim y\}$. We denote the set of all equivalence classes of $\sim$ by $\Omega/\sim$.

Parameterized Complexity. We give the basic definitions of parameterized complexity that are relevant to this work and refer to [14, 16] for details. Let Σ be an alphabet. A *parameterized problem* is a set $\Pi \subseteq \Sigma^* \times \mathbb{N}$, the second component being the *parameter* which usually expresses a structural measure of the input. A parameterized problem Π is said to be *fixed-parameter tractable*, or in the complexity class FPT, if there is an algorithm that for any $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether or not $(x, k) \in \Pi$, and runs in time $f(k) \cdot |x|^c$ for some computable function $f \colon \mathbb{N} \to \mathbb{N}$ and constant c. We say that a parameterized problem is in the complexity class XP, if there is an algorithm that for each $(x, k) \in \Sigma^* \times \mathbb{N}$ correctly decides whether or not $(x, k) \in \Pi$, and runs in time $f(k) \cdot |x|^{g(k)}$, for some computable functions f and g.

The concept analogous to NP-hardness in parameterized complexity is that of $\mathsf{W}[1]$-hardness, whose formal definition we omit. The basic assumption is that $\mathsf{FPT} \neq \mathsf{W}[1]$, under which no $\mathsf{W}[1]$-hard problem admits an FPT-algorithm. For more details, see [14, 16].

Strong Exponential Time Hypothesis. In 2001, Impagliazzo et al. [20, 21] conjectured that a brute force algorithm to solve the q-SAT problem for every fixed q which given a CNF-formula with clauses of size at most q, asks whether it has a satisfying assignment, is 'essentially optimal.' This conjecture is called the *Strong Exponential Time Hypothesis*, and can be formally stated as follows. (For a survey of conditional lower bounds based on SETH and related conjectures, see [35].)

Conjecture (SETH, Impagliazzo et al. [20, 21]). For every $\epsilon > 0$, there is a $q \in \mathbb{N}$ such that q-SAT on n variables cannot be solved in time $\mathcal{O}^\star((2 - \epsilon)^n)$.

2.1 Treewidth

We now define the treewidth and pathwidth of a graph, and later the notion of a nice tree decomposition that we will use later in this work.

Definition 2.1 (Treewidth, Pathwidth). Let G be a graph. A *tree decomposition* of G is a pair $(T, \mathcal{B})$ of a tree T and an indexed family of vertex subsets $\mathcal{B} = \{B_t \subseteq V(G)\}_{t \in V(T)}$, called *bags*, satisfying the following properties.

(i) $\bigcup_{t \in V(T)} B_t = V(G)$.

(ii) For each $uv \in E(G)$ there exists some $t \in V(T)$ such that $\{u, v\} \subseteq B_t$.

(iii) For each $v \in V(G)$, let $U_v := \{t \in V(T) \mid v \in B_t\}$ be the nodes in T whose bags contain v. Then, $T[U_v]$ is connected.

The *width* of $(T, \mathcal{B})$ is $\max_{t \in V(T)} |B_t| - 1$, and the *treewidth* of a graph is the minimum width over all its tree decompositions. If T is a path, then $(T, \mathcal{B})$ is called a *path decomposition*, and the *pathwidth* of a graph is the minimum width over all its path decompositions.

Let G be a graph with tree decomposition $(T, \mathcal{B})$, and assume that T is rooted in some node $r \in V(T)$. Then, for each node $t \in V(T)$, we let V_t be the set of vertices of G appearing in bags of the subtree of T rooted at t, i.e. $V_t := \bigcup_{s \in V(T_t)} B_s$. We let $G_t := G[V_t]$. The following notion of a *nice* tree decomposition allows for streamlining the description of dynamic programming algorithms over tree decompositions.

Definition 2.2 (Nice Tree Decomposition). Let G be a graph and $(T, \mathcal{B})$ a tree decomposition of G. Then, $(T, \mathcal{B})$ is called a *nice tree decomposition*, if T is rooted and each node is of one of the following types.

Leaf. A node $t \in V(T)$ is a *leaf node*, if t is a leaf of T and $B_t = \emptyset$.

Introduce. A node $t \in V(T)$ is an *introduce node* if it has precisely one child s, and there is a unique vertex $v \in V(G) \setminus B_s$ such that $B_t = B_s \cup \{v\}$. In this case we say that v *is introduced at* t.

Forget. A node $t \in V(T)$ is a *forget node*, if it has precisely one child s, and there is a unique vertex $v \in B_s$ such that $B_t = B_s \setminus \{v\}$. In this case we say that v *is forgotten at* t.

Join. A node $t \in V(T)$ is a *join node*, if it has precisely two children s_1 and s_2, and $B_t = B_{s_1} = B_{s_2}$.

It is known that any tree decomposition of a graph can be transformed in linear time into a nice tree decomposition of the same width, with a relatively small number of bags.

Lemma 2.3 (Kloks [25]). *Let G be a graph on n vertices, and let k be a positive integer. Any width-k tree decomposition $(T, \mathcal{B})$ of G can be transformed in time $\mathcal{O}(k \cdot |V(T)|)$ into a nice tree decomposition $(T', \mathcal{B}')$ of width k such that $|V(T')| = \mathcal{O}(k \cdot n)$.*

2.2 Clique-Width, Branch Decompositions, and Module-Width

We first define clique-width, introduced by Courcelle, Engelfriet, and Rozenberg [11], and then the equivalent measure of *module-width* that we will use in our algorithm. We keep the definition of clique-width slightly informal and refer to [11, 12] for more details.

Let G be a graph. The *clique-width* of G, denoted by $\mathsf{cw}(G)$, is the minimum number of labels $\{1, \ldots, k\}$ needed to obtain G using the following four operations:

(i) Create a new graph consisting of a single vertex labeled i.

(ii) Take the disjoint union of two labeled graphs G_1 and G_2.

(iii) Add all edges between pairs of vertices of label i and label j.

(iv) Relabel every vertex labeled i to label j.

We now turn to the definition of module-width which is based on the notion of a rooted branch decomposition.

Definition 2.4 (Branch decomposition). Let G be a graph. A *branch decomposition* of G is a pair $(T, \mathcal{L})$ of a subcubic tree T and a bijection $\mathcal{L}\colon V(G) \to \mathrm{L}(T)$. If T is a caterpillar, then $(T, \mathcal{L})$ is called a *linear branch decomposition.* If T is rooted, then we call $(T, \mathcal{L})$ a *rooted branch decomposition.* In this case, for $t \in V(T)$, we define $V_t \coloneqq \{v \in V(G) \mid \mathcal{L}(v) \in \mathrm{L}(T_t)\}$, $\overline{V_t} \coloneqq V(G) \setminus V_t$, and $G_t \coloneqq G[V_t]$.

Module-width is attributed to Rao [32, 33]. On a high level, the module-width of a rooted branch decomposition bounds, at each of its nodes t, the maximum number of subsets of $\overline{V_t}$ that make up the intersection of $\overline{V_t}$ with the neighborhood of some vertex in V_t.

Definition 2.5 (Module-width). Let G be a graph, and $(T, \mathcal{L})$ be a rooted branch decomposition of G. For each $t \in V(T)$, let $\sim_t$ be the equivalence relation on V_t defined as follows:

$$\forall u, v \in V_t\colon u \sim_t v \Leftrightarrow N_G(u) \cap \overline{V_t} = N_G(v) \cap \overline{V_t}$$

The *module-width* of $(T, \mathcal{L})$ is $\mathsf{mw}(T, \mathcal{L}) \coloneqq \max_{t \in V(T)} |V_t / {\sim_t}|$. The *module-width of* G, denoted by $\mathsf{mw}(G)$, is the minimum module width over all rooted branch decompositions of G.

We introduce some notation. For a node $t \in V(T)$ and a set $S \subseteq V(G_t)$, we let $\mathsf{eqc}_t(S)$ be the set of all equivalence classes of $\sim_t$ which have a nonempty intersection with S, and $\overline{\mathsf{eqc}}_t(S)$ be the remaining equivalence classes of $\sim_t$. Formally, $\mathsf{eqc}_t(S) := \{Q \in V_t/{\sim_t} \mid Q \cap S \neq \emptyset\}$ and $\overline{\mathsf{eqc}}_t(S) := V_t/{\sim_t} \setminus \mathsf{eqc}_t(S)$. Moreover, for a set of equivalence classes $\mathcal{Q} \subseteq V_t/{\sim_t}$, we let $V(\mathcal{Q}) := \bigcup_{Q \in \mathcal{Q}} Q$.

Let $(T, \mathcal{L})$ be a rooted branch decomposition of a graph G and let $t \in V(T)$ be a node with children r and s. We now describe an operator associated with t that tells us how the graph G_t is formed from its subgraphs G_r and G_s, and how the equivalence classes of $\sim_t$ are formed from the equivalence classes of $\sim_r$ and $\sim_s$. Concretely, we associate with t a bipartite graph H_t on bipartition $(V_r/{\sim_r}, V_s/{\sim_s})$ such that:

(i) $E(G_t) = E(G_r) \cup E(G_s) \cup F$, where $F = \{uv \mid u \in V_r, v \in V_s, \{[u], [v]\} \in E(H_t)\}$, and

(ii) there is a partition $\mathcal{P} = \{P_1, \ldots, P_h\}$ of $V(H_t)$ such that $V_t/{\sim_t} = \{Q_1, \ldots, Q_h\}$, where for $1 \leq i \leq h$, $Q_i = \bigcup_{Q \in P_i} Q$. For each $1 \leq i \leq h$, we call P_i the *bubble* of the resulting equivalence class $\bigcup_{Q \in P_i} Q$ of $\sim_t$.

As auxiliary structures, for $p \in \{r, s\}$, we let $\eta_p \colon V_p/{\sim_p} \to V_t/{\sim_t}$ be the map such that for all $Q_p \in V_p/{\sim_p}$, $Q_p \subseteq \eta_p(Q_p)$, i.e. $\eta_p(Q_p)$ is the equivalence class of $\sim_t$ whose bubble contains Q_p. We call (H_t, η_r, η_s) the *operator* of t.

Theorem 2.6 (Rao, Thm. 6.6 in [32]). *For any graph G, $\mathsf{mw}(G) \leq \mathsf{cw}(G) \leq 2 \cdot \mathsf{mw}(G)$, and given a decomposition of bounded clique-width, a decomposition of bounded module-width, and vice versa, can be constructed in time $\mathcal{O}(n^2)$, where $n = |V(G)|$.*

2.3 Colorings

Let G be a graph. An ordered partition $\mathcal{C} = (C_1, \ldots, C_k)$ of $V(G)$ is called a *coloring* of G with k colors, or a *k-coloring* of G. (Observe that for $i \in \{1, \ldots, k\}$, C_i may be empty.) For $i \in \{1, \ldots, k\}$, we call C_i the *color class* i, and say that the vertices in C_i *have color* i. $\mathcal{C}$ is called *proper* if for all $i \in \{1, \ldots, k\}$, C_i is an independent set in G.

A coloring $\mathcal{C} = (C_1, \ldots, C_k)$ of a graph G is called a *clique coloring (with k colors)* if there is no monochromatic maximal clique, i.e. no maximal clique X in G such that $X \subseteq C_i$ for some i. In this work, we study the following computational problems.

Clique Coloring

Input:	Graph G, integer k
Question:	Does G have a clique coloring with k colors?

q-Clique Coloring for $q \geq 2$

Input:	Graph G
Question:	Does G have a clique coloring with q colors?

The q-Coloring and q-List Coloring problems also make an appearance. In the former, we are given a graph G and the question is whether G has a proper coloring with q colors. In the latter, we are additionally given a list $L(v) \subseteq \{1, \ldots, q\}$ for each vertex $v \in V(G)$, and additionally require the color of each vertex to be from its list.

Whenever convenient, we alternatively denote a coloring of a graph with k colors as a map $\phi \colon V(G) \to \{1, \ldots, k\}$. In this case, a restriction of ϕ to S is the map $\phi|_S \colon S \to \{1, \ldots, k\}$ with $\phi|_S(v) = \phi(v)$ for all $v \in S$. For any $T \subseteq V(G)$ with $S \subseteq T$, we say that $\phi|_T$ extends $\phi|_S$.

3 Parameterized by Treewidth

In this section, we consider the q-Clique Coloring problem, for fixed $q \geq 2$, parameterized by treewidth. First, in Section 3.1, we show that if we are given a tree decomposition of width tw of the input graph, then q-Clique Coloring can be solved in time $\mathcal{O}^\star(q^{\mathsf{tw}})$. After that, in Section 3.2, we show that this is tight according to SETH, by providing one reduction ruling out $\mathcal{O}^\star((2-\epsilon)^{\mathsf{tw}})$-time algorithms for 2-Clique Coloring and another one ruling out $\mathcal{O}^\star((q-\epsilon)^{\mathsf{tw}})$-time algorithms for q-Clique Coloring when $q \geq 3$.

3.1 Algorithm

The algorithm is bottom-up dynamic programming along a nice tree decomposition $(T, \mathcal{B})$ of the input graph G. At each bag B_t, we enumerate all colorings of $G[B_t]$ and verify for each such coloring if it can be extended to G_t such that there are no monochromatic maximal cliques that use a vertex from $V_t \setminus B_t$. Necessarily, we have to allow monochromatic maximal cliques S that are contained inside $G[B_t]$, since further up in the tree decomposition, there may be a vertex v that is complete to S. Therefore, all vertices in S may receive the same color, as long as v (or another such vertex) receives a different color. If on the other hand a monochromatic maximal clique has a vertex that has already been 'forgotten' at or below t, i.e. it is contained in $V_t \setminus B_t$, then this vertex has no neighbors in $V(G) \setminus V_t$; therefore, no vertex from $V(G) \setminus V_t$ can 'fix' this monochromatic maximal clique, and we can disregard the coloring at hand.

As a subroutine, we will have to be able to check at each bag B_t, if some subset $S \subseteq B_t$ contains a maximal clique in G_t. Doing this by brute force would add a *multiplicative* factor of roughly $2^{\mathsf{tw}} \cdot n$ to the runtime which we cannot afford. To avoid this increase in the runtime, we use fast subset convolution[2] to build an oracle $\mathbb{O}_t$ that, once constructed, can tell us in constant time whether or not any subset $S \subseteq B_t$ contains a maximal clique in G_t, for each node t. We give a dynamic programming algorithm that constructs such oracles for all nodes in the tree decomposition, to ensure that we can maintain a runtime that is linear in n. Since it suffices to construct this oracle once per node, this will infer only an *additive* factor of $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot n$ to the runtime, which does not increase the worst-case complexity for any $q \geq 2$.

Proposition 3.1. *Let G be a graph and $(T, \mathcal{B})$ a nice tree decomposition of G of width tw. There is an algorithm that constructs a family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ in time $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot |V(T)|$ that, once constructed, has the following property. For every $t \in V(T)$ and $S \subseteq B_t$, $\mathbb{O}_t$ answers in constant time whether or not S contains a maximal clique in G_t.*

Proof. For each $t \in V(T)$, Let $f_t \colon 2^{B_t} \to \{0,1\}$ be the function defined as follows. For all $S \subseteq B_t$, we let

$$f_t(S) := \begin{cases} 1, & \text{if } S \text{ contains a maximal clique in } G_t, \\ 0, & \text{otherwise.} \end{cases}$$

To prove the statement, we have to show how to compute all values of f_t, for all $t \in V(T)$, within the claimed time bound.

As a first step, we show how to compute a family of functions $\{g_t \colon 2^{B_t} \to \{0,1\}\}_{t \in V(T)}$ such that for all $t \in V(T)$ and all $S \subseteq B_t$, $g_t(S) = 1$ if and only if S *is* a maximal clique in G_t. We do this by bottom-up dynamic programming, and now describe how to compute the function g_t assuming that the functions at the children of t, if any, have been computed.

[2] Similar ideas have been used by Cochefert and Kratsch [10] to give an $\mathcal{O}^\star(2^n)$-time algorithm for Clique Coloring.

Leaf Node. If $t \in V(T)$ is a leaf node, then $B_t = \emptyset$, and there is nothing to compute.

Introduce Node. Suppose $t \in V(T)$ is an introduce node with child s and let v be the vertex introduced at t. Let $S \subseteq B_t$. There are two cases we have to consider, first when $v \notin S$ and second when $v \in S$. If $v \notin S$, then S is a maximal clique in G_t if and only if S is a maximal clique in G_s and v is *not* complete to S. If $v \in S$, then any clique containing S must be fully contained in B_t, since v has no neighbors in $V_t \setminus B_t$. To summarize, we set:

$$g_t(S) \coloneqq \begin{cases} 1, & \text{if either } v \notin S, g_s(S) = 1, \text{ and } S \not\subseteq N(v) \\ & \quad \text{or } v \in S \text{ and } S \text{ is a maximal clique in } G[B_t] \\ 0, & \text{otherwise} \end{cases}$$

Forget Node. If $t \in V(T)$ is a forget node with child s, then we have that $G_t = G_s$. Therefore, for each $S \subseteq B_t$, it suffices to set $g_t(S) \coloneqq g_s(S)$.

Join Node. Suppose $t \in V(T)$ is a join node with children s_1 and s_2. Then, for each $S \subseteq B_t$, we have that S is a maximal clique in G_t if and only if it is both a maximal clique in G_{s_1} and in G_{s_2}. Therefore, we let $g_t(S) \coloneqq g_{s_1}(S) \cdot g_{s_2}(S)$.

Computing an entry of a function at an introduce node takes time at most $\mathsf{tw}^{\mathcal{O}(1)}$, and for a forget or join node it can be done in time $\mathcal{O}(1)$. Therefore, the family $\{g_t\}_{t \in V(T)}$ can be computed in time $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot |V(T)|$.

For the remainder of the proof, recall that for a set Ω, and two functions α and β defined on 2^Ω, their *subset convolution* $\circledast$ is defined as: for all $S \in 2^\Omega$, $(\alpha \circledast \beta)(S) = \sum_{T \subseteq S} \alpha(T)\beta(S \setminus T)$. Fix some $t \in V(T)$. We define a constant function $c_t \colon 2^{B_t} \to \{1\}$, meaning that $c_t(S) = 1$ for all $S \subseteq B_t$, and construct a function $h_t \coloneqq g_t \circledast c_t$. Using the algorithm of Björklund et al. [4], all values of h_t can be computed in time $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)}$. By construction, each set $S \subseteq B_t$ contains $h_t(S)$ maximal cliques in G_t. We therefore obtain f_t as:

$$\forall S \subseteq B_t \colon f_t(S) \coloneqq \begin{cases} 1, & \text{if } h_t(S) \geq 1, \\ 0, & \text{otherwise,} \end{cases}$$

Computing the family of functions $\{f_t\}_{t \in V(T)}$ and therefore the family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ this way can be done within an additional runtime of $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot |V(T)|$. ◻

We are now ready to give the algorithm. We assume that we are given a width-tw tree decomposition of the input graph whose tree has $\mathcal{O}(\mathsf{tw} \cdot n)$ nodes. This requirement on the number of nodes is standard, see e.g. [14].

Theorem 3.2. *For any fixed $q \geq 2$, there is an algorithm that given an n-vertex graph G and a tree decomposition of G of width tw which has $\mathcal{O}(\mathsf{tw} \cdot n)$ nodes, decides whether G has a clique coloring with q colors in time $\mathcal{O}(q^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot n)$, and constructs one such coloring, if it exists.*

Proof. First, we transform the given tree decomposition of G into a nice tree deocmposition $(T, \mathcal{B})$. This can be done in $\mathcal{O}(\mathsf{tw}^2 \cdot n)$ time by Lemma 2.3. We may assume that the bags at leaf nodes are empty, and that T is rooted in some node $\mathfrak{r} \in V(T)$, and $B_{\mathfrak{r}} = \emptyset$.

We do standard bottom-up dynamic programming along T. Let $t \in V(T)$. A partial solution is a q-coloring of G_t that satisfies one additional property. Suppose that in some coloring of G_t, there is a monochromatic maximal clique X in G_t that has some vertex $v \in V_t \setminus B_t$. Then, v has no neighbors in $V(G) \setminus V_t$, therefore X is also a maximal clique in G. This means that the present coloring cannot be extended to a coloring in which X becomes non-maximal, and therefore we can disregard it.

In light of this, we define the table entries as follows. For each $t \in V(T)$ and function $\gamma_t \colon B_t \to \{1, \ldots, q\}$, we let $\mathsf{tab}[t, \gamma_t] = 1$ if and only if there is a q-coloring γ of G_t such that

- $\gamma|_{B_t} = \gamma_t$, and
- for each maximal clique X in G_t that is monochromatic under γ, $X \subseteq B_t$.

Since $B_{\mathfrak{r}} = \emptyset$, we can immediately observe that the solution to the instance can be read off the table entries at the root node, once computed. Throughout the following we denote by $\gamma_\emptyset$ the q-coloring defined on an empty domain.

Observation 3.2.1. G has a clique coloring with q colors if and only if $\mathsf{tab}[\mathfrak{r}, \gamma_\emptyset] = 1$.

As a preprocessing step, we compute the family of oracles $\{\mathbb{O}_t\}_{t \in V(T)}$ from Proposition 3.1 which will be used at forget nodes. We now show how to compute the table entries for the different types of nodes, assuming that the table entries at the children, if any, have previously been computed.

Leaf Node. If t is a leaf node, then $B_t = \emptyset$ and we only have to consider the empty coloring. We set $\mathsf{tab}[t, \gamma_\emptyset] = 1$.

Introduce Node. Let $t \in V(T)$ be an introduce node with child s, and let v be the vertex introduced at t, i.e. we have that $B_t = B_s \cup \{v\}$. Since $V_t \setminus B_t = (V_t \setminus \{v\}) \setminus (B_t \setminus \{v\}) = V_s \setminus B_s$, and since v has no neighbors in $V_t \setminus B_t$ by the properties of a tree decomposition, it is clear that a coloring of G_t has a monochromatic maximal clique with a vertex in $V_t \setminus B_t$ if and only if its restriction to V_s is a coloring of G_s that has a monochromatic maximal clique with a vertex in $V_s \setminus B_s$. Therefore, for each $\gamma_t \colon B_t \to \{1, \ldots, q\}$, we simply let $\mathsf{tab}[t, \gamma_t] = 1$ if and only if $\mathsf{tab}[s, \gamma_t|_{B_s}] = 1$.

Join Node. Let $t \in V(T)$ be a join node with children s_1 and s_2 and recall that $B_t = B_{s_1} = B_{s_2}$. In this case, for any $\gamma_t \colon B_t \to \{1, \ldots, q\}$, G_t has a q-coloring γ with $\gamma|_{B_t} = \gamma_t$ without a monochromatic maximal clique in $V_t \setminus B_t$ if and only if the analogous condition holds for both G_{s_1} and G_{s_2}. Therefore, for all such γ_t, we let $\mathsf{tab}[t, \gamma_t] = 1$ if and only if $\mathsf{tab}[s_1, \gamma_t] = \mathsf{tab}[s_2, \gamma_t] = 1$.

Forget Node. Let $t \in V(T)$ be a forget node with child s and let v be the vertex forgotten at t, i.e. $B_s = B_t \cup \{v\}$. A partial solution at node s, i.e., a coloring γ_s of G_s, may have a monochromatic maximal clique using the vertex v, provided that the clique is fully contained in B_s, while partial solutions at the node t may not. On the other hand, a clique that is maximal inside B_s may not be maximal in G_s. Moreover, as soon as a maximal clique uses a vertex from $V_s \setminus B_s$ it is not monochromatic in any partial solution at the node s, as asserted by the definition of the table entries; provided that $\mathsf{tab}[s, \gamma_s] = 1$. We can therefore consult with the oracle $\mathbb{O}_s$ to verify if the intersection of any color class with the neighborhood of v, together with the vertex v, contains a maximal clique in G_s (and not just in $G[B_s]$). Therefore, for a given coloring $\gamma_t \colon B_t \to \{1, \ldots, q\}$, we can check whether or not there is a partial solution in G_t whose restriction to B_t is equal to γ_t as follows. For each color $c \in \{1, \ldots, q\}$, extend γ_t to a coloring γ_s of B_s by assigning vertex v color c. If $\mathsf{tab}[s, \gamma_s] = 1$, then we check if the set consisting of v and its neighbors colored c does *not* contain a maximal clique in G_s, in which case we can set $\mathsf{tab}[t, \gamma_t]$ to 1. If there is no color c passing these checks then we know that we can set $\mathsf{tab}[t, \gamma_t]$ to 0. We summarize this in Algorithm 1.

Input: G, $(T, \mathcal{B})$ as above, forget node $t \in V(T)$

Let $v \in B_s \setminus B_t$ be the vertex forgotten at t;
Let $\mathbb{O}_t$ be the clique oracle of t from Proposition 3.1;
foreach $\gamma_t \colon B_t \to \{1, \ldots, q\}$ **do**
    $\mathsf{tab}[t, \gamma_t] \leftarrow 0$;
    foreach $c \in \{1, \ldots, q\}$ **do**
        Let $\gamma_s \colon B_s \to \{1, \ldots, q\}$ be such that for all $u \in B_t$, $\gamma_s(u) = \gamma_t(u)$, and $\gamma_s(v) = c$;
        if $\mathsf{tab}[s, \gamma_s] = 1$ **then**
            Let $S \leftarrow (N(v) \cap \gamma_t^{-1}(c)) \cup \{v\}$;
            if $\mathbb{O}_s(S) = 0$ **then** $\mathsf{tab}[t, \gamma_t] \leftarrow 1$;

Algorithm 1: Algorithm to compute all table entries at a forget node t with child s, assuming all table entries at s have been computed. (Notation: For a set $S \subseteq B_t$, $\mathbb{O}_t(S) = 0$ if and only S contains no maximal clique in G_t.)

This completes the description of the algorithm. Correctness follows from the description of the computation of the table entries, by induction on the height of each node. For the runtime, we first take $2^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot n$ time to construct the clique oracles. At each node, there are at most $q^{\mathsf{tw}+1}$ table entries to consider, and it is clear that the computation of a table entry at a leaf, introduce, or join node takes constant time. With the clique oracle at hand, computing an entry at a forget node takes time $\mathcal{O}(q) = \mathcal{O}(1)$. Therefore, at each node all table entries can be computed in time $\mathcal{O}(q^{\mathsf{tw}})$ and since there are $\mathcal{O}(\mathsf{tw} \cdot n)$ nodes in the tree decomposition, all table entries are computed in time $q^{\mathsf{tw}} \cdot \mathsf{tw}^{\mathcal{O}(1)} \cdot n$, which, since $q \geq 2$, bounds the total runtime of the algorithm. Using standard memoization techniques, the algorithm can also construct a coloring, if one exists. ☐

3.2 Lower Bound

In this section we show that the previously presented algorithm is optimal under SETH. In fact, we give lower bounds for much larger parameters than treewidth, which we now define briefly. The *feedback vertex set number* of a graph G is the size of a smallest set of vertices $S \subseteq V(G)$ such that $G - S$ is a forest. The *distance to a linear/caterpillar forest* of a graph G is the size of a smallest set $S \subseteq V(G)$ such that $G - S$ is a linear/caterpillar forest.

Our proofs give lower bounds under SETH for the parameters distance to a linear forest (for $q = 2$), and distance to a caterpillar forest (for $q \geq 3$). Note that both paths and caterpillars have pathwidth 1, and clearly, they do not contain any cycles. Therefore, a lower bound parameterized by the distance to a linear/caterpillar forest implies a lower bound for the parameter pathwidth plus feedback vertex set number. For $q = 2$, we give a reduction from s-Not-All-Equal SAT (s-NAE-SAT) on n variables. Cygan et al. [13] showed that under SETH, for any $\epsilon > 0$, there is some constant s such that s-NAE-SAT cannot be solved in time $\mathcal{O}^\star((2-\epsilon)^n)$. For all $q \geq 3$, we reduce from q-List Coloring, where we are given a graph G and a list for each of its vertices which is a subset of $\{1, 2, \ldots, q\}$, and the question is whether G has a proper coloring such that each vertex receives a color from its list. Parameterized by the size t of a deletion set to a linear forest, this problem is known to have no $\mathcal{O}^\star((q-\epsilon)^t)$-time algorithms under SETH [22]. Our construction uses the fact that on triangle-free graphs, the proper colorings and the clique colorings coincide, and exploits properties of Mycielski graphs.

We first give the lower bound for the case $q = 2$. We would like to remark that Kratochvíl and Tuza [26] gave a reduction from s-Not-All-Equal SAT to 2-Clique Coloring as well, but

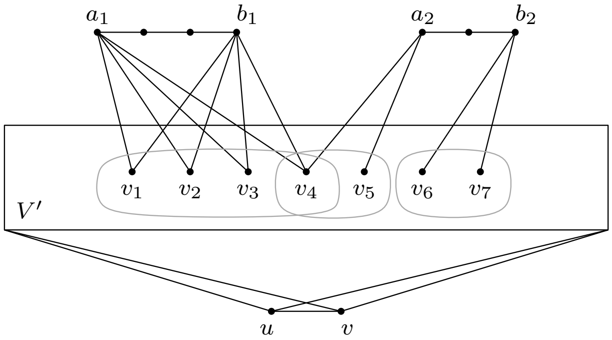

Figure 1: Depiction of G_ϕ with two clauses, namely a monotone clause $C_1 = \neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4$ and a non-monotone clause $C_2 = x_4 \vee x_5 \vee \neg x_6 \vee \neg x_7$. Note that $G_\phi - V'$ is a linear forest.

their reduction does not imply the fine-grained lower bound we aim for here: the resulting graph is at distance $2n$ to a disjoint union of cliques of constant size (at most s). This only rules out $\mathcal{O}^\star((\sqrt{2}-\epsilon)^t)$-time algorithms parameterized by pathwidth, and does not give any lower bound if the feedback vertex set number is another component of the parameter.

Theorem 3.3. *For any $\epsilon > 0$, 2-Clique Coloring parameterized by the distance t to a linear forest cannot be solved in time $\mathcal{O}^\star((2-\epsilon)^t)$, unless* SETH *fails.*

Proof. We give a reduction from the well-known s-NAE-SAT problem, in which we are given a boolean CNF formula ϕ whose clauses are of size at most s, and the question is whether there is a truth assignment to the variables of ϕ, such that in each clause, at least one literal evaluates to true and at least one literal evaluates to false.

Let ϕ be a boolean CNF formula on n variables $x_1, \ldots, x_n$ with maximum clause size s. We denote by $\texttt{clauses}(\phi)$ the set of clauses of ϕ and by $\texttt{vars}(C)$ the set of variables that appear in the clause C of ϕ. A clause is called *monotone* if either all literals are positive or all literals are negated.

Given ϕ, we construct an instance G_ϕ for 2-Clique Coloring as follows. For each variable x_i, we create a vertex v_i in G_ϕ. Let $V' = \{v_1, \ldots, v_n\}$. For each set S of variables, let $V_S = \{v_i \mid x_i \in S\}$. For each clause C_i of ϕ, we add the following clause gadget to G_ϕ. If C_i is monotone, add a path on four vertices to G_ϕ, the end vertices of which are a_i and b_i. Make $N(a_i) \cap V' = N(b_i) \cap V' = V_{\texttt{vars}(C_i)}$, and make $V_{\texttt{vars}(C_i)} \subset V'$ a clique. If C_i is not monotone, let $\texttt{pos}(C_i)$ (resp. $\texttt{neg}(C_i)$) denote the set of variables with positive (resp. negative) literals in C_i. Add a path on three vertices to G_ϕ, the end vertices of which are a_i and b_i, make $N(a_i) \cap V' = V_{\texttt{pos}(C_i)}$ and make $V_{\texttt{pos}(C_i)}$ a clique. Analogously, make $N(b_i) \cap V' = V_{\texttt{neg}(C_i)}$ and make $V_{\texttt{neg}(C_i)}$ a clique. Finally, add two adjacent vertices u, v to G_ϕ and make $N[u] = N[v] = \{u, v\} \cup V'$. See Figure 1.

We will show that G_ϕ is a yes-instance to 2-Clique Coloring if and only if ϕ is a yes-instance to s-NAE-SAT. We first make the following observation about the maximal cliques of G_ϕ, which follows directly from the fact that the vertices u and v are complete to V'.

Observation 3.3.1. The vertices u and v belong to every maximal clique of $G_\phi[V' \cup \{u, v\}]$.

Claim 3.3.2. Let $f : V(G_\phi) \to \{0, 1\}$ be a 2-clique coloring of G_ϕ and C_i be a clause of ϕ. If C_i is monotone, then $f(a_i) \neq f(b_i)$. Otherwise, $f(a_i) = f(b_i)$.

Proof. If C_i is monotone, a_i and b_i are the end vertices of a path on four vertices, each edge of which is a maximal clique of G_ϕ. Thus, $f(a_i) \neq f(b_i)$ in any 2-clique coloring f of G_ϕ. Similarly, if

C_i is not monotone, a_i and b_i are the end vertices of a path on three vertices, each edge of which is a maximal clique of G_ϕ. Hence $f(a_i) = f(b_i)$. ⌟

Now, suppose G_ϕ has a 2-clique coloring $f : V(G_\phi) \to \{0, 1\}$. We construct a truth assignment for $\{x_1, \ldots, x_n\}$ according to the colors assigned to the vertices of V' by f. That is, if $f(v_i) = 0$, we set x_i to false, and if $f(v_i) = 1$, we set x_i to true. We will now show that this assignment satisfies all clauses of ϕ. Let C_i be a clause of ϕ. First, assume that C_i is monotone. By Claim 3.3.2, $f(a_i) \neq f(b_i)$. Since $V_{\texttt{vars}(C_i)} \cup \{a_i\}$ is a maximal clique of G_ϕ, the vertices of $V_{\texttt{vars}(C_i)}$ cannot all be colored with $f(a_i)$. Similarly, $V_{\texttt{vars}(C_i)} \cup \{b_i\}$ is a maximal clique of G_ϕ, the vertices of $V_{\texttt{vars}(C_i)}$ cannot all be colored with $f(b_i)$. Thus, there exist two vertices $v_j, v_k \in V_{\texttt{vars}(C_i)}$ such that $f(v_j) \neq f(v_k)$. Since C_i is monotone, this implies that x_j and x_k are not both evaluated to the same value and therefore C_i is satisfied. Now assume C_i is not monotone. By Claim 3.3.2, $f(a_i) = f(b_i)$. Hence, since $V_{\texttt{pos}(C_i)} \cup \{a_i\}$ and $V_{\texttt{neg}(C_i)} \cup \{b_i\}$ are maximal cliques of G, there exists $v_j \in V_{\texttt{pos}(C_i)}$ and $v_k \in V_{\texttt{neg}(C_i)}$ such that $f(v_j) = f(v_k)$. This implies that x_j and x_k are not evaluated to the same value under the proposed assignment and thus C_i is satisfied.

For the other direction, assume ϕ admits an assignment ξ satisfying all clauses. We construct a clique coloring $f : V(G_\phi) \to \{0, 1\}$ for G_ϕ in the following way. Color the vertices of V' according to the assignment of the variables of ϕ. That is, if $\xi(x_i) = \text{true}$ (resp. $\xi(x_i) = \text{false}$), define $f(v_i) = 1$ (resp. $f(v_i) = 0$). If C_i is monotone, let $a_i a_i' b_i' b_i$ be the path on four vertices connecting a_i and b_i in the clause gadget of C_i. Define $f(a_i) = f(b_i') = 1$ and $f(a_i') = f(b_i) = 0$. If C_i is not monotone, let $a_i a_i' b_i$ be the three vertex path connecting a_i and b_i in the clause gadget of C_i. If all the vertices of either $V_{\texttt{pos}(C_i)}$ or $V_{\texttt{neg}(C_i)}$ are colored 1, set $f(a_i) = f(b_i) = 0$ and $f(a_i') = 1$. Otherwise set $f(a_i) = f(b_i) = 1$ and $f(a_i') = 0$. Finally, define $f(u) = 0$ and $f(v) = 1$. To see that this is indeed a 2-clique coloring of G_ϕ, first note that by Observation 3.3.1, no maximal clique contained in $G_\phi[V' \cup \{u, v\}]$ is monochromatic. Furthermore, since all paths of the clause gadgets are properly colored, no maximal clique contained in $G_\phi - (V' \cup \{u, v\})$ is monochromatic. It remains to show that for each clause C_i, the maximal cliques $A_i = \{a_i\} \cup (N(a_i) \cap V')$ and $B_i = \{b_i\} \cap (N(b_i) \cap V')$ are not monochromatic. Let C_i be a monotone clause. Since C_i is satisfied, there exist $x_j, x_k \in \texttt{vars}(C_i)$ such that $\xi(x_j) \neq \xi(x_k)$. Hence, $f(v_j) \neq f(v_k)$, which shows that A_i and B_i are each not monochromatic. If C_i is not monotone, by definition the vertices of A_i and B_i are not all colored 1. Suppose all the vertices of A_i are colored 0. In particular, we have $f(a_i) = f(b_i) = 0$. This implies that, by construction, all the vertices of $N(b_i) = V_{\texttt{neg}(C_i)}$ are colored 1. However, this is a contradiction with the fact that the clause C_i is satisfied, since all its literals are evaluated to false. Hence, f is indeed a 2-clique coloring of G_ϕ.

Finally, note that $G - V'$ is a disjoint union of paths of length at most four. Hence, G is at distance n to a linear forest. Therefore, if for some $\epsilon > 0$, 2-Clique Coloring parameterized by the distance t to a linear forest can be solved in time $\mathcal{O}^\star((2-\epsilon)^t)$, then s-NAE-SAT can be solved in time $\mathcal{O}^\star((2-\epsilon)^n)$, which would contradict SETH [13]. This concludes the proof. □

We now turn to the case $q \geq 3$. Our reduction is from q-List-Coloring parameterized by the distance t to a linear forest, which is known to have no $\mathcal{O}^\star((q-\epsilon)^t)$-time algorithms under SETH.

Theorem 3.4 (Jaffke and Jansen [22]). *For any $\epsilon > 0$ and any fixed $q \geq 3$, q-List Coloring on triangle-free graphs parameterized by the distance t to a linear forest cannot be solved in time $\mathcal{O}^\star((q-\epsilon)^t)$, unless SETH fails.*

We crucially use the fact that in triangle-free graphs, the proper colorings and the clique colorings coincide, which is both the key and the challenging part of the reduction. In [22], it is not explicitly mentioned that the lower bound from the previous theorem holds on triangle-free graphs, so let us briefly justify this. The reduction presented in [22] is from s-SAT on n variables, and given a formula ϕ, the graph G_ϕ of the resulting q-List Coloring instance has the following structure. The truth assignments of the variables of ϕ are encoded as colorings of a set of vertices V' that are independent in G_ϕ, and for each clause C in ϕ and each coloring of some subset $V_C \subseteq V'$ that corresponds to a truth assignment μ that does not satisfy C, there is a path P_μ in G that cannot be properly list colored if and only if the coloring μ appears on V_C. This is ensured by connecting P_μ to V_C via a matching, which does not introduce triangles. Since each edge of G_ϕ is either on such a path or part of one of such matching, there are no triangles in G_ϕ.

Now, Theorem 3.4 already gives a lower bound, under SETH, for the *list*-version of q-Clique Coloring parameterized by the distance to a linear forest. To obtain the desired lower bound (without lists), we need to simulate the lists without introducing triangles. For proper colorings, there is a standard way to simulate lists that is frequently used in such reductions (e.g., [22, 27]): We add a clique on vertex set $[q]$ to the input graph G, and for each $i \in [q]$ and $v \in V(G)$, we add an edge between i and v if color i does *not* appear on the list of v. Then, G has a coloring respecting the given lists if and only if the resulting graph has a proper coloring with q colors. In our setting, however, this clearly does not work, since the previous step introduces triangles of two kinds:

(i) Between any triple of vertices in the q-clique.

(ii) Between vertices of G and the q-clique.

To avoid triangles of type (i), we replace the q-clique by a gadget H_q consisting of several Mycielski graphs that are connected to each other in such a way that we can identify a set of q vertices that receive pairwise distinct colors in any proper coloring.[3] These q vertices can then be used in the same way as the vertices of the q-clique mentioned above. Since Mycielski graphs are triangle-free, and by the way we connect them, H_q is a triangle-free graph. To avoid triangles of type (ii), we do not connect the remaining vertices to H_q directly, but instead we add a middle layer of vertices that we connect to H_q in such a way that only one given color can appear on each vertex in any proper coloring. This way we propagate the forcing of colors while avoiding triangles of type (ii). The latter step is also the reason why our lower bound only holds when parameterized by the distance to a caterpillar forest and not by the distance to a linear forest: after removing the modulator of the q-List-Coloring instance and the gadget H_q, the remaining graph consists of the linear forest from the q-List-Coloring instance and the vertices of the middle layer; together they form a caterpillar forest.

Theorem 3.5. *For any $\epsilon > 0$ and any fixed $q \geq 3$, q-Clique Coloring parameterized by the distance t to a caterpillar forest cannot be solved in time $\mathcal{O}^\star((q-\epsilon)^t)$, unless SETH fails.*

Proof. We give a reduction from q-List Coloring on triangle-free graphs parameterized by distance to linear forest. In this proof we use the phrases "q-colorable" as short for "can be properly colored with at most q colors", and "q-coloring" as short for "a proper coloring with at most q colors". To construct our instance of q-Clique Coloring, we will first describe the construction of a color selection gadget, and then describe how this gadget is attached to the rest of the graph. The description of the color selection gadget makes use of the famous Mycielski graphs. For completeness, we briefly describe how Mycielski graphs are recursively constructed and some of their useful properties. For every $p \geq 2$, the Mycielski graph M_p is a triangle-free graph with chromatic

[3] We would like to remark that also Marx [29] used Mycielski graphs and their properties in hardness proofs for the Clique Coloring problem.

number p. For $p = 2$, we define $M_2 = K_2$. For $p \geq 3$, the graph M_p is obtained from M_{p-1} as follows. Let $V(M_{p-1}) = \{v_1, \dots, v_n\}$. Then $V(M_p) = V(M_{p-1}) \cup \{u_1, \dots, u_n, w\}$. The vertices of $V(M_{p-1})$ induce a copy of M_{p-1} in M_p, each u_i is adjacent to all the neighbors of v_i in M_{p-1} and $N(w) = \{u_1, ..., u_n\}$. Hence, $|V(M_p)| = 3 \cdot 2^{p-2} - 1$. Moreover, it is known that M_p is edge-critical, that is, the deletion of any edge of M_p leads to a $(p-1)$-colorable graph (see for instance [5, 28]). For our construction, we will use the graph M'_p, obtained from M_p by the deletion of an arbitrary edge xy. The following observation follows directly from the fact that M_p is edge-critical.

Observation 3.5.1. Let M'_p be the graph obtained from M_p by the deletion of an edge xy. Then, M'_p is $(p-1)$-colorable, and in any $(p-1)$-coloring of M'_p, the vertices x and y receive the same color.

Color selection gadget. We construct a gadget H_q in the following way. Consider q disjoint copies of M'_{q+1}. For $1 \leq i \leq q$, let $x_i y_i$ be the edge removed from M_{q+1} in order to obtain the i-th copy of M'_{q+1}. For each i, add $q-1$ false twins to y_i. We denote these vertices by y_{ij}, with $1 \leq j \leq q$, $j \neq i$. Then delete the vertex y_i, for every i. Note that this graph is still q-colorable and, by Observation 3.5.1, in every such q-coloring, for each i, the vertices x_i and y_{ij}, for all $j \neq i$, receive the same color. Now we add $\binom{q}{2}$ edges to connect the copies of M'_{q+1}: for $1 \leq i < j \leq q$, add the edge $y_{ij} y_{ji}$ to H_q. Note that H_q remains triangle-free after the addition of these edges, since for all $1 \leq i < j \leq q$, $N(y_{ij}) \cap N(y_{ji}) = \emptyset$. We will need the following property of the q-colorings of H_q.

Claim 3.5.2. The graph H_q is q-colorable. Moreover, in any q-coloring ϕ of H_q, $\phi(x_i) \neq \phi(x_j)$ for all $1 \leq i < j \leq q$.

Proof. Suppose for a contradiction that there exists a q-coloring of H_q such that $\phi(x_i) = \phi(x_j)$, for some $i \neq j$. By Observation 3.5.1, we know that $\phi(x_i) = \phi(y_{ij})$. Similarly, $\phi(x_j) = \phi(y_{ji})$. This implies that $\phi(y_{ij}) = \phi(y_{ji})$, which is a contradiction, since y_{ij} and y_{ji} are adjacent by construction. To see that a q-coloring indeed exists for H_q, first note that, by Observation 3.5.1, each copy of M'_{q+1} has a q-coloring in which x_i and y_i are assigned the same color. We can then permute the colors within a copy to obtain a proper coloring of that copy in which x_i and y_i receive color i. To complete the coloring, assign color i to every y_{ij} that is a false twin of y_i. This yields a proper q-coloring of H_q. ⌟

We are now ready to describe the construction of our instance G' to q-Clique Coloring. Let (G, L) be an instance of q-List Coloring on triangle-free graphs that is at distance t from a linear forest. We construct G' as follows. Add a copy of G and a copy of H_q to G'. We denote by V' the set of vertices corresponding to $V(G)$ in G'. For each $v \in V'$, add $q - |L(v)|$ vertices adjacent to v. We denote these vertices by $\{v_j \mid j \notin L(v)\}$. Finally, make v_j adjacent to all the vertices of $\{x_\ell \mid \ell \neq j\}$. See Figure 2.

Note that G' is triangle-free since H_q and G are triangle-free, and $N(v_j) \cap V' = \{v\}$ and $N(v_j) \cap V(H_q)$ is an independent set. Furthermore, let $S \subseteq V(G)$ be a set such that $G - S$ is a linear forest and $|S| = t$. Then $S \cup V(H_q)$ is such that each connected component of $G' - (S \cup V(H_q))$ is a caterpillar and $|S \cup V(H_q)| = t + q(3 \cdot 2^{q-1} + q - 3) = t + \mathcal{O}(1)$, since q is a constant.

We will show that (G, L) is a yes-instance to q-List Coloring if and only if G' is a yes-instance to q-Clique Coloring. Note that since G' is a triangle-free graph, every clique coloring of G' is a proper coloring of it as well. First, suppose (G, L) is a yes-instance to q-List Coloring and let ϕ be a q-list coloring for G. We give a q-coloring ϕ' for G' in the following way. If $v \in V'$, make $\phi'(v) = \phi(v)$. For each $v_j \in N(v)$, make $\phi'(v_j) = j$. Note that since $j \notin L(v)$, we have that $\phi'(v) \neq \phi(v_j)$. Finally, consider a proper q-coloring of H_q. By Claim 3.5.2, the vertices $x_1, \dots, x_q$ were assigned pairwise distinct colors. Without loss of generality, we can assume x_i received color i.

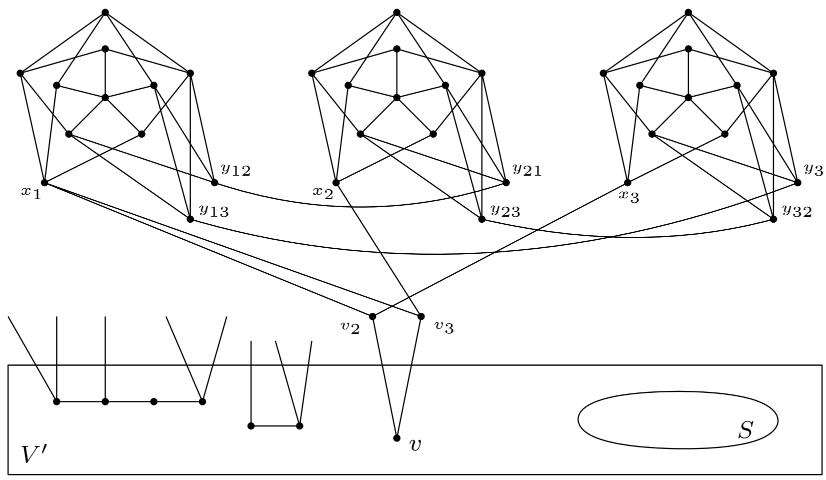

Figure 2: In this instance, $q = 3$ and $L(v) = \{1\}$. Note that $G' - (S \cup V(H_q))$ is a caterpillar forest.

Extend ϕ to the remaining vertices of G' according to this coloring of H_q. This leads to a proper q-coloring of G', since $\phi(v_j) = j$ and v_j is not adjacent to x_j.

Now assume G' admits a q-clique coloring ϕ. We will show that $\phi|_{V'}$ is a q-list coloring for (G, L). Since G' is triangle-free, it is clear that $\phi|_{V'}$ is a proper coloring of G. It remains to show it satisfies the constraints imposed by the lists. By Claim 3.5.2, we can again assume that $\phi(x_i) = i$, for every i. For every $v \in V'$, since $\{x_\ell \mid \ell \neq j\} \subset N(v_j)$, we necessarily have $\phi(v_j) = j$. Finally, since for every $c \notin L(v)$ there is a neighbor of v that is colored c (namely v_c), we conclude that $\phi(v) \in L(v)$.

Now, suppose that q-Clique Coloring admits an algorithm running in time $\mathcal{O}^\star((q-\epsilon)^{t'})$, for some $\epsilon > 0$, where t' is the distance of the input graph to a caterpillar forest. Then, we can solve q-List-Coloring paramterized by the distance t to a linear forest by applying the above reduction, giving a q-Clique Coloring instance at distance $t + \mathcal{O}(1)$ to a caterpillar forest, and solving the resulting q-Clique Coloring instance. Correctness is argued in the previous paragraphs, and the runtime of the resulting algorithm is $\mathcal{O}^\star((q-\epsilon)^{t+\mathcal{O}(1)}) = \mathcal{O}^\star((q-\epsilon)^t)$, contradicting SETH by Theorem 3.4. □

Since the instance of q-Clique Coloring constructed in the proof of Theorem 3.5 is a triangle-free graph, we obtain the following corollary.

Corollary 3.6. For any $\epsilon > 0$ and any fixed $q \geq 3$, q-Coloring on triangle-free graphs parameterized by the distance t to a caterpillar forest cannot be solved in time $\mathcal{O}^\star((q-\epsilon)^t)$, unless SETH fails.

4 Parameterized by Clique-width

In this section, we give an XP-time algorithm for Clique Coloring parameterized by clique-width, more precisely, parameterized by the equivalent measure module-width. We provide an algorithm that given an n-vertex graph G with one of its rooted branch decompositions of module-width w and an integer k, decides whether G has a clique coloring with k colors in time $k^{f(w)} \cdot n$, where $f(w) = 2^{2^{\mathcal{O}(w)}}$. Before we describe the algorithm, we give a high level outline of its main ideas, and where the double exponential dependence on w in the degree of the polynomial comes from.

The algorithm is bottom-up dynamic programming along the given branch decomposition of the input graph. Let t be some node in the branch decomposition. To keep the number of table entries bounded by something that is XP in the module-width, we have to find a way to group color classes into *types* whose number is upper bounded by a function of w alone. The intention is that two color classes of the same type are interchangeable with respect to the underlying coloring being completable to a valid clique coloring of the whole graph. Partial solutions (colorings of the subgraph G_t) can then be described by remembering, for each type, how many color classes of that type there are. If the number of types is $f(w)$ for some function f, this gives an upper bound of $k^{f(w)}$ on the number of table entries at each node of the branch decomposition.

Let us discuss what kind of information goes into the definition of a type. Since the final coloring of G has to avoid monochromatic maximal cliques, we maintain information about cliques in G_t that are or may become monochromatic maximal cliques in some extension of the coloring at hand. A natural attempt would be to consider and describe maximal cliques in G_t by their intersection patterns with the equivalence classes of $\sim_t$. However, it is not sufficient to consider only maximal cliques in G_t; given a maximal clique X in G_t, it may happen that in $\overline{V_t}$ there is a vertex v that is adjacent to a strict subset $Y \subset X$ of that clique, forming a maximal clique with Y – which does not fully contain X – in a supergraph of G_t. Considering the equivalence classes of $\sim_t$, this implies that the equivalence classes containing Y and the ones containing $X \setminus Y$ are disjoint. We therefore consider cliques X that are *maximal in the subgraph induced by the equivalence classes containing vertices of* X. We call such cliques X *eqc-maximal*, and observe that with a little extra information, we can keep track of the forming and disintegrating of eqc-maximal cliques along the branch decomposition. If an eqc-maximal clique is fully contained in some set of vertices (/color class) C, then we call it *potentially bad for* C. A potentially bad clique is described via its *profile*, which consists of the intersection pattern with the equivalence classes of $\sim_t$, and some extra information. At each node, there are at most $2^{\mathcal{O}(w)}$ profiles.

Equipped with this definition, we can define the notion of a t-type of a color class C, which is simply the subset of profiles at t, such that G_t contains a potentially bad clique with that C-profile. It immediately follows that the number of t-types is $2^{2^{\mathcal{O}(w)}}$. Now, colorings $\mathcal{C}_t$ of G_t are described by their *t-signature*, which records how many color classes of each type $\mathcal{C}_t$ has. There are at most $k^{f(w)}$ many t-signatures, where $f(w) = 2^{2^{\mathcal{O}(w)}}$, and this essentially bounds the runtime of the resulting algorithm to $n \cdot k^{f(w)} = n^{\mathcal{O}(f(w))}$.

At the root node $\mathfrak{r} \in V(T)$, there is only one equivalence class, namely $V_\mathfrak{r} = V(G)$, and if in a coloring, there is a clique that is potentially bad for some color class, then it is indeed a monochromatic maximal clique. Therefore, at the root node, we only have to check whether there is a coloring all of whose color classes have no potentially bad cliques.

4.1 Potentially Bad Cliques

We now introduce the main concept used to describe color classes in partial solutions of our algorithms, namely potentially bad cliques. These are cliques that are monochromatic in some subgraph induced by a set of equivalence classes.

Definition 4.1 (Potentially Bad Clique). Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T)$. A clique X in G_t is called *eqc-maximal (in G_t)* if it is maximal in $G_t[V(\mathsf{eqc}_t(X))]$. Let $C \subseteq V_t$ and let X be a clique in G_t. Then, X is called *potentially bad for C (in G_t)*, if X is eqc-maximal in G_t and $X \subseteq C$.

Naturally, it is not feasible to keep track of *all* potentially bad cliques. We therefore capture the most vital information about potentially bad cliques in the following notion of a *profile*. For our

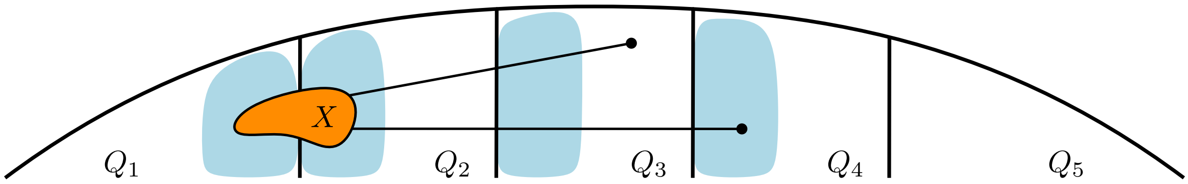

Figure 3: Illustration of the C-profile of a clique X that is potentially bad for a color class C, depicted as the shaded areas within the equivalence classes. In this case, we have that $\pi(X \mid C) = (\{Q_1, Q_2\}, \{Q_3, Q_4\})$.

algorithm, it is only important to know for a color class whether or not it has some potentially bad clique with a given profile, rather than how many, or what its vertices are. This is key to reduce the amount of information we need to store about partial solutions.

There are two components of a profile of a potentially bad clique X; the first one is the set of equivalence classes $\mathcal{Q}$ containing its vertices, and the second one consists of the equivalence classes $P \notin \mathcal{Q}$ that have a vertex that is complete to X. This is because, at a later stage, P may be merged with an equivalence class containing vertices of X (via the bubbles), in which case X is no longer potentially bad. We illustrate the following definition in Figure 3.

Definition 4.2 (Profile). Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$ and let $t \in V(T)$. Let $C \subseteq V_t$ and let X be a clique in G_t that is potentially bad for C. The *C-profile* of X is a pair of subsets of $V_t/{\sim_t}$, $\pi(X \mid C) := (\mathcal{Q}, \mathcal{P})$, where

$$\mathcal{Q} = \mathsf{eqc}_t(X) \text{ and } \mathcal{P} = \{P \in \overline{\mathsf{eqc}}_t(X) \mid \exists v \in P \colon X \subseteq N(v)\}.$$

We call the set of all pairs of disjoint subsets of $V_t/{\sim_t}$, where the first coordinate is nonempty, the *profiles at t*, formally, $\Pi_t := \{(\mathcal{Q}, \mathcal{P}) \mid \mathcal{Q}, \mathcal{P} \subseteq V_t/\sim_t \colon \mathcal{Q} \neq \emptyset \wedge \mathcal{Q} \cap \mathcal{P} = \emptyset\}$.

Observation 4.3. Let $(T, \mathcal{L})$ be a rooted branch decomposition. For each $t \in V(T)$, there are at most $2^{\mathcal{O}(w)}$ profiles at t, where $w = \mathsf{mw}(T, \mathcal{L})$.

Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s), and let $\pi_r \in \Pi_r$ and $\pi_s \in \Pi_s$ be a pair of profiles. We are now working towards a notion that precisely captures when and how a potentially bad clique in G_r for some $C_r \subseteq V_r$ with C_r-profile π_r can be merged with a potentially bad clique in G_s for some $C_s \subseteq V_s$ with C_s-profile π_s to obtain a potentially bad clique for $C_r \cup C_s$ in G_t. As it turns out, if this is possible, then the profile of the resulting clique only depends on π_r, π_s, and the operator of t. Note that for now, we focus on the case when the cliques in G_r and G_s are both nonempty, and we discuss the case when one of them is empty below.

Before we proceed with this description, we need to introduce some more concepts. We illustrate all of the following concepts in Figure 4. For a set of equivalence classes $\mathcal{S} \subseteq V_r/{\sim_r} \cup V_s/{\sim_s}$, its *bubble buddies at t*, denoted by $\mathsf{bb}_t(\mathcal{S})$, are the equivalence classes of $V_r/{\sim_r} \cup V_s/{\sim_s}$ that are in the same bubble as some equivalence class in $\mathcal{S}$:

$$\mathsf{bb}_t(\mathcal{S}) := \bigcup\nolimits_{p \in \{r,s\}} \{Q_p \in V_p/{\sim_p} \mid \eta_p(Q_p) \in \eta_p(\mathcal{S} \cap V_p/{\sim_p})\}$$

We say that $\pi_r = (\mathcal{Q}_r, \mathcal{P}_r)$ and $\pi_s = (\mathcal{Q}_s, \mathcal{P}_s)$ are *compatible*, if $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a maximal biclique in

$$H'_t(\pi_r, \pi_s) := H_t[(\mathcal{Q}_r \cup \mathcal{Q}_s) \cup ((\mathcal{P}_r \cup \mathcal{P}_s) \cap \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s))]. \tag{1}$$

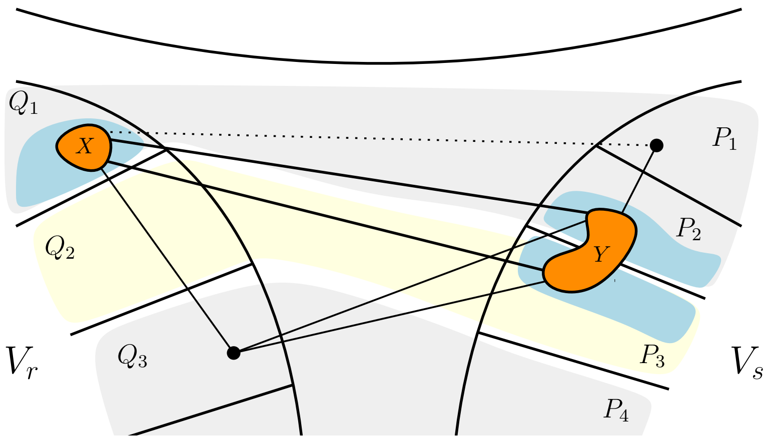

Figure 4: Merging a potentially bad clique X in G_r with a potentially bad clique Y in G_s to obtain a potentially bad clique in G_t. The color class at hand is depicted in blue and the gray and yellow areas show the (three) bubbles. Note that the equivalence classes P_1 and Q_2 are bubble buddies of $\mathsf{eqc}_r(X)$ and $\mathsf{eqc}_s(Y)$. Moreover, the types of X and Y are compatible, since $\{Q_1, P_2, P_3\}$ is a maximal biclique in $H_t[\{Q_1, P_1, P_2, P_3\}]$. The dotted line between the vertex in P_1 and Q_1 shows that there is no edge between the vertices in P_1 and Q_1. Observe that if these edges were present, then X and Y would not be compatible, since $\{Q_1, P_2, P_3\}$ would no longer be a maximal biclique in $H_t[\{Q_1, P_1, P_2, P_3\}]$. Finally, note that the equivalence class of $\sim_t$ corresponding to the bubble containing Q_3 will have a vertex that is complete to the potentially bad clique $X \cup Y$.

As we show below, the notion of compatibility precisely captures the 'merging behavior' of potentially bad cliques. Moreover, for π_r and π_s compatible, we can immediately construct the profile of the resulting potentially bad clique: the *merge profile* of π_r and π_s is the profile $\mu(\pi_r, \pi_s) = (\mathcal{Q}_t, \mathcal{P}_t)$ such that

- $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s)$ and
- $\mathcal{P}_t = \bigcup_{\{o,p\}=\{r,s\}} \{\eta_p(Q_p) \mid Q_p \in \mathcal{P}_p \setminus \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s) \colon \mathcal{Q}_o \subseteq N_{H_t}(Q_p)\}$.

Lemma 4.4. *Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s). For all $p \in \{r, s\}$, let $C_p \subseteq V_p$, let X_p be a clique in G_p that is potentially bad for C_p, and let $\pi_p := \pi(X_p \mid C_p) = (\mathcal{Q}_p, \mathcal{P}_p)$. If π_r and π_s are compatible, then $X_t := X_r \cup X_s$ is a clique that is potentially bad for $C_t := C_r \cup C_s$, and $\pi(X_t \mid C_t) = \mu(\pi_r, \pi_s)$.*

Proof. We first argue that X_t is a clique. Since X_r and X_s are cliques, we only have to show that for each $v_r \in X_r$ and $v_s \in X_s$, $v_r v_s \in E(G_t)$. In other words, if Q_r is the equivalence class of $\sim_r$ containing v_r, and Q_s is the equivalence class of $\sim_s$ containing v_s, then $Q_r Q_s \in E(H_t)$. Now, $Q_r \in \mathsf{eqc}_r(X_r) = \mathcal{Q}_r$ and $Q_s \in \mathsf{eqc}_s(X_s) = \mathcal{Q}_s$, and since π_r and π_s are compatible, we have that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in H_t, therefore $Q_r Q_s \in E(H_t)$.

Next, we show that X_t is potentially bad for C_t. Since X_r and X_s are potentially bad for C_r and C_s, respectively, we have that $X_r \subseteq C_r$ and $X_s \subseteq C_s$, and therefore $X_t = X_r \cup X_s \subseteq C_r \cup C_s = C_t$. It remains to show that X_t is eqc-maximal. Suppose not, and let $y \in V(\mathsf{eqc}_t(X_t))$ be a vertex that is complete to X_t. First, we know that $y \notin V(\mathsf{eqc}_r(X_r) \cup \mathsf{eqc}_s(X_s))$, for if $y \in V(\mathsf{eqc}_p(X_p))$ for some $p \in \{r, s\}$, then X_p is not eqc-maximal, contradicting X_p being potentially bad for C_p. On the other hand, we have that $\mathsf{eqc}_t(X_t) = \mathsf{bb}_t(\mathsf{eqc}_r(X_r) \cup \mathsf{eqc}_s(X_s)) = \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$. We may assume that for some $p \in \{r, s\}$, the vertex y is contained in some $Q_p \in \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s) \setminus (\mathcal{Q}_r \cup \mathcal{Q}_s)$. Assume up to renaming that $p = r$. Since y is complete to X_t, we have that y is complete to X_r, and therefore

$Q_r \in \mathcal{P}_r$. In other words, Q_r is contained in the graph $H'_t(\pi_r, \pi_s)$ as described in Equation (1). Moreover, since y is complete to X_s, we have that Q_r is complete to $\mathsf{eqc}_s(X_s) = \mathcal{Q}_s$. This implies that $\{Q_r\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in $H'_t(\pi_r, \pi_s)$, contradicting π_r and π_s being compatible.

To conclude the proof, we need to show that $\pi(X_t \mid C_t) = \mu(\pi_r, \pi_s)$. Let $\mu(\pi_r, \pi_s) = (\mathcal{Q}_t, \mathcal{P}_t)$. We first show that $\mathsf{eqc}_t(X_t) = \mathcal{Q}_t$. To see that $\mathcal{Q}_t = \eta_r(\mathcal{Q}_r) \cup \eta_s(\mathcal{Q}_s) \subseteq \mathsf{eqc}_t(X_t)$, we observe that for all $Q_p \in \mathcal{Q}_p$, there is an $x \in X_p \cap Q_p$. This means that $x \in \eta_p(Q_p)$, therefore $X_t \cap \eta_p(Q_p) \neq \emptyset$ and $\eta_p(Q_p) \in \mathsf{eqc}_t(X_t)$. The other inclusion can be argued similarly.

Now suppose that $Q_t \in \mathcal{P}_t$. Then, for some $\{o, p\} = \{r, s\}$, $Q_t = \eta_p(Q_p)$ for some $Q_p \in \mathcal{P}_p \setminus \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$ with $\mathcal{Q}_o \subseteq N_{H_t}(Q_p)$. In other words, there is a vertex $v \in Q_p$ that is complete to X_t, and $\eta_p(Q_p) \notin \mathsf{eqc}_t(X_t)$. According to the definition of a profile, $Q_t = \eta_p(Q_p)$ is contained in the second coordinate of π_t. The other inclusion can be shown similarly. ☐

Now we show the other direction, i.e. that if we have a potentially bad clique for some $C_t \subseteq V_t$ in G_t, then its restrictions to V_r and V_s necessarily also form potentially bad cliques for the restriction of C_t to V_r and V_s in G_r and G_s, respectively. Furthermore, in that case, the profiles of the resulting cliques are compatible.

Lemma 4.5. *Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s). Let $C_t \subseteq V_t$, and let X_t be a clique in G_t that is potentially bad for C_t. For all $p \in \{r, s\}$, let $X_p := X_t \cap V_p$ and $C_p := C_t \cap V_p$. Suppose that for all $p \in \{r, s\}$, $X_p \neq \emptyset$. Then, for all $p \in \{r, s\}$, X_p is a potentially bad clique for C_p, and $\pi_r := \pi(X_r \mid C_r)$ and $\pi_s := \pi(X_s \mid C_s)$ are compatible.*

Proof. Since X_t is a potentially bad clique for C_t, we have that $X_t \subseteq C_t$, and so for $p \in \{r, s\}$, $X_p \subseteq C_p$. It remains to show that X_p is eqc-maximal for all $p \in \{r, s\}$. Up to renaming, it suffices to show that X_r is eqc-maximal. Suppose not and let $y \in \mathsf{eqc}_r(X_r)$ be a vertex that is complete to X_r. Since X_t is a clique in G_t, we have that $\mathsf{eqc}_r(X_r) \cup \mathsf{eqc}_s(X_s)$ is a biclique in H_t. Therefore, y is also complete to X_s and therefore to X_t. Clearly, $y \in \mathsf{eqc}_t(X_t)$, and we have a contradiction with X_t being eqc-maximal.

What remains to be shown is that $\pi_r = (\mathcal{Q}_r, \mathcal{P}_r)$ and $\pi_s = (\mathcal{Q}_s, \mathcal{P}_s)$ are compatible. We have already argued that $\mathcal{Q}_r \cup \mathcal{Q}_s = \mathsf{eqc}_r(X_r) \cup \mathsf{eqc}_s(X_s)$ is a biclique in H_t; we have to show that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is a maximal biclique in $H'_t := H'_t(\pi_r, \pi_s)$ as defined in Equation (1). Clearly, $\mathcal{Q}_r \cup \mathcal{Q}_s \subseteq V(H'_t)$, so suppose that $\mathcal{Q}_r \cup \mathcal{Q}_s$ is not a maximal biclique in H'_t. This means that for some $p \in \{r, s\}$, there is some $Q_p \in \mathcal{P}_p \cap \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$ such that $\{Q_p\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique in H'_t. In that case, there is a vertex $y \in Q_p$ that is complete to X_t (since $Q_p \in \mathcal{P}_p$ and $\{Q_p\} \cup \mathcal{Q}_r \cup \mathcal{Q}_s$ is a biclique), and $y \in V(\mathsf{eqc}_t(X_t))$ (since $Q_p \in \mathsf{bb}_t(\mathcal{Q}_r \cup \mathcal{Q}_s)$); we obtained a contradiction with X_t being eqc-maximal. ☐

As mentioned above, we treat the case when a clique X_p in one of the children $p \in \{r, s\}$ remains potentially bad in G_t separately. This is because in that case, the notion of a maximal biclique in H'_t as defined in Equation (1) does not hold up very naturally. We formulate the analogous requirements for this case here, and we skip some of the details.

Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s and operator (H_t, η_r, η_s). Let $\pi_r \in \Pi_r$. We say that $\pi_r = (\mathcal{Q}_r, \mathcal{P}_r)$ is *liftable* if

- there is no $Q_s \in \mathsf{bb}_t(\mathcal{Q}_r)$ that is complete to $\mathcal{Q}_r$ in H_t, and
- $\mathsf{bb}_t(\mathcal{Q}_r) \cap \mathcal{P}_r = \emptyset$.

The *lift profile* of π_r, denoted by $\lambda(\pi_r)$, is constructed as the merge profile of π_r with the empty set; i.e. we take $(\mathcal{Q}_s, \mathcal{P}_s) = (\emptyset, V_s/{\sim_s})$ and apply the definition given above, meaning $\lambda(\pi_r) = \mu(\pi_r, (\emptyset, V_s/{\sim_s}))$.

Lemma 4.6. *Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s. Let $C_r \subseteq V_r$, $C_s \subseteq V_s$, let X_r be a clique in G_r, and let $\pi_r := \pi(X_r \mid C_r)$. Then, X_r is a potentially bad clique for $C_r \cup C_s$ in G_t if and only if X_r is a potentially bad clique for C_r in G_r and π_r is liftable, in which case $\pi_t(X_r \mid C_r \cup C_s) = \lambda(\pi_r)$.*

Proof. The proof can be done with very similar arguments to those given above and is therefore omitted. One only needs to observe that the notion of 'liftable' modulates the notion of a profile being compatible with the profile of an empty set. □

4.2 The type of a color class

We now describe the t-type of a color class C, which is the subset of profiles at t such that there is a clique in G_t that is potentially bad for C, with that C-profile. For our algorithm, two color classes with the same type will be interchangeable, therefore we only have to remember the number of color classes of each type.

Definition 4.7 (t-Type). Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, and let $t \in V(T)$. For a set $C \subseteq V_t$, the *t-type* of C, denoted by $\gamma_t(C)$ is

$$\begin{aligned}\gamma_t(C) := \{\pi_t \in \Pi_t \mid\ & \exists \text{ clique } X \text{ in } G_t \text{ which is potentially bad for } C \\ & \text{and } \pi(X \mid C) = \pi_t\}.\end{aligned}$$

With slight abuse of notation, we call the set $\Gamma_t = 2^{\Pi_t}$ of all subsets of profiles at t the *t-types*.

Since for each $t \in V(T)$, $|\Pi_t| \leq 2^{\mathcal{O}(w)}$ by Observation 4.3, the number of t-types can be upper bounded as follows.

Observation 4.8. Let $(T, \mathcal{L})$ be a rooted branch decomposition, and let $t \in V(T)$. There are at most $2^{2^{\mathcal{O}(w)}}$ many t-types, where $w := \mathsf{mw}(T, \mathcal{L})$.

In our algorithm we want to be able to determine the t-type of the union of a color class in G_r and a color class in G_s. This is done via the following notion of a merge type, which is based on the notion of merge and lift profiles given in the previous section.

Definition 4.9 (Merge Type). Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. For a pair of an r-type $\gamma_r \in \Gamma_r$ and an s-type $\gamma_s \in \Gamma_s$, the *merge type* of γ_r and γ_s, denoted by $\mu(\gamma_r, \gamma_s)$, is the t-type obtained as follows.

$$\begin{aligned}\mu(\gamma_r, \gamma_s) :=\ & \{\mu(\pi_r, \pi_s) \mid \pi_r \in \gamma_r,\ \pi_s \in \gamma_s, \text{where } \pi_r \text{ and } \pi_s \text{ are compatible}\} \\ & \bigcup\nolimits_{p \in \{r,s\}} \{\lambda(\pi_p) \mid \pi_p \in \gamma_p, \text{where } \pi_p \text{ is liftable}\}\end{aligned}$$

Lemma 4.10. *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. Let $C_r \subseteq V_r$ and $C_s \subseteq V_s$. Then, $\gamma_t(C_r \cup C_s) = \mu(\gamma_r(C_r), \gamma_s(C_s))$.*

Proof. Let $C_t := C_r \cup C_s$. For one inclusion, let $\pi_t \in \gamma_t(C_t)$. Then, there is a clique X_t in G_t that is potentially bad for C_t whose C_t-profile is π_t. If for all $p \in \{r, s\}$, $X_p := X_t \cap V_p \neq \emptyset$, then by Lemma 4.5, we know that for all $p \in \{r, s\}$, X_p is a potentially bad clique for $C_p := C_t \cap V_p$, therefore $\pi_p := \pi(X_p \mid C_p) \in \gamma_p(C_p)$. Moreover, the lemma asserts that π_r and π_s are compatible, so by construction, we can conclude that $\pi_t = \mu(\pi_r, \pi_s) \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. On the other hand, if for some $p \in \{r, s\}$, $X_t \subseteq V_p$, then by Lemma 4.6, X_t is a potentially bad clique for C_p, so

$\pi_p := \pi_p(X_t \mid C_p) \in \gamma(C_p)$. The lemma also asserts that π_p is liftable and that $\lambda(\pi_r) = \pi_t$, in which case we also have that $\pi_t \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. We have argued that $\gamma_t(C_t) \subseteq \mu(\gamma_r(C_r), \gamma_s(C_s))$.

For the other inclusion, suppose that $\pi_t \in \mu(\gamma_r(C_r), \gamma_s(C_s))$. Then, either there is a pair of profiles $\pi_r \in \gamma_r(C_r)$, $\pi_s \in \gamma_s(C_s)$ such that π_r and π_s are compatible and $\pi_t = \mu(\pi_r, \pi_s)$ or for some $p \in \{r, s\}$, there is a profile $\pi_p \in \gamma_p(C_p)$ that is liftable and $\pi_t = \lambda(\pi_p)$. In the former case, we can use Lemma 4.4 to conclude that $\pi_t \in \gamma_t(C_t)$, and in the latter case, we have that $\pi_t \in \gamma_t(C_t)$ by Lemma 4.6. This shows that $\mu(\gamma_r(C_r), \gamma_s(C_s)) \subseteq \gamma_t(C_t)$ which concludes the proof. $\square$

4.3 The algorithm

We are now ready to describe the algorithm. As alluded to above, partial solutions at a node t, i.e. colorings of G_t, are described via the notion of a *t-signature* which records the number of color classes of each type in a coloring. If two colorings have the same t-signature, then they are interchangeable as far as our algorithm is concerned. We show that this information suffices to solve the problem in a bottom-up dynamic programming fashion.

Definition 4.11 (t-Signature). Let k be a positive integer. Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, let $t \in V(T)$, and let $\mathcal{C} = (C_1, \ldots, C_k)$ be a k-coloring of G_t. Then, $\sigma_{\mathcal{C}} \colon \Gamma_t \to \{0, 1, \ldots, k\}$ where

$$\forall \gamma_t \in \Gamma_t \colon \sigma_{\mathcal{C}}(\gamma_t) := |\{i \in \{1, \ldots, k\} \mid \gamma_t(C_i) = \gamma_t\}|,$$

is called the *t-signature* of $\mathcal{C}$. The set of *t-signatures* is defined as:

$$\mathsf{sig}_t := \left\{\sigma_t \colon \Gamma_t \to \{0, 1, \ldots, k\} \;\middle|\; \sum\nolimits_{\gamma_t \in \Gamma_t} \sigma_t(\gamma_t) = k\right\}$$

The following bound on the number of t-signatures immediately follows from Observation 4.8, stating that the number of t-types is upper bounded by $2^{2^{\mathcal{O}(w)}}$.

Observation 4.12. Let $(T, \mathcal{L})$ be a rooted branch decomposition of an n-vertex graph, and let $t \in V(T)$. There are at most $k^{2^{2^{\mathcal{O}(w)}}}$ many t-signatures, where $w := \mathsf{mw}(T, \mathcal{L})$ and k is the number of colors.

Definition of the table entries. For each $t \in V(T)$ and $\sigma_t \in \mathsf{sig}_t$, we let $\mathsf{tab}[t, \sigma_t] = 1$ if and only if there is a k-coloring $\mathcal{C}$ of G_t such that $\sigma_{\mathcal{C}} = \sigma_t$.

We now show that the information stored at the table entries suffices to determine whether or not our input is a Yes-instance; that is, after filling all the table entries, we can read off the solution to the problem at the root node.

Lemma 4.13. *Let G be a graph with rooted branch decomposition $(T, \mathcal{L})$, and let $\mathfrak{r}$ be the root of T. G has a clique coloring with k colors if and only if $\mathsf{tab}[\mathfrak{r}, \sigma^\star] = 1$, where $\sigma^\star$ is the $\mathfrak{r}$-signature for which $\sigma^\star(\emptyset) = k$.*

Proof. The lemma immediately follows from two facts. First, since $\sigma^\star(\emptyset) = k$, we have that $\sigma^\star(\gamma_{\mathfrak{r}}) = 0$ for any other $\mathfrak{r}$-type $\gamma_{\mathfrak{r}} \neq \emptyset$. Second, that for each set $C \subseteq V_{\mathfrak{r}} = V(G)$, the set of potentially bad cliques for C is precisely the set of maximal cliques that are fully contained in C, i.e. it is the set of monochromatic maximal cliques in the corresponding coloring that are contained in C. $\square$

We first describe how to compute the table entries at the leaves, by brute-force.

foreach $\sigma_t \in \mathsf{sig}_t$ **do** set $\mathsf{tab}[t, \sigma_t] \leftarrow 0$;
Let $(\mathfrak{J}, \mathfrak{m})$ be the merge skeleton of t;
foreach $\sigma_r \in \mathsf{sig}_r$, $\sigma_s \in \mathsf{sig}_s$ *such that* $\mathsf{tab}[r, \sigma_r] = 1$ *and* $\mathsf{tab}[s, \sigma_s] = 1$ **do**
  foreach $\mathfrak{n} \colon E(\mathfrak{J}) \to \{0, 1, \ldots, k\}$ *such that*
    (i) $\sum_{e \in E(\mathfrak{J})} \mathfrak{n}(e) = k$, *and*
    (ii) for all $p \in \{r, s\}$ *and all* $\gamma_p \in \Gamma_p$, *it holds that* $\sum_{\gamma_p \gamma_o \in E(\mathfrak{J})} \mathfrak{n}(\gamma_p \gamma_o) = \sigma_p(\gamma_p)$
  do
    Let $\sigma_t \colon \Gamma_t \to \{0, 1, \ldots, k\}$ be such that for all $\gamma_t \in \Gamma_t$, $\sigma_t(\gamma_t) = \sum_{e \in E(\mathfrak{J}), \mathfrak{m}(e) = \gamma_t} \mathfrak{n}(e)$;
    update $\mathsf{tab}[t, \sigma_t] \leftarrow 1$;

Algorithm 2: Algorithm to set the table entries at an internal node $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s, assuming the table entries at r and s have been computed.

Leaves of T. Let $t \in \mathrm{L}(T)$ be a leaf node in T and let $v \in V(G)$ be the vertex such that $\mathcal{L}(v) = t$. We show how to compute the table entries $\mathsf{tab}[t, \cdot]$. Note that $G_t = (\{v\}, \emptyset)$, and that $\{v\}$ is the only equivalence class of $\sim_t$. To describe the types of color classes of G_t, observe that the only eqc-maximal clique in G_t is $\{v\} =: X_v$, which is potentially bad for $C_v := \{v\} = X_v$. In that case, we have that $\pi_v := \pi(X_v \mid C_v) = (\{v\}, \emptyset)$, and the type of color class C_v is $\{\pi_v\}$. The type of the remaining $k - 1$ color classes is $\emptyset$, since they are all empty. Therefore, for each t-signature σ_t, we set $\mathsf{tab}[t, \sigma_t] := 1$ if and only if $\sigma_t(\{\pi_v\}) = 1$ and $\sigma_t(\emptyset) = k - 1$.

Next, we move on to the computation of the table entries at internal nodes of the branch decomposition. To describe this part of the algorithm, we borrow the following notion of a *merge skeleton* from [23].[4]

Definition 4.14 (Merge skeleton). Let G be a graph and $(T, \mathcal{L})$ one of its rooted branch decompositions. Let $t \in V(T) \setminus \mathrm{L}(T)$ with children r and s. The *merge skeleton* of r and s is a pair $(\mathfrak{J}, \mathfrak{m})$, where $\mathfrak{J}$ is a complete bipartite graph and $\mathfrak{m} \colon E(\mathfrak{J}) \to \Gamma_t$ is an edge-labeling of $\mathfrak{J}$ with

- $V(\mathfrak{J}) = \Gamma_r \cup \Gamma_s$, and
- for all $\gamma_r \in \Gamma_r$, $\gamma_s \in \Gamma_s$, $\mathfrak{m}(\gamma_r \gamma_s) = \mu(\gamma_r, \gamma_s)$.

Internal nodes of T. Let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s. We discuss how to compute the table entries at t, assuming the table entries at r and s have been computed. Each coloring of G_t can be obtained from a coloring of G_r and a coloring of G_s, by merging pairs of color classes. Therefore, for each pair $\sigma_r \in \mathsf{sig}_r$, $\sigma_s \in \mathsf{sig}_s$ such that $\mathsf{tab}[r, \sigma_r] = 1$ and $\mathsf{tab}[s, \sigma_s] = 1$, we do the following. We enumerate all labelings of the edge set of the merge skeleton with numbers from $\{0, 1, \ldots, k\}$, with the following interpretation. If an edge $\gamma_r \gamma_s$ has label j, then it means that j color classes of r-type γ_r will be merged with j color classes of s-type γ_s; this gives j color classes of t-type $\mu(\gamma_r, \gamma_s) = \mathfrak{m}(\gamma_r \gamma_s)$. Each such labeling that respects the number of color classes available of each type will produce a coloring of G_t with some signature σ_t, which can then be read off the edge labeling. For all such σ_t, we set $\mathsf{tab}[t, \sigma_t] = 1$. We give the formal details in Algorithm 2.

We now prove the correctness of the algorithm.

[4]Note that in [23], the graph structure of the bipartite graph plays a role, in that there is only edges between compatible types. In the present setting, there is no notion of compatibility of color class types which is why the bipartite graph of the merge skeleton is always complete.

Lemma 4.15. *Let G be a graph and $(T, \mathcal{L})$ one of its rooted branch decompositions, and let $t \in V(T)$. The above algorithm computes the table entries $\mathsf{tab}[t, \cdot]$ correctly, i.e. for each $\sigma_t \in \mathsf{sig}_t$, it sets $\mathsf{tab}[t, \sigma_t] = 1$ if and only if G_t has a k-coloring $\mathcal{C}$ with $\sigma_{\mathcal{C}} = \sigma_t$.*

Proof. The proof is by induction on the height of t. In the base case, when t is a leaf, it is straightforward to verify correctness.

Now suppose that $t \in V(T) \setminus \mathrm{L}(T)$ is an internal node with children r and s, and let $(\mathfrak{J}, \mathfrak{m})$ be the merge skeleton at t. Suppose for some t-signature $\sigma_t \in \mathsf{sig}_t$, the algorithm set $\mathsf{tab}[t, \sigma_t] = 1$. Then, there is some r-signature σ_r and some s-signature σ_s such that $\mathsf{tab}[r, \sigma_r] = 1$, $\mathsf{tab}[s, \sigma_s] = 1$, and there is a map $\mathfrak{n} \colon E(\mathfrak{J}) \to \{0, 1, \ldots, k\}$ satisfying the conditions of lines 4 and 6 in Algorithm 2. By induction, there is a k-coloring $\mathcal{C}_r$ of G_r whose r-signature is σ_r, and a k-coloring of $\mathcal{C}_s$ of G_s whose s-signature is σ_s. We construct the desired coloring $\mathcal{C}_t$ of G_t whose t-signature is σ_t as follows: For each pair of an r-type γ_r and an s-type γ_s, we take $\mathfrak{n}(\gamma_r\gamma_s)$ pairs of a color class C_r of r-type σ_r and a color class C_s of s-type C_s, and for each such pair, we add $C_r \cup C_s$ as a color class to $\mathcal{C}_t$. By Lemma 4.10, the t-type of $C_r \cup C_s$ is $\mu(\gamma_r, \gamma_s) = \mathfrak{m}(\gamma_r\gamma_s)$. The condition in line 4 ensures that each color class of $\mathcal{C}_r$ and each color class of $\mathcal{C}_s$ is used precisely once to create a color class of $\mathcal{C}_t$ (which also implies that $\mathcal{C}_t$ has k color classes), and the condition in line 6 ensures that the t-signature of $\mathcal{C}_t$ is indeed σ_t.

For the other direction, suppose that there is a k-coloring $\mathcal{C}_t$ of G_t with t-signature σ_t. We construct a pair of a coloring $\mathcal{C}_r$ of G_r and a coloring of $\mathcal{C}_s$ of G_s, together with their signatures σ_r and σ_s, respectively, and a map $\mathfrak{n} \colon E(\mathfrak{J}) \to \{0, 1, \ldots, k\}$. Initially, for all $p \in \{r, s\}$, we let $\mathcal{C}_p = \emptyset$, and for all $\gamma_p \in \Gamma_p$, $\sigma_p(\gamma_p) \coloneqq 0$. Moreover, we let $\mathfrak{n}(e) \coloneqq 0$ for all $e \in E(\mathfrak{J})$.

For each color class $C_t \in \mathcal{C}_t$, we add $C_r \coloneqq C_t \cap V_r$ to $\mathcal{C}_r$ and $C_s \coloneqq C_t \cap V_s$ to $\mathcal{C}_s$. Let γ_t be the t-type of C_t. By Lemma 4.10, C_r has some r-type γ_r and C_s has some s-type γ_s such that γ_t is the merge type $\mu(\gamma_r, \gamma_s)$ of γ_r and γ_s. We increase the values of $\sigma_r(\gamma_r)$ and $\sigma_s(\gamma_s)$ by 1, since we added one more color class of r-type γ_r to $\mathcal{C}_r$, and one more color class of s-type γ_s to $\mathcal{C}_s$. Additionally, we add 1 to the value of $\mathfrak{n}(\gamma_r\gamma_s)$, since C_t is a color class of t-type $\mu(\gamma_r, \gamma_s) = \mathfrak{m}(\gamma_r\gamma_s)$ obtained from merging C_r (a color class of r-type γ_r) with C_s (a color class of s-type γ_s).

After doing this for all color classes of $\mathcal{C}_t$, we have that $\mathcal{C}_r$ is a k-coloring with r-signature σ_r, and that $\mathcal{C}_s$ is a k-coloring with s-signature σ_s. By induction, $\mathsf{tab}[r, \sigma_r] = 1$ and $\mathsf{tab}[s, \sigma_s] = 1$. It remains to argue that $\mathfrak{n}$ satisfies the conditions expressed in lines 4 and 6 in Algorithm 2. The first item of line 4 is clearly satisfied, since we increased $|\mathcal{C}_t| = k$ values of $\mathfrak{n}$ by 1 in the above process. The second item holds since we increased the value of some $\sigma_p(\gamma_p)$ by 1 if and only if we increased the value of an edge e incident with γ_p in $\mathfrak{J}$ by 1. To see that for each γ_t, $\sigma_t(\gamma_t) = \sum_{e \in E(\mathfrak{J}), \mathfrak{m}(e) = \gamma_t} \mathfrak{n}(e)$, observe that we identified for each color class of type γ_t, the occurrence of γ_t as a merge type of a pair of an r-type and an s-type, and therefore a label of some edge $e \in E(\mathfrak{J})$, and increased $\mathfrak{n}(e)$ by 1 in such a case. We can conclude that σ_t can be obtained as shown in line 6 of Algorithm 2, and so the algorithm set $\mathsf{tab}[t, \sigma_t] = 1$. $\square$

To wrap up, it remains to argue the runtime of the algorithm. Suppose we are given a graph G with rooted branch decomposition $(T, \mathcal{L})$ and let $w \coloneqq \mathsf{mw}(T, \mathcal{L})$. By Observation 4.12, there are at most $k^{2^{2^{\mathcal{O}(w)}}}$ table entries at each node of T. The entries of leaf nodes can clearly be computed in constant time. Now let $t \in V(T) \setminus \mathrm{L}(T)$ be an internal node with children r and s. To compute all table entries at t, we execute Algorithm 2. In the worst case, it loops over each pair of an r-signature and an s-signature, and given such a pair, it enumerates all labelings of the edges of the merge skeleton $\mathfrak{J}$ with numbers from $\{0, 1, \ldots, k\}$ (such that all entries sum up to k). We have that $|E(\mathfrak{J})| = |\Gamma_r| \cdot |\Gamma_s| = \left(2^{2^{\mathcal{O}(w)}}\right)^2 = 2^{2^{\mathcal{O}(w)}}$ (see Observation 4.8), therefore the number of labelings to consider is upper bounded by $k^{2^{2^{\mathcal{O}(w)}}}$. The runtime of Algorithm 2 can therefore be upper bounded by

$$\left(k^{2^{2^{\mathcal{O}(w)}}}\right)^2 \cdot k^{2^{2^{\mathcal{O}(w)}}} = k^{2^{2^{\mathcal{O}(w)}}},$$

and since $|V(T)| = \mathcal{O}(n)$, the runtime of the whole procedure is $k^{2^{2^{\mathcal{O}(w)}}} \cdot n$. Correctness is proved in Lemma 4.15, and Lemma 4.13 asserts that the solution to the problem can be read off the table entries at the root, once computed. Using standard memoization techniques, we can modify the above algorithm so that it returns a coloring if one exists. Lastly, we observe that we may assume that $k < n$. For if $k \geq n$, then the input instance is a trivial YES-instance: we can simply assign each vertex of the input graph G a distinct color; this clearly results in a clique coloring of G. We have the following theorem.

Theorem 4.16. *There is an algorithm that given an n-vertex graph G together with one of its rooted branch decompositions $(T, \mathcal{L})$ and a positive integer k, decides whether G has a clique coloring with k colors in time $k^{2^{2^{\mathcal{O}(w)}}} \cdot n \leq n^{2^{2^{\mathcal{O}(w)}}}$, where $w := \mathsf{mw}(T, \mathcal{L})$. If such a coloring exists, the algorithm can construct it.*

5 Conclusion

In this work, we considered structural parameterizations of the CLIQUE COLORING problem by two of the most commonly used width measures of graphs: treewidth and clique-width. We showed that for fixed number of colors $q \geq 2$, q-CLIQUE COLORING can be solved in time $\mathcal{O}^\star(q^{\mathsf{tw}})$, where tw denotes the width of a given tree decomposition of the input graph, and that under SETH, there is no such algorithm running in time $\mathcal{O}^\star((q-\epsilon)^{\mathsf{tw}})$, for any $\epsilon > 0$. Regarding the clique-width parameterization, we gave a $k^{2^{2^{\mathcal{O}(\mathsf{cw})}}} \cdot n$ time algorithm, where k is the requested number of colors and the input graph is given together with a clique-width cw-expression. We would like to end this work by recalling and explicitly stating the open questions from the introduction that are raised by the clique-width based algorithm. First, we would be interested in the coarse parameterized complexity of this problem.

Open Problem 1. Is CLIQUE COLORING parameterized by clique-width W[1]-hard?

Arguably the most promising route to an FPT-algorithm for CLIQUE COLORING parameterized by clique-width (if it exists) is via a proof that the number of colors that is needed in any clique coloring is upper bounded in terms of some function of the clique-width of a graph.

Open Problem 2. Is there a function $g\colon \mathbb{N} \to \mathbb{N}$ such that each graph G can be clique colored with at most $g(\mathsf{cw})$ colors, where cw denotes the clique-width of G?

Next, it would be interesting to see if the triple-exponential dependence on cw of our algorithm can be avoided – both in the case when the number of colors is unbounded and when the number of colors is bounded.

Open Problem 3. Is there an algorithm for CLIQUE COLORING running in time $n^{2^{2^{o(\mathsf{cw})}}}$, or for fixed $q \geq 2$, an algorithm for q-CLIQUE COLORING running in time $q^{2^{2^{o(\mathsf{cw})}}} \cdot n^{\mathcal{O}(1)}$, when the input graph is given together with a clique-width cw-expression, or would any such algorithm violate ETH?

Acknowledgements. We would like to thank the anonymous reviewers for comments that helped improving the presentation of this paper.

References

[1] Thomas Andreae, Martin Schughart, and Zsolt Tuza. Clique-transversal sets of line graphs and complements of line graphs. *Discrete Mathematics*, 88(1):11–20, 1991.

[2] Gábor Bacsó, Sylvain Gravier, András Gyárfás, Myriam Preissmann, and András Sebo. Coloring the maximal cliques of graphs. *SIAM Journal on Discrete Mathematics*, 17(3):361–376, 2004.

[3] Gábor Bacsó and Zsolt Tuza. Clique-transversal sets and weak 2-colorings in graphs of small maximum degree. *Discrete Mathematics and Theoretical Computer Science*, 11(2):15–24, 2009.

[4] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 67–74. ACM, 2007.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[6] C. N. Campos, Simone Dantas, and Célia Picinin de Mello. Colouring clique-hypergraphs of circulant graphs. *Electronic Notes in Discrete Mathematics*, 30:189–194, 2008.

[7] Márcia R. Cerioli and André L. Korenchendler. Clique-coloring circular-arc graphs. *Electronic Notes in Discrete Mathematics*, 35:287–292, 2009.

[8] Pierre Charbit, Irena Penev, Stéphan Thomassé, and Nicolas Trotignon. Perfect graphs of arbitrarily large clique-chromatic number. *Journal of Combinatorial Theory, Series B*, 116:456–464, 2016.

[9] Maria Chudnovsky and Irene Lo. Decomposing and clique-coloring (diamond, odd-hole)-free graphs. *Journal of Graph Theory*, 86(1):5–41, 2017.

[10] Manfred Cochefert and Dieter Kratsch. Exact algorithms to clique-colour graphs. In Viliam Geffert, Bart Preneel, Branislav Rovan, Julius Stuller, and A Min Tjoa, editors, *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2014)*, volume 8327 of *LNCS*, pages 187–198. Springer, 2014.

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

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

[13] 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 Transactions on Algorithms*, 12(3):41:1–41:24, 2016.

[14] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[15] David Défossez. Clique-coloring some classes of odd-hole-free graphs. *Journal of Graph Theory*, 53(3):233–249, 2006.

[16] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[17] Dwight Duffus, Bill Sands, Norbert Sauer, and Robert E. Woodrow. Two-colouring all two-element maximal antichains. *Journal of Combinatorial Theory, Series A*, 57(1):109–116, 1991.

[18] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

[19] 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 Transactions on Algorithms*, 15(1):9:1–9:27, 2019.

[20] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[21] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[22] Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC 2017)*, volume 10236 of *LNCS*, pages 345–356. Springer, 2017.

[23] Lars Jaffke, Paloma T. Lima, and Daniel Lokshtanov. b-Coloring parameterized by clique-width. In Markus Bläser and Benjamin Monmege, editors, *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *LIPIcs*, pages 43:1–43:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[24] Sulamita Klein and Aurora Morgana. On clique-colouring of graphs with few P_4's. *Journal of the Brazilian Computer Society*, 18(2):113–119, 2012.

[25] Ton Kloks. *Treewidth: Computations and approximations*, volume 842 of *LNCS*. Springer, 1994.

[26] Jan Kratochvíl and Zsolt Tuza. On the complexity of bicoloring clique hypergraphs of graphs. *Journal of Algorithms*, 45(1):40–54, 2002.

[27] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Transactions on Algorithms*, 14(2):13:1–13:30, 2018.

[28] Lásló Lovász. *Combinatorial Problems and Exercises*. North-Holland Publishing Co., 1993.

[29] Dániel Marx. Complexity of clique coloring and related problems. *Theoretical Computer Science*, 412(29):3487–3500, 2011.

[30] Bojan Mohar and Riste Škrekovski. The Grötzsch theorem for the hypergraph of maximal cliques. *Electronic Journal of Combinatorics*, 6(1):128, 1999.

[31] Irena Penev. Perfect graphs with no balanced skew-partition are 2-clique-colorable. *Journal of Graph Theory*, 81(3):213–235, 2016.

[32] Michaël Rao. *Décompositions de graphes et algorithmes efficaces.* PhD thesis, University of Metz, 2006.

[33] Michaël Rao. Clique-width of graphs defined by one-vertex extensions. *Discrete Mathematics*, 308(24):6157–6165, 2008.

[34] Erfang Shan, Zuosong Liang, and Liying Kang. Clique-transversal sets and clique-coloring in planar graphs. *European Journal of Combinatorics*, 36:367–376, 2014.

[35] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *LIPIcs*, pages 17–29. Schloss Dagstuhl, 2015.