Vertex Partitioning Problems on Graphs with Bounded Tree Width*

N. R. Aravind[1], Subrahmanyam Kalyanasundaram[1], and Anjeneya Swami Kare[†2]

[1]Indian Institute of Technology, Hyderabad, India,
{aravind,subruk}@iith.ac.in
[2]University of Hyderabad, India, askcs@uohyd.ac.in

Abstract

In an undirected graph, a matching cut is a partition of vertices into two sets such that the edges across the sets induce a matching. The Matching Cut problem is the problem of deciding whether a given graph has a matching cut.

Let H be a fixed undirected graph. A vertex coloring of an undirected input graph G is said to be an H-Free Coloring if none of the color classes contain H as an induced subgraph. The H-Free Chromatic Number of G is the minimum number of colors required for an H-Free Coloring of G.

Both The Matching Cut problem and the H-Free Coloring problem can be expressed using a monadic second-order logic (MSOL) formula and hence is solvable in linear time for graphs with bounded tree-width. However, this approach leads to a running time of $f(||\varphi||, t)n^{O(1)}$, where $||\varphi||$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph. The dependency of $f(||\varphi||, t)$ on $||\varphi||$ can be as bad as a tower of exponentials.

In this paper, we provide an explicit combinatorial FPT algorithms for Matching Cut problem and H-Free Coloring problem, parameterized by the tree-width of G. The techniques are also used to provide an FPT algorithm when H is forbidden as a subgraph (not necessarily induced) in the color classes of G.

1 Introduction

Consider an undirected graph $G = (V, E)$ such that $|V| = n$. An *edge cut* is an edge set $S \subseteq E$ such that the removal of S from the graph increase the number of components in the graph. A *matching* is an edge set such that no two edges in the set have a common end point. A *matching cut* is an edge cut which is also

*This manuscript is a combination of the works [1, 2] which appeared in the proceedings of COCOA-2017 and CALDAM-2019 respectively.

†This work was done while the author was a Ph.D. student at IIT Hyderabad, India.

a matching. The *matching cut* problem is the decision problem of determining whether a given graph G has a matching cut.

The matching cut problem was first introduced by Graham in [3], in the name of *decomposable graphs*. Farley and Proskurowski [4] pointed out the applications of the matching cut problem in computer networks – in studying the networks which are immune to failures of non-adjacent links. Patrignani and Pizzonia [5] pointed out the applications of the matching cut problem in graph drawing. They refer to a method of graph drawing, where one starts with a degenerate drawing where all the vertices and edges are at the same point. At each step, the vertices in the drawing are partitioned and progressively the drawing approaches the original graph. In this regard, the cut involving the non-adjacent edges (matching cut) yields a more efficient and effective performance.

The matching cut problem is NP-Complete for the following graph classes:

- Graphs with maximum degree 4 (Chvátal [6], Patrignani and Pizzonia [5]).
- Bipartite graphs with one partite set has maximum degree 3 and the other partite set has maximum degree 4 (Le and Randerath [7]).
- Planar graphs with maximum degree 4 and planar graphs with girth 5 (Bonsma [8]).
- $K_{1,4}$-free graphs with maximum degree 4 (inferred from the reduction in [6]).

The matching cut problem has polynomial time algorithms for the following graph classes:

- Graphs with maximum degree 3 (Chvátal [6]).
- Line graphs (Moshi [9]).
- Graphs without chordless cycles of length 5 or more (Moshi [9]).
- Series parallel graphs (Patrignani and Pizzonia [5]).
- Claw-free graphs, cographs, graphs with bounded tree-width and graphs with bounded clique-width (Bonsma [8]).
- Graphs with diameter 2 (Borowiecki and Jesse-Józefczyk [10]).
- $(K_{1,4}, K_{1,4} + e)$-free graphs (Kratsch and Le [11]).

When the graph G has degree at least 2, the matching cut problem in G is equivalent to the problem of deciding whether the line graph of G has a stable cut set. A *stable cut set* is a set $S \subseteq V$ of independent vertices, such that the removal of S from the graph G increases the number of components of G. Algorithmic aspects of stable cut set of line graphs have been studied in [7, 12, 13, 14].

Recently, Kratsch and Le [11] presented a $2^{n/2}n^{O(1)}$ time algorithm for the matching cut problem using branching techniques. They also showed that the matching cut problem is tractable for graphs with bounded vertex cover.

Let G be an undirected graph. The classical q-Coloring problem asks to color the vertices of the graph using at most q colors such that no pair of adjacent vertices are of the same color. The CHROMATIC NUMBER of the graph is the minimum number of colors required for q-coloring the graph and is denoted by $\chi(G)$. The graph coloring problem has been extensively studied in various settings.

In this paper we consider a generalization of the graph coloring problem called H-FREE q-COLORING which asks to color the vertices of the graph using at most q colors such that none of the color classes contain H as an induced subgraph. Here, H is any fixed graph, $|V(H)| = r$, for some fixed r. The H-FREE CHROMATIC NUMBER is the minimum number of colors required to H-free color the graph. Note that when $H = K_2$, the H-FREE q-COLORING problem is same as the classical q-COLORING problem.

For $q \geq 3$, H-FREE q-COLORING problem is NP-complete as the q-COLORING problem is NP-complete. The 2-COLORING problem is polynomial time solvable as it is equivalent to decide whether the graph is bipartite. The H-FREE 2-COLORING problem has been shown to be NP-complete as long as H has 3 or more vertices [15]. A variant of H-FREE COLORING problem which we call H-(SUBGRAPH)FREE q-COLORING which asks to color the vertices of the graph such that none of the color classes contain H as a subgraph (not necessarily induced) is studied in [16, 17].

Graph bipartitioning (2-coloring) problems with other constraints have been explored in the past. Many variants of 2-coloring have been shown to be NP-hard. Recently, Karpiński [18] studied a problem which asks to color the vertices of the graph using 2 colors such that there is no monochromatic cycle of a fixed length. The degree bounded bipartitioning problem asks to partition the vertices of G into two sets A and B such that the maximum degree in the induced subgraphs $G[A]$ and $G[B]$ are at most a and b respectively. Xiao and Nagamochi [19] proved that this problem is NP-complete for any non-negative integers a and b except for the case $a = b = 0$, in which case the problem is equivalent to testing whether G is bipartite. Other variants that place constraints on the degree of the vertices within the partitions have also been studied [20, 21]. Wu, Yuan and Zhao [22] showed the NP-completeness of the variant that asks to partition the vertices of the graph G into two sets such that both the induced graphs are acyclic. Farrugia [23] showed the NP-completeness of a problem called $(\mathcal{P}, \mathcal{Q})$-coloring problem. Here, $\mathcal{P}$ and $\mathcal{Q}$ are any additive induced-hereditary graph properties. The problem asks to partition the vertices of G into A and B such that $G[A]$ and $G[B]$ have properties $\mathcal{P}$ and $\mathcal{Q}$ respectively.

The MATCHING CUT problem, for a fixed q, the H-FREE q-COLORING problem can be expressed in monadic second order logic (MSOL) [24]. The MSOL formulation together with Courcelle's theorem [25, 26] implies linear time solvability on graphs with bounded tree-width. This approach yields an algorithm with running time $f(||\varphi||, t) \cdot n$, where $||\varphi||$ is the length of the MSOL formula, t is the tree-width of the graph and n is the number of vertices of the graph. The dependency of $f(||\varphi||, t)$ on $||\varphi||$ can be as bad as a tower of exponentials.

In this paper we present explicit combinatorial algorithms for the MATCHING CUT problem and H-FREE q-COLORING problem. We have the following results:

- a $2^{O(t)}n^{O(1)}$ algorithm for the matching cut problem, where t is the tree-width of the graph.

- $O(q^{4t^r} \cdot n)$ time algorithm for the H-Free q-Coloring problem for any arbitrary fixed graph H on r vertices.
- $O(2^{t+r\log t} \cdot n)$ time algorithm for K_r-Free 2-Coloring problem, where K_r is a complete graph on r vertices.
- $O(2^{3t^2} \cdot n)$ time algorithm for C_4-Free 2-Coloring problem, where C_4 is a cycle on 4 vertices.
- We also show that the matching cut problem is tractable for graphs with bounded neighborhood diversity and other structural parameters.

From the above we get the explicit FPT algorithm for H-Free Chromatic Number problem. The techniques can also be extended to obtain analogous results for the H-(Subgraph)Free q-Coloring.

2 Preliminaries

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed and finite alphabet. For $(x,k) \in \Sigma^* \times \mathbb{N}$, k is referred to as the parameter. A parameterized problem L is *fixed parameter tractable (FPT)* if there is an algorithm A, a computable non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ and a constant c such that, given $(x,k) \in \Sigma^* \times \mathbb{N}$ the algorithm A correctly decides whether $(x,k) \in L$ in time bounded by $f(k).|x|^c$.

Sometimes, we write $f(n) = O^*(g(n))$ if $f(n) = O(g(n)\text{poly}(n))$, where poly$(n)$ is a polynomial in n. Two vertices u, v are called *neighbors* if $\{u,v\} \in E$, we say v is a *neighbor* of u and vice versa. The set of all neighbors of u (*open neighborhood*) is denoted by $N(u)$. The *closed neighborhood* of u, is denoted by $N[u]$, is defined as $N[u] = N(u) \cup \{u\}$. For a vertex set $S \subseteq V$, the subgraph induced by S is denoted by $G[S]$. For a vertex set $S \subseteq V$, $G \backslash S$ denotes the graph $G[V \backslash S]$. When there is no ambiguity, we use the simpler notations $S \backslash x$ to denote $S \backslash \{x\}$ and $S \cup x$ to denote $S \cup \{x\}$.

For a vertex set $S \subseteq V$, the subgraph induced by S is denoted by $G[S]$. A graph G is said to be H-free if G does not have H as an induced subgraph. We follow the standard graph theoretic terminology from [27].

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed and finite alphabet. For $(x,k) \in \Sigma^* \times \mathbb{N}$, k is referred to as the parameter. A parameterized problem L is *fixed parameter tractable (FPT)* if there is an algorithm A, a computable non-decreasing function $f : \mathbb{N} \to \mathbb{N}$ and a constant c such that, given $(x,k) \in \Sigma^* \times \mathbb{N}$ the algorithm A correctly decides whether $(x,k) \in L$ in time bounded by $f(k).|x|^c$. For more details on parameterized algorithms refer to [28].

A *tree decomposition* of G is a pair $(T, \{X_i, i \in I\})$, where for $i \in I$, $X_i \subseteq V$ (usually called bags) and T is a tree with elements of I as the nodes such that:

1. For each vertex $v \in V$, there is an $i \in I$ such that $v \in X_i$.
2. For each edge $\{u,v\} \in E$, there is an $i \in I$ such that $\{u,v\} \subseteq X_i$.
3. For each vertex $v \in V$, $T[\{i \in I | v \in X_i\}]$ is connected.

The width of the tree decomposition is $\max_{i \in I}(|X_i| - 1)$. The tree-width of G is the minimum width taken over all tree decompositions of G and we denote it as t. For more details on tree-width, we refer the reader to [29]. A rooted tree decomposition is called a *nice tree decomposition*, if every node $i \in I$ is one of the following types:

1. Leaf Node: For a leaf node i, $X_i = \emptyset$.
2. Introduce Node: An introduce node i has exactly one child j and there is a vertex $v \in V \backslash X_j$ such that $X_i = X_j \cup \{v\}$.
3. Forget Node: A forget node i has exactly one child j and there is a vertex $v \in V \backslash X_i$ such that $X_j = X_i \cup \{v\}$.
4. Join Node: A join node i has exactly two children j_1 and j_2 such that $X_i = X_{j_1} = X_{j_2}$.

The notion of *nice tree decomposition* was introduced by Kloks [30]. Every graph G has a nice tree decomposition with $|I| = O(n)$ nodes and width equal to the tree-width of G. Moreover, such a decomposition can be found in linear time if the tree-width is bounded.

3 Matching Cut Problem parameterized by Tree-width

We present an $O^*(2^{O(t)})$ time algorithm for the matching cut problem. The algorithm we present is based on dynamic programming technique on the nice tree decomposition.

The matching cut problem is a graph partitioning problem, where we need to partition the vertices into two sets A and B such that the edges across the sets induce a matching. And we denote such a matching cut by (A, B). We use the following notation in the algorithm.

- i: A node in the tree decomposition.
- X_i: The set of vertices associated with bag at node i.
- $G[X_i]$: Subgraph induced by X_i.
- T_i: The sub-tree rooted at node i of the tree decomposition. This includes node i and all its descendants.
- $G[T_i]$: Subgraph induced by the vertices in node i and all its descendants.

Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i, we say that the partition Ψ is **legal** at node i if it satisfies the following conditions ($\star$):

1. Every vertex of A_1 (respectively B_1) has exactly one neighbor in B_1 (resp. A_1) and no neighbors in $B_2 \cup B_3$ (resp. $A_2 \cup A_3$).
2. Every vertex of $A_2 \cup A_3$ (resp. $B_2 \cup B_3$) has no neighbors in any of the B_i's (resp. A_i's).

We say that a legal partition ψ is **valid** for the node i if there exists a matching cut (A, B) of $G[T_i]$ such that the following conditions $(\star\star)$ hold:

1. The A_i's are contained in A and the B_i's are contained in B.
2. Every vertex of A_1 (resp. B_1) has a matching cut neighbor in B_1 (resp. A_1).
3. Every vertex of $A_2 \cup B_2$ has a matching cut neighbor in $G[T_i] \setminus X_i$.
4. The vertices of $A_3 \cup B_3$ are not part of the cut-edges, i.e. every vertex of A_3 (resp. B_3) has no neighbor in B (resp. A).

A matching cut is empty if there are no edges in cut. We say that a valid partition Ψ of X_i is *locally empty* in $G[T_i]$, if every matching cut of $G[T_i]$ extending ψ (i.e. satisfying $\star\star$) is empty. Note that, a necessary condition for Ψ to be locally empty is: $A_1 \cup A_2 \cup B_1 \cup B_2 = \emptyset$.

We define $M_i[\Psi]$ to be $+1$ if Ψ is valid for the node X_i and not locally empty, 0 if it is valid and locally empty, and -1 otherwise. Now, we explain how to compute $M_i[\Psi]$ for each partition Ψ at the nodes of the nice tree decomposition.

Leaf node: For a leaf node i, $X_i = \emptyset$. We have $\Psi = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ and $M_i[\Psi] = 0$. This step can be executed in constant time.

Introduce node: Let j be the only child of the node i. Suppose, $v \in X_i$ is the new node present in X_i, $v \notin X_j$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i. If Ψ is not legal, we straightaway set $M_i[\Psi]$ to -1. Otherwise, we use the below procedure to compute $M_i[\Psi]$ for $v \in A_i$, and analogously for $v \in B_i$.

Case 1: $v \in A_1$, then $M_i[\Psi] = +1$, if there exists a unique $x \in B_1$, such that, $(v, x) \in E$ and $M_j[\Psi'] \geq 0$ for $\Psi' = (A_1 \backslash v, A_2, A_3, B_1 \backslash x, B_2, B_3 \cup x)$. Otherwise $M_i[\Psi] = -1$. Note that, $M_i[\Psi]$ can not be 0, as $v \in A_1$ brings an edge into the cut if it is valid.

Case 2: $v \in A_2$, this case is not valid as v does not have any neighbor in $V(T_i) \backslash X_i$ (it is the property of the nice tree decomposition).

Case 3 $v \in A_3$, $M_i[\Psi] = M_j[\Psi']$ where $\Psi' = (A_1, A_2, A_3 \backslash v, B_1, B_2, B_3)$.

The total number of possible Ψ's for X_i is 6^{t+1}. For each Ψ, the above cases can be executed in polynomial time. Hence, the total time complexity at the introduce node is $O^*(6^t)$.

Forget node: Let j be the only child of the node i. Suppose, $v \in X_j$ is the node missing in X_i, $v \notin X_i$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i. If Ψ is not legal, we straightaway set $M_i[\Psi]$ to -1.

Otherwise, $M_i[\Psi] = \max_{k=1}^{k=6}\{\delta_k\}$, where δ_k is computed as follows: If Ψ is valid, it should be possible to add v to one of the six sets to get a valid partition at node j.

Case 1: v is in the first set at the node j. If there is a unique $x \in B_2$ such that $(v, x) \in E$ then $\delta_1 = M_j[\Psi']$ where $\Psi' = (A_1 \cup v, A_2, A_3, B_1 \cup x, B_2 \backslash x, B_3)$. If no such x exists, then δ_1 is set to -1.

Case 2: v is in the second set at the node j.
Let $\Psi' = (A_1, A_2 \cup v, A_3, B_1, B_2, B_3)$ and $\delta_2 = M_j[\Psi']$.

Case 3: v is in the third set at the node j.
Let $\Psi' = (A_1, A_2, A_3 \cup v, B_1, B_2, B_3)$ and $\delta_3 = M_j[\Psi']$.

The values δ_4, δ_5 and δ_6 are computed analogously. The total number of possible Ψ's for X_i is 6^t. For each Ψ, the above cases can be executed in polynomial time. Hence, the total time complexity at the forget node is $O^*(6^t)$.

Join node: Let j_1 and j_2 be the children of the node i. $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1})\backslash X_i$ and $V(T_{j_2})\backslash X_i$. Let $\Psi = (A_1, A_2, A_3, B_1, B_2, B_3)$ be a partition of X_i. For $X \subseteq A_2$ and $Y \subseteq B_2$ let $\Psi_1 = (A_1, X, A_3 \cup \{A_2 \backslash X\}, B_1, Y, B_3 \cup \{B_2 \backslash Y\})$ and $\Psi_2 = (A_1, A_2 \backslash X, A_3 \cup X, B_1, B_2 \backslash Y, B_3 \cup Y)$.

$$M_i[\Psi] = \begin{cases} +1, & \text{If } \exists X \subseteq A_2 \text{ and } Y \subseteq B_2 \text{ such that } M_{j_1}[\Psi_1] + M_{j_2}[\Psi_2] \geq 1; \\ 0, & \text{If } \Psi \text{ is locally empty, (i.e } M_{j_1}[\Psi] = 0 \text{ and } M_{j_2}[\Psi] = 0); \\ -1, & \text{Otherwise} \end{cases}$$

The total number of possible Ψ's for X_i is 6^{t+1}. For each Ψ, we need to check 2^{t+1} different Ψ_1 and Ψ_2. The total time complexity at the join node is $O^*(12^t)$.

At each node i, let $\Delta_i = \max_\Psi\{M_i[\Psi]\}$. If $\Delta_i = +1$, then $G[T_i]$ has a valid non-empty matching cut. If r is the root of the nice tree decomposition, the graph G has a matching cut if $\Delta_r = +1$. By induction and the correctness of $M_i[\Psi]$ values, we can conclude the correctness of the algorithm. The total time complexity of the algorithm is $O^*(12^t) = O^*(2^{O(t)})$.

Theorem 1. *There is an algorithm with running time $O^*(2^{O(t)})$ that solves the matching cut problem, where t is the tree-width of the graph.*

4 Matching Cut Problem parameterized by Neighborhood Diversity

Lampis [31] introduced a structural parameter called *neighborhood diversity* which is defined as follows:

Definition 1 (Neighborhood Diversity [31]). *In an undirected graph G, two vertices u and v have the same type if and only if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

The graph G has neighborhood diversity d if there exists a partition of $V(G)$ into d sets $P_1, P_2, \ldots, P_d$ such that all the vertices in each set have the same type. Such a partition is called a type partition. Moreover, it can be computed in linear time.

Note that, each P_i forms either a clique or an independent set in G.

If a graph has vertex cover number q, then the neighborhood diversity of the graph is at most $2^q + q$ [31]. Hence, graphs with bounded vertex cover number also have bounded neighborhood diversity. However, the converse is not true since complete graphs have neighborhood diversity 1. Some NP-hard problems are shown to be tractable on graphs with bounded neighborhood diversity (see e.g., [32]). Here, we show that the matching cut problem is tractable for graphs with bounded neighborhood diversity. We describe an algorithm with time complexity $O^*(2^{2d})$, where d is the neighborhood diversity of the graph.

We start with a graph G, and its type partitioning with d partitions, i.e neighborhood diversity of G is d. We label the vertices of G (using the type partitioning) such that vertices having the same label should be entirely on one side of the cut. We assume that the graph is connected and so is the type partitioning graph. Let $P_1, P_2, \ldots, P_d$ be the sets of the type partition. We say P_i is an I-set if P_i induces an independent set. Similarly, we say P_i is a C-set if P_i induces a clique. The size of a set P_i is the number of vertices in the set P_i.

Observe that a clique K_c with $c \geq 3$ and $K_{r,s}$ with $r \geq 2$ and $s \geq 3$ do not have a matching cut. It means that all the vertices of these graphs should be entirely on one side of the cut. Consider a partition P_i, vertices of P_i are labeled according to the following rules in order:

- If P_i is a C-set with size ≥ 2, vertices in the set P_i and all the vertices in its neighboring sets get the same label.
- If P_i is an I-set with size ≥ 3 and is adjacent to an I-set with size ≥ 2, then the vertices in both the sets get the same label.
- If P_i is an I-set with size ≥ 3 and is adjacent to two or more sets of size ≥ 1, then vertices in all these sets get the same label.
- If P_i is an I-set with size ≥ 3 and has only one adjacent set of size 1, then G has a matching cut.
- If P_i is an I-set with size 2 and is adjacent to an I-set of size 2 and a set of size 1, then vertices in all these sets get the same label.
- If P_i is an I-set with size 2 and is adjacent to only one I-set of size 2, in these two sets, each vertex will get different label.
- If P_i is an I-set with size 2 and is adjacent to two sets of size 1, in these three sets, each vertex will get different label.
- If P_i is an I-set with size 2 and is adjacent to a set of size 1, then G has a matching cut.
- All the remaining sets of size 1 will get different labels.

If we apply the above rules, either we conclude that G has a matching cut, or for each set we use at most 2 labels, hence we can state the following:

Lemma 2. *The number of labels required is at most* $2d$*.*

The vertices of each label should entirely be in the same set of the matching cut. Hence, there are 2^{2d} possible label combinations. Thus we have the following:

Theorem 3. *There is an algorithm with running time* $O^*(2^{2d})$ *that solves the matching cut problem, where d is the neighbourhood diversity of the graph.*

5 Matching Cut Problem for Other Structural Parameters

For graphs with bounded feedback vertex number, the tree-width is also bounded. As the matching cut problem is in FPT for tree-width, it is also in FPT for feedback vertex number. Kratsch and Le [11] showed that the matching cut problem is in FPT for the size of the vertex cover. We use the techniques used in [11] to show that the matching cut problem is in FPT for the parameters *twin cover* and the *distance to split graphs*.

Lemma 4 (stated as Lemma 3 in [11]). *Let I be an independent set and let $U = V \backslash I$. Given a partition (X, Y) of U, it can be decided in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Two non-adjacent (adjacent) vertices having the same open (closed) neighborhood are called *twins*. A *twin cover* is a vertex set S such that for each edge $\{u, v\} \in E$, either $u \in S$ or $v \in S$ or u and v are twins. Note that, for a twin cover $S \subseteq V$, $G[V \backslash S]$ is a collection of disjoint cliques.

Lemma 5. *Let $S \subseteq V$ be a twin cover of G. Given a partition (X, Y) of S, it can be decided in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Proof. Clearly, $V \backslash S$ induces a collection of disjoint cliques. Consider a maximal clique C on two or more vertices in $V \backslash S$. Let u, v be any two vertices of the clique C. Clearly, u and v are twins. If u and v has a common neighbor in both X and Y, then the graph has no matching cut such that $X \subseteq A$ and $Y \subseteq B$. Hence, without loss of generality we can assume that u and v have common neighbors only in X. Let $X' = X \cup V(C)$. Clearly, $V \backslash (S \cup V(C))$ is an independent set. Using Lemma 4, we can decide in $O(n^2)$ time if the graph has a matching cut (A, B) such that $X' \subseteq A$ and $Y \subseteq B$.

Let S be a twin cover of the graph. By guessing a partition (X, Y) of S, we can check in $O(n^2)$ time if G has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$. Hence we can state the following theorem.

Theorem 6. *There is an algorithm with running time $O^*(2^{|S|})$ to solve the matching cut problem, where S is the twin cover of the graph.*

Lemma 7. *Let G be a graph with vertex set V, if $S \subseteq V$ be such that $G[V \backslash S]$ is a split graph. Given a partition (X, Y) of S, it can be decided in $O(n^2)$ time whether the graph G has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$.*

Proof. Let $V \backslash S = C \cup I$ be the vertex set of the split graph, where C is a clique and I is an independent set. If $|C| = 1$ or $|C| \geq 3$, then let $X' = X \cup V(C)$ and $Y' = Y \cup V(C)$. Clearly, $V \backslash (S \cup V(C))$ is an independent set. Hence, G has matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$ if and only if G has a matching cut such that either $X' \subseteq A$ and $Y \subseteq B$ or $X \subseteq A$ and $Y' \subseteq B$. Both these instances can be solved in $O(n^2)$ time using Lemma 4. If $|C| = 2$, depending on whether the vertices of C go to X or Y, we solve four instances of Lemma 4 to check whether the graph has a matching cut (A, B) such that $X \subseteq A$ and $Y \subseteq B$. Therefore the time complexity is $O(n^2)$.

Similar to Theorem 6, we can state the following theorem.

Theorem 8. *There is an algorithm with running time $O^*(2^{|S|})$ to solve the matching cut problem, where $S \subseteq V$ such that $G[V \setminus S]$ is a split graph.*

6 Algorithms for H-Free 2-Coloring Problems

6.1 Overview of the Techniques Used

In the rest of the paper, we assume that the nice tree decomposition is given. Let i be a node in the nice tree decomposition, X_i is the bag of vertices associated with the node i. Let T_i be the subtree rooted at the node i and $G[T_i]$ denote the graph induced by all the vertices in T_i.

We use dynamic programming on the nice tree decomposition. We process the nodes of the nice tree decomposition according to its post order traversal. We say that a partition (A, B) of G is a *valid* partition if neither $G[A]$ nor $G[B]$ has H as an induced subgraph. At each node i, we check each bipartition (A_i, B_i) of the bag X_i to see if (A_i, B_i) leads to a valid partition in the graph $G[T_i]$. For each partition, we also keep some extra information that will help us to detect if the partition leads to an invalid partition at some ancestral (parent) node. We have four types of nodes in the tree decomposition – leaf, introduce, forget and join nodes. In the algorithm, we explain the procedure for updating the information at each of these nodes and consequently, to certify whether a partition is valid or not. During the description of the algorithms, we refer to the set $V(T_i) \setminus X_i$, i.e., the vertices in the subtree T_i but not in the bag X_i, as *forgotten vertices* of the subtree T_i.

In Section 6, we start the discussion with H-Free 2-Coloring problems. In Sections 6.2 and 6.3, we discuss the algorithm for the cases when $H = K_r$ and $H = C_4$ respectively before moving on to the case of general H in Section 6.4. In Section 7, we give the algorithm for H-Free q-Coloring problem. In Section 8, we give the algorithm for H-(Subgraph)Free q-Coloring problem. Presenting the algorithms for $H = K_r$ and $H = C_4$ initially will help in the exposition, as they will help to understand the setup before moving to the more involved general case.

6.2 K_r-Free 2-Coloring

In this section, we consider the H-Free 2-Coloring problem when $H = K_r$, a complete graph on r vertices.

Let $\Psi = (A_i, B_i)$ be a partition of a bag X_i. We set $M_i[\Psi]$ to 1 if there exists a partition (A, B) of $V(T_i)$ such that $A_i \subseteq A$, $B_i \subseteq B$ and both $G[A]$ and $G[B]$ are K_r-free. Otherwise, $M_i[\Psi]$ is set to 0.

Leaf node: For a leaf node $\Psi = (\emptyset, \emptyset)$ and $M_i[\Psi] = 1$. This step takes constant time.

Introduce node: Let j be the only child of the node i. Let v be the lone vertex in $X_i \setminus X_j$. Let $\Psi = (A_i, B_i)$ be a partition of X_i. If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, we use the following cases to compute $M_i[\Psi]$ value. Since v cannot have forgotten neighbors, it can form a K_r only within the bag X_i.

Case 1: $v \in A_i$, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \backslash \{v\}, B_i)$.

Case 2: $v \in B_i$, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i, B_i \backslash \{v\})$.

The total number of Ψ's for X_i is 2^{t+1}, for each Ψ checking if $G[A_i]$ or $G[B_i]$ contains K_r as subgraph can be done in $(t+1)^r r^2$ time. Hence the total time complexity at the introduce node is $O(2^t t^r)$.

Forget node: Let j be the only child of the node i. Let v be the lone vertex in $X_j \backslash X_i$. Let $\Psi = (A_i, B_i)$ be a partition of X_i. If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi] = \max\{M_j[\Psi'], M_j[\Psi'']\}$, where, $\Psi' = (A_i \cup \{v\}, B_i)$ and $\Psi'' = (A_i, B_i \cup \{v\})$.

The total number of Ψ's for X_i is 2^t, for each Ψ checking if $G[A_i]$ or $G[B_i]$ contains K_r as subgraph can be done in $t^r r^2$ time. Hence the total time complexity at the forget node is $O(2^t t^r)$.

Join node: Let j_1 and j_2 be the children of the node i. $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. Let $\Psi = (A_i, B_i)$ be a partition of X_i. If $G[A_i]$ or $G[B_i]$ has K_r as a subgraph, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute $M_i[\Psi]$ value. Since there are no edges between $V(T_{j_1}) \backslash X_i$ and $V(T_{j_2}) \backslash X_i$, a K_r cannot contain forgotten vertices from both T_{j_1} and T_{j_2}.

$$M_i[\Psi] = \begin{cases} 1, & \text{If } M_{j_1}[\Psi] = 1 \text{ and } M_{j_2}[\Psi] = 1. \\ 0, & \text{Otherwise.} \end{cases} \tag{1}$$

The total number of Ψ's for X_i is 2^{t+1}, for each Ψ checking if $G[A_i]$ or $G[B_i]$ contains K_r as subgraph can be done in $(t+1)^r r^2$ time. Hence the total time complexity at the join node is $O(2^t t^r)$.

The correctness of the algorithm is implied from the correctness of $M_i[\Psi]$ values, which can be proved using bottom up induction on the nice tree decomposition. G has a valid bipartitioning if there exists a Ψ such that $M_r[\Psi] = 1$, where r is the root node of the nice tree decomposition. The total time complexity of the algorithm is $O(2^t t^r \cdot n) = O(2^{t + r \log t} \cdot n)$. With this we state the following theorem.

Theorem 9. *There is an $O(2^{t+r \log t} \cdot n)$ time algorithm that solves the H-*Free *2-*Coloring *problem when $H = K_r$, on graphs with tree-width at most t.*

6.3 C_4-Free 2-Coloring

In this section, we describe the combinatorial algorithm for the H-Free 2-Coloring problem for the case when $H = C_4$, a cycle of length 4.

Note that an induced cycle of length 4 is formed when a pair of non-adjacent vertices have two non-adjacent neighbors. If a graph has no induced C_4 then any non-adjacent vertex pairs cannot have two or more non-adjacent vertices as neighbors. They can have neighbors which are pairwise adjacent. We keep track of such vertex pairs as they can form an induced C_4 at some ancestral (introduce/join) nodes. Let X_i be a bag at the node i of the nice tree decomposition. We consider partitions (A_i, B_i) of the bag X_i and see if they lead to a valid partition (A, B) of $V(T_i)$. For each non-adjacent pair of vertices from A_i (similarly B_i), we also guess if the pair has a common forgotten neighbor in part A (similarly B) of the partition. We check if the above guesses lead to a valid partitioning in the subgraph $G[T_i]$, which is the graph induced by the vertices in the node i and all its descendant nodes. In this section, we use the standard notation of $\binom{S}{2}$ to denote the set of all 2-subsets of a set S.

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple defined as follows: (A_i, B_i) is a partition of X_i, $P_i \subseteq \binom{A_i}{2}$ and $Q_i \subseteq \binom{B_i}{2}$. Intuitively, P_i and Q_i are the set of those non-adjacent pairs that have common forgotten neighbor.

We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. Every pair in P_i has a common neighbor in $A \backslash A_i$.
3. Every pair in $\binom{A_i}{2} \setminus P_i$ does not have a common neighbor in $A \backslash A_i$.
4. Every pair in Q_i has a common neighbor in $B \backslash B_i$.
5. Every pair in $\binom{B_i}{2} \setminus Q_i$ does not have a common neighbor in $B \backslash B_i$.
6. $G[A]$ and $G[B]$ are C_4-free.

Otherwise, $M_i[\Psi]$ is set to 0. Suppose there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition. Then the above conditions 1 and 6 ensure that G can be partitioned in the required manner.

When one of the following occurs, it is easy to see that the 4-tuple does not lead to a required partition. We say that the 4-tuple Ψ is *invalid* if one of the below cases occur:

(i) $G[A_i]$ or $G[B_i]$ contains an induced C_4.

(ii) There exists a pair $\{x, y\} \in P_i$ such that $\{x, y\} \in E$.

(iii) There exists a pair $\{x, y\} \in Q_i$ such that $\{x, y\} \in E$.

Note that it takes $O(t^4)$ time to check if a given Ψ is invalid. Below we explain how to compute $M_i[\Psi]$ value at each node i.

Leaf node: For a leaf node i, $\Psi = (\emptyset, \emptyset, \emptyset, \emptyset)$ and $M_i[\Psi] = 1$. This step takes constant time.

Introduce node: Let j be the only child of the node i. Suppose $v \in X_i$ is the new vertex present in X_i, $v \notin X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i, If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following cases to compute the $M_i[\Psi]$ value.

Case 1, $v \in A_i$: If $\exists \{v, x\} \in P_i$ for some $x \in A_i$ or if $\exists \{x, y\} \in P_i$ such that $\{x, y\} \subseteq N(v) \cap A_i$, then $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \backslash \{v\}, B_i, P_i, Q_i)$.

As v is a newly introduced vertex, it cannot have any forgotten neighbors. Hence, $\{v, x\} \in P_i \implies M_i[\Psi] = 0$. If x and y have a common forgotten neighbor, they all form an induced C_4, together with v. Hence $\{x, y\} \in P_i \implies M_i[\Psi] = 0$.

Case 2, $v \in B_i$: If $\exists \{v, x\} \in Q_i$ for some $x \in B_i$ or if $\exists \{x, y\} \in Q_i$ such that $\{x, y\} \subseteq N(v) \cap B_i$, then $M_i[\Psi] = 0$. Otherwise, $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i, B_i \backslash \{v\}, P_i, Q_i)$.

The total number of Ψ's for X_i is $2^{t+1}2^{(t+1)^2}$. It takes $O(t^4)$ time to check if Ψ is invalid. Hence total time complexity at the introduce node is $O(2^{t^2+3t}t^4)$.

Forget node: Let j be the only child of the node i. Suppose $v \in X_j$ is the vertex missing in X_i, $v \notin X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i, If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi]$ is computed as follows:

Case 1, $v \in A_j$: If $\exists x, y \in A_i$ such that $xy \notin E$ and $xv, yv \in E$, then v is a common forgotten neighbor for x and y. Hence we set $M_i[\Psi] = 0$ whenever $\{x, y\} \notin P_i$. Otherwise, let $R = \{\{x, y\} | x, y \in A_i \cap N(v)\}$. Some of the vertex pairs in R can still have a common forgotten neighbor (other than v) at node j which is adjacent to v. Also there can be new pairs formed with v at the node j. Let $S = \{\{v, x\} | x \in A_i\}$. We have the following equation.

$$\delta_1 = \max_{X \subseteq S, Y \subseteq R} \{M_j[A_i \cup \{v\}, B_i, (P_i \backslash R) \cup (X \cup Y), Q_i]\}. \tag{2}$$

Case 2, $v \in B_j$: This is analogous to Case 1. We set $M_i[\Psi] = 0$, whenever $\{x, y\} \notin Q_i$. Otherwise, let $R = \{\{x, y\} | x, y \in B_i \cap N(v)\}$ and $S = \{\{v, x\} | x \in B_i\}$.

$$\delta_2 = \max_{X \subseteq S, Y \subseteq R} \{M_j[A_i, B_i \cup \{v\}, P_i, (Q_i \backslash R) \cup (X \cup Y)]\}. \tag{3}$$

If $M_i[\Psi]$ is not set to 0 already, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$.

The total number of Ψ's for X_i is $2^t 2^{t^2}$. It takes $O(t^4)$ time to check if Ψ is invalid. The computations of δ_1 and δ_2 requires us to iterate over every subset of S which is of size at most t and every subset of R which is of size at most t^2. Hence, we get a factor of 2^{t+t^2} in the overall time complexity. Thus the total time complexity at the forget node is $O(2^{2t^2+2t}t^4)$.

Join node: Let j_1 and j_2 be the children of the node i. By the property of nice tree decomposition, we have $X_i = X_{j_1} = X_{j_2}$ and $V(T_{j_1}) \cap V(T_{j_2}) = X_i$. There are no edges between $V(T_{j_1}) \backslash X_i$ and $V(T_{j_2}) \backslash X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple of X_i. If Ψ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, we use the following expression to compute the value of $M_i[\Psi]$.

A pair $\{x, y\} \in P_i$ can come either from the left subtree or from the right subtree but not from both, for that would imply two distinct non-adjacent common neighbors for x and y and hence an induced C_4. For $X \subseteq P_i$ and $Y \subseteq Q_i$, $\Psi_1 = (A_i, B_i, X, Y)$ and $\Psi_2 = (A_i, B_i, P_i \backslash X, Q_i \backslash Y)$.

$$M_i[\Psi] = \begin{cases} 1, & \exists X \subseteq P_i, Y \subseteq Q_i \text{ such that } M_{j_1}[\Psi_1] = M_{j_2}[\Psi_2] = 1. \\ 0, & \text{Otherwise.} \end{cases} \tag{4}$$

The total number of Ψ's for X_i is $2^{t+1}2^{(t+1)^2}$. It takes $O(t^4)$ time to check if Ψ is invalid. As we solve the equation 4, a factor of $2^{(t+1)^2}$ comes in the overall time complexity. Hence total time complexity at the join node is $O(2^{2t^2+5t}t^4)$.

The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which follows by a bottom-up induction on the nice tree decomposition. G has a valid bipartitioning if there exists a 4-tuple Ψ such that $M_r[\Psi] = 1$, where r is the root of the nice tree decomposition. We have the following theorem.

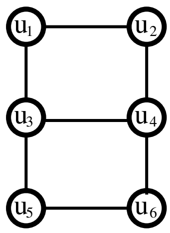

Figure 1: An example graph H.

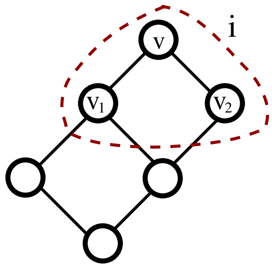

Figure 2: Forming H at an introduce node. Sequence $s = (v, v_2, v_1, \mathrm{fg}, \mathrm{fg}, \mathrm{fg})$.

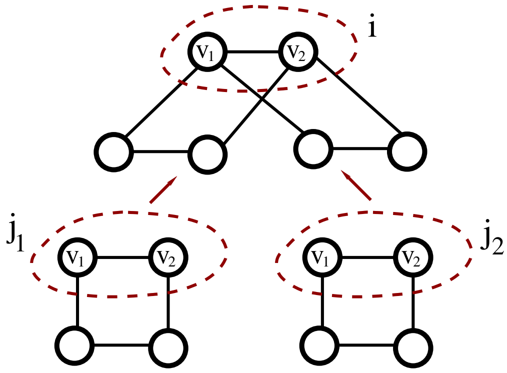

Figure 3: Forming H at join node. Sequences at node j_1, $s' = (\mathrm{dc}, \mathrm{dc}, v_1, v_2, \mathrm{fg}, \mathrm{fg})$, at node j_2, $s'' = (\mathrm{fg}, \mathrm{fg}, v_1, v_2, \mathrm{dc}, \mathrm{dc})$ gives a sequence $s = (\mathrm{fg}, \mathrm{fg}, v_1, v_2, \mathrm{fg}, \mathrm{fg})$ at node i. The vertices outside the dashed lines are forgotten vertices.

Theorem 10. *There is an $O(2^{3t^2} \cdot n)$ time algorithm that solves the H-*Free *2-*Coloring *problem when $H = C_4$ on graphs with tree-width at most t.*

6.4 H-Free 2-Coloring Problem

Let X_i be a bag at node i of the nice tree decomposition. Let (A_i, B_i) be a partition of X_i. We can easily check if $G[A_i]$ or $G[B_i]$ has H as an induced subgraph. Otherwise, we need to see if there is a partition (A, B) of $V(T_i)$ such that $A_i \subseteq A$, $B_i \subseteq B$ and both $G[A]$ and $G[B]$ are H-free. If there is such a partition (A, B), then $G[A]$ and $G[B]$ may have subgraph H', an induced subgraph of H which can lead to H at some ancestral node (introduce node or join node) of the nice tree decomposition (see Figures 2 and 3).

We perform dynamic programming over the nice tree decomposition. At each node i we guess a partition (A_i, B_i) of X_i and possible induced subgraphs of H that are part of A and B respectively. We check if such a partition is possible. Below we explain the algorithm in detail.

Let the vertices of the graph H be labeled as $u_1, u_2, u_3, \ldots, u_r$. Let (A_i, B_i) be a partition of vertices in the bag X_i. Let (A, B) be a partition of $V(T_i)$ such that $A \supseteq A_i$ and $B \supseteq B_i$. We define Γ_{A_i} as follows:

$$\begin{aligned} S_{A_i} =& \{(w_1, w_2, w_3, \dots, w_r) | w_\ell \in \{A_i \cup \{\text{fg}, \text{dc}\}\}, \\ & \forall \ell_1 \neq \ell_2, w_{\ell_1} = w_{\ell_2} \implies w_{\ell_1} \in \{\text{fg}, \text{dc}\}\}. \\ I_{A_i} =& \{s = (w_1, w_2, w_3, \dots, w_r) \in S_{A_i} | \text{ there exists } \ell_1 \neq \ell_2 \\ & \text{such that } w_{\ell_1} = \text{fg}, w_{\ell_2} = \text{dc and } \{u_{\ell_1}, u_{\ell_2}\} \in E(H)\}. \\ \Gamma_{A_i} =& S_{A_i} \backslash I_{A_i}. \end{aligned}$$

Here 'fg' represents a vertex in $A \backslash A_i$, i.e. the forgotten vertices in A. The label 'dc' (can be thought of as "don't care") represents the vertices that are not part of the subgraph right now, and can potentially be added at some ancestral nodes to form a larger induced subgraph of H.

Similarly, we can define Γ_{B_i} with respect to the sets B_i and B.

A sequence in S_{A_i} corresponds to an induced subgraph H' of H in A as follows:

1. If $w_\ell = \text{fg}$ then u_ℓ is part of $A \backslash A_i$, the forgotten vertices in A.
2. If $w_\ell = \text{dc}$ then u_ℓ is not be part of the subgraph H'.
3. If $w_\ell \in A_i$ then the vertex w_ℓ corresponds to the vertex u_ℓ of H'.

Γ_{A_i} is the set of sequences that can become H in future at some ancestral (introduce/join) node of the tree decomposition. Note that the sequences I_{A_i} are excluded from Γ_{A_i} because a forgotten vertex cannot have an edge to a vertex which will come in future at some ancestral node (introduce or join nodes).

Definition 2 (Induced Subgraph Legal Sequence in Γ_{A_i} with respect to A). *A sequence $s = (w_1, w_2, w_3, \dots, w_r) \in \Gamma_{A_i}$ is legal if the sequence s corresponds to an induced subgraph H' of H within A as follows.*

Let $FG(s) = \{\ell | w_\ell = fg\}$, $DC(s) = \{\ell | w_\ell = dc\}$ and $VI(s) = [r] \backslash \{FG(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ, $\ell \in \{VI(s) \cup FG(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FG(s)\}]$.

If there exist $|FG(s)|$ distinct vertices $z_\ell \in A \backslash A_i$ corresponding to each index in $FG(s)$ such that H' is isomorphic to $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FG(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

Analogously, we define legal/illegal sequences in Γ_{B_i} with respect to B.

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple. Here, (A_i, B_i) is a partition of X_i, $P_i \subseteq \Gamma_{A_i}$ and $Q_i \subseteq \Gamma_{B_i}$.

We define $M_i[\Psi]$ to be 1 if there is a partition (A, B) of $V(T_i)$ such that:

1. $A_i \subseteq A$ and $B_i \subseteq B$.
2. Every sequence in P_i is legal with respect to A.
3. Every sequence in Q_i is legal with respect to B.
4. Every sequence in $\Gamma_{A_i} \backslash P_i$ is illegal with respect to A.
5. Every sequence in $\Gamma_{B_i} \backslash Q_i$ is illegal with respect to B.
6. Neither $G[A]$ nor $G[B]$ contains H as an induced subgraph.

Otherwise $M_i[\Psi]$ is set to 0.

We call a 4-tuple Ψ as invalid if one of the following conditions occur. If Ψ is invalid we set $M_i[\Psi]$ to 0.

1. There exists a sequence $s \in P_i$ such that s does not contain dc.
2. There exists a sequence $s \in Q_i$ such that s does not contain dc.

As $|P_i| + |Q_i| \leq (t+5)^r$, it takes $(t+5)^r r$ time to check if Ψ is invalid.

Now we explain how to compute $M_i[\Psi]$ values at the leaf, introduce, forget and join nodes of the nice tree decomposition.

Leaf node: Let i be a leaf node, $X_i = \emptyset$, for $\Psi = (A_i, B_i, P_i, Q_i)$, we have $M_i[\Psi] = 1$. Here $A_i = B_i = \emptyset$, $P_i \subseteq \{([\mathrm{dc}]^r)\}$ and $Q_i \subseteq \{([\mathrm{dc}]^r)\}$. This step takes constant time.

Introduce node: Let i be an introduce node and j be the child node of i. Let $\{v\} = X_i \backslash X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i. If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise depending on whether $v \in A_i$ or $v \in B_i$ we have two cases. We discuss only the case $v \in A_i$, the case $v \in B_i$ can be analogously defined.

$v \in A_i$: We set $M_i[\Psi] = 0$, if there exists an illegal sequence s (in P_i) containing v or if there exists a trivial legal sequence s containing v but s is not in P_i.

That is, we set $M_i[\Psi] = 0$ if one of the following ($\star$) conditions occurs:

[$\star$ **Conditions**]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \notin E(H)$ but $\{v, w_{\ell_2}\} \in E(G)$.
3. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = \mathrm{fg}$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
4. Let $s = (w_1, w_2, w_3, \ldots, w_r) \in \Gamma_{A_i} \backslash P_i$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$, $w_{\ell_2} \in A_i \cup \{\mathrm{dc}\}$. For all $\ell_1 \neq \ell_2$, $w_{\ell_1}, w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \Longleftrightarrow \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

The conditions $1-3$ are to check if a sequence $s \in P_i$ containing the vertex v is an illegal sequence. The condition 4 is to check if a sequence $s \notin P_i$ containing the vertex v is a trivial legal sequence. Otherwise we set $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \backslash \{v\}, B_i, P_j, Q_i)$. Here P_j is computed as $P_j = \cup_{s \in P_i} \{\mathrm{Rep}_{\mathrm{dc}}(s, v)\}$, where $\mathrm{Rep}_{\mathrm{dc}}$ is defined as follows:

Definition 3. *$Rep_{dc}(s, v) = s'$, sequence s' obtained by replacing v (if present) with dc in s.*

Note that, $\mathrm{Rep}_{\mathrm{dc}}(s, v) = s$, if v not present in s.

The total number of Ψ's for X_i is $2^{(t+1)}2^{(t+5)^r}$. Checking if Ψ is invalid takes $(t+5)^r r$ time. Checking for illegal sequences containing v (steps 1 to 3 in $\star$ Conditions) takes $(t+5)^r r$ time. Checking for legal sequences containing v not part of P_i/Q_i (steps 4 in $\star$ Conditions) takes $(t+5)^r r^2$. Computing Ψ' takes $(t+5)^r r$. Hence total time complexity is $O(2^{(t+1)}2^{(t+5)^r}(t+5)^{2r}r^2) = O(2^{2t^r})$.

Forget node: Let i be a forget node and j be the only child of node i. Let $\{v\} = X_j \backslash X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i. If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we set $M_i[\Psi] = \max\{\delta_1, \delta_2\}$ where δ_1 and δ_2 are computed as follows:

Computing δ_1: Set $A_j = A_i \cup \{v\}$. As v is the extra vertex in A_j, there could be many possible P_j at node j.

Definition 4. *$Rep_{fg}(s, v) = s'$, sequence s' obtained by replacing v (if present) with fg in s.*

Note that, if s does not contain the vertex v then $\mathrm{Rep}_{\mathrm{fg}}(s, v) = s$.

We also extend the definition of $\mathrm{Rep}_{\mathrm{fg}}$ to a set of sequences as follows:

$$\mathrm{Rep}_{\mathrm{fg}}(S, v) = \cup_{s \in S}\{\mathrm{Rep}_{\mathrm{fg}}(s, v)\}.$$

Note that, if s is a legal sequence at the node j with respect to A, then $\mathrm{Rep}_{\mathrm{fg}}(s, v)$ is also a legal sequence at node i with respect to A.

$$\delta_1 = \max_{\substack{P_j \subseteq \Gamma_{A_j} \\ \mathrm{Rep}_{\mathrm{fg}}(P_j, v) = P_i}} \{M_j[(A_j, B_i, P_j, Q_i)]\}$$

Computing δ_2: $B_j = B_i \cup \{v\}$. It is analogous to computing δ_1 but we process on B.

The total number of Ψ's for X_i is $2^t(t+4)^r$. Checking for invalid case takes $(t+4)^r r$ time. computing δ_1 and δ_2 takes $2^{(t+4)^r}(t+4)^r r$ time. Hence the total time complexity is $O(2^t 2^{2(t+4)^r}(t+4)^{2r}r^2) = O(2^{3t^r})$.

Join node: Let i be a join node, j_1, j_2 be the left and right children of the node i respectively. $X_i = X_{j_1} = X_{j_2}$ and there are no edges between $V(T_{j_1})\backslash X_i$ and $V(T_{j_2})\backslash X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node i. If Ψ is invalid we set $M_i[\Psi] = 0$. Otherwise, we compute $M_i[\Psi]$ value as follows:

Definition 5. *Let $s = (w_1, w_2, w_3, \ldots, w_r)$, $s' = (w'_1, w'_2, w'_3, \ldots, w'_r)$ and $s'' = (w''_1, w''_2, w''_3, \ldots, w''_r)$ be three sequences. We say that $s = Merge(s', s'')$ if the following conditions are satisfied.*

1. *$\forall \ell \; w_\ell \in X_i \implies w'_\ell = w''_\ell = w_\ell$.*
2. *$\forall \ell \; w_\ell = fg \implies$ either ($w'_\ell = fg$ and $w''_\ell = dc$) or ($w'_\ell = dc$ and $w''_\ell = fg$).*
3. *$\forall \ell \; w_\ell = dc \implies w'_\ell = w''_\ell = dc$.*

Note that, if $s' \in \Gamma_{A_{j_1}}$ and $s'' \in \Gamma_{A_{j_2}}$ are legal sequences at node j_1 and j_2 respectively then s is a legal sequence at node i with respect to A. We extend the Merge operation to sets of sequences as follows:

$$\text{Merge}(S_1, S_2) = \{s | \exists s' \in S_1, s'' \in S_2 \text{ such that } s = \text{Merge}(s', s'')\}.$$

We set $M_i[\Psi] = 1$ if there exists P_{j_1}, Q_{j_1}, P_{j_2} and Q_{j_2} such that the following conditions are satisfied:

(i) $P_i = \text{Merge}(P_{j_1}, P_{j_2})$, (ii) $Q_i = \text{Merge}(Q_{j_1}, Q_{j_2})$,
(iii) $M_{j_1}[A_i, B_i, P_{j_1}, Q_{j_1}] = 1$, and (iv) $M_{j_2}[A_i, B_i, P_{j_2}, Q_{j_2}] = 1$.

The total number of Ψ's for X_i is $2^{(t+1)}2^{(t+5)^r}$. Checking if Ψ is invalid takes $(t+5)^r r$. A factor of $4^{(t+5)^r}(t+5)^r r$ comes as we try all possible $P_{j_1}, Q_{j_1}, P_{j_2}, Q_{j_2}$. Hence the total time complexity at join node is $O(2^{(t+1)}2^{3(t+5)^r}(t+5)^r r) = O(2^{4t^r})$.

The graph has a valid bipartitioning if there exists a Ψ such that $M_r[\Psi] = 1$, where r is the root node of the nice tree decomposition. The correctness of the algorithm is implied by the correctness of $M_i[\Psi]$ values, which can be proved using a bottom up induction on the nice tree decomposition. Thus we get the following:

Theorem 11. *There is an $O(2^{4t^r} \cdot n)$ time algorithm that solves the H-Free 2-Coloring problem for any arbitrary fixed H, on graphs with tree-width at most t.*

7 Algorithm for H-Free q-Coloring Problem

We note that our techniques extend in a straightforward manner to solve the H-Free q-Coloring problem. In this case, we have to consider tuples Ψ that have $2q$ sets. That is $\Psi = (A_i^1, A_i^2, \ldots, A_i^q, P_i^1, P_i^2, \ldots, P_i^q)$. Here $A_i^j \subseteq X_i$ and $P_i^j \subseteq \Gamma_{A_i^j}$. The operations at the leaf, introduce and forget nodes are very similar to the case of 2-coloring problem. At introduce and forget nodes we will have q cases instead of 2 cases. At the join node we need to define the Merge operation on q sets instead of 2 sets. Below is the modified definition of Merge.

Definition 6. *Let $s = (w_1, w_2, w_3, \ldots, w_r)$, $s^1 = (w_1^1, w_2^1, w_3^1, \ldots, w_r^1)$, $s^2 = (w_1^2, w_2^2, w_3^2, \ldots, w_r^2)$, ..., $s^q = (w_1^q, w_2^q, w_3^q, \ldots, w_r^q)$ be three sequences. We say that $s = \text{Merge}(s^1, s^2, s^3, \ldots, s^q)$ if the following conditions are satisfied.*

1. *$\forall \ell \; w_\ell \in X_i \implies w_\ell^1 = w_\ell^2 = \cdots = w_\ell^q = w_\ell$.*
2. *$\forall \ell \; w_\ell = fg \implies \exists i$ such that $w_\ell^i = fg$ and $\forall j \neq i$, $w_\ell^j = dc$.*
3. *$\forall \ell \; w_\ell = dc \implies w_\ell^1 = w_\ell^2 = \cdots = w_\ell^q = dc$.*

Thus we state the following theorem.

Theorem 12. *There is an $O(q^{4t^r} \cdot n)$ time algorithm that solves the H-Free q-Coloring problem for any arbitrary fixed H, on graphs with tree-width at most t.*

The H-Free Chromatic Number is at most the chromatic number $\chi(G)$. For graphs with tree-width t, we have $\chi(G) \leq t+1$. Our techniques can also be used to compute the H-Free Chromatic Number of the graph by searching for the smallest q for which there is an H-free q-coloring. We have the following theorem.

Theorem 13. *There is an $O(t^{4t^r} \cdot n \log t)$ time algorithm to compute H-Free Chromatic Number of the graph whose tree-width is at most t.*

8 Algorithm for H-(Subgraph)Free q-Coloring Problem

We can solve the H-(Subgraph)Free 2-Coloring problem using the techniques described in Section 6.4. As we are looking for bipartitioning without H as a subgraph, we need to modify the Definition 2 and $(\star)$ conditions.

Instead of Definition 2 we have Definition 7.

Definition 7 (Subgraph Legal Sequence in Γ_{A_i} with respect to A). *A sequence $s = (w_1, w_2, w_3, \ldots, w_r) \in \Gamma_{A_i}$ is legal if the sequence s corresponds to a subgraph H' of H within A as follows.*

Let $FG(s) = \{\ell | w_\ell = fg\}$, $DC(s) = \{\ell | w_\ell = dc\}$ and $VI(s) = [r] \backslash \{FG(s) \cup DC(s)\}$. Let H' be the induced subgraph of H formed by u_ℓ, $\ell \in \{VI(s) \cup FG(s)\}$. That is $H' = H[\{u_\ell | \ell \in VI(s) \cup FG(s)\}]$.

If there exist $|FG(s)|$ distinct vertices $z_\ell \in A \backslash A_i$ corresponding to each index in $FG(s)$ such that H' is a subgraph of $G[\{w_\ell | \ell \in VI(s)\} \cup \{z_\ell | \ell \in FG(s)\}]$, then s is legal. Otherwise, the sequence is illegal.

At the introduce node, instead of $(\star)$ conditions we have to check the following $(\star\star)$ conditions:

[$\star\star$ **Conditions**]

1. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$ but $\{v, w_{\ell_2}\} \notin E(G)$.
2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v$, $w_{\ell_2} = \text{fg}$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H)$.
3. Let $s = (w_1, w_2, w_3, \ldots, w_r) \in \Gamma_{A_i} \backslash P_i$. There exists ℓ_1 such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$, $w_{\ell_2} \in A_i \cup \{\text{dc}\}$. For all $\ell_1 \neq \ell_2$, $w_{\ell_1}, w_{\ell_2} \in A_i$, $\{u_{\ell_1}, u_{\ell_2}\} \in E(H) \Longrightarrow \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$.

Thus we get the following:

Theorem 14. *There is an $O(q^{4t^r} \cdot n)$ time algorithm that solves the H-(Subgraph)Free q-Coloring problem for any arbitrary fixed H, on graphs with tree-width at most t.*

Theorem 15. *There is an $O(t^{4t^r} \cdot n \log t)$ time algorithm to compute H-(Subgraph)Free Chromatic Number of the graph whose tree-width is at most t.*

References

[1] Aravind, N., Kalyanasundaram, S., Kare, A.: On structural parameterizations of the matching cut problem. In: Combinatorial Optimization and Applications. COCOA 2017. (2017) 475–482

[2] Aravind, N., Kalyanasundaram, S., Kare, A.: h-free coloring on graphs with bounded tree-width. In: Algorithms and Discrete Applied Mathematics. CALDAM 2019. (2019) 231–244

[3] Graham, R.L.: On primitive graphs and optimal vertex assignments. Annals of the New York Academy of Sciences **175**(1) (1970) 170–186

[4] Farley, A.M., Proskurowski, A.: Networks immune to isolated line failures. Networks **12**(4) (1982) 393–403

[5] Patrignani, M., Pizzonia, M.: The complexity of the matching-cut problem. In: 27th International Workshop Graph-Theoretic Concepts in Computer Science (WG 2001) Boltenhagen, Germany. (2001) 284–295

[6] Chvátal, V.: Recognizing decomposable graphs. Journal of Graph Theory **8**(1) (1984) 51–53

[7] Le, V.B., Randerath, B.: On stable cutsets in line graphs. In: 27th International Workshop Graph-Theoretic Concepts in Computer Science (WG 2001) Boltenhagen, Germany. (2001) 263–271

[8] Bonsma, P.: The complexity of the matching-cut problem for planar graphs and other graph classes. Journal of Graph Theory **62**(2) (2009) 109–126

[9] Moshi, A.M.: Matching cutsets in graphs. Journal of Graph Theory **13**(5) (1989) 527–536

[10] Borowiecki, M., Jesse-Józefczyk, K.: Matching cutsets in graphs of diameter 2. Theoretical Computer Science **407**(1-3) (2008) 574–582

[11] Kratsch, D., Le, V.B.: Algorithms solving the matching cut problem. Theoretical Computer Science **609**(2) (2016) 328–335

[12] Klein, S., de Figueiredo, C.M.H.: The NP-completeness of multi-partite cutset testing. Congressus Numerantium **119** (1996) 217–222

[13] Brandstädt, A., Dragan, F.F., Le, V.B., Szymczak, T.: On stable cutsets in graphs. Discrete Applied Mathematics **105**(1) (2000) 39–50

[14] Le, V.B., Mosca, R., Müller, H.: On stable cutsets in claw-free graphs and planar graphs. Journal of Discrete Algorithms **6**(2) (2008) 256–276

[15] Achlioptas, D.: The complexity of G-free colourability. Discrete Mathematics **165-166**(Supplement C) (1997) 21–30

[16] Kubicka, E., Kubicki, G., McKeon, K.A.: Chromatic sums for colorings avoiding monochromatic subgraphs. Electronic Notes in Discrete Mathematics **43** (2013) 247–254

[17] Kubicka, E., Kubicki, G., McKeon, K.A.: Chromatic sums for colorings avoiding monochromatic subgraphs. Discussiones Mathematicae Graph Theory **43** (08 2015) 541–555

[18] Karpiński, M.: Vertex 2-coloring without monochromatic cycles of fixed size is NP-complete. Theoretical Computer Science **659**(Supplement C) (2017) 88–94

[19] Xiao, M., Nagamochi, H.: Complexity and kernels for bipartition into degree-bounded induced graphs. Theoretical Computer Science **659** (2017) 72–82

[20] Cowen, L.J., Cowen, R.H., Woodall, D.R.: Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. Journal of Graph Theory **10**(2) (1986) 187–195

[21] Bazgan, C., Tuza, Z., Vanderpooten, D.: Degree-constrained decompositions of graphs: Bounded treewidth and planarity. Theoretical Computer Science **355**(3) (2006) 389–395

[22] Wu, Y., Yuan, J., Zhao, Y.: Partition a graph into two induced forests. Journal of Mathematical Study **1** (1996) 1–6

[23] Farrugia, A.: Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. The Electronic Journal of Combinatorics **11** (2004)

[24] Rao, M.: MSOL partitioning problems on graphs of bounded treewidth and clique-width. Theoretical Computer Science **377**(1) (2007) 260–267

[25] Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation **85**(1) (1990) 12–75

[26] Courcelle, B.: The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. Theoretical Informatics and Applications **26** (1992) 257–286

[27] Diestel, R.: Graph Theory. Springer-Verlag Heidelberg (2005)

[28] Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)

[29] Robertson, N., Seymour, P.: Graph minors. X. Obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B **52**(2) (1991) 153–190

[30] Kloks, T., ed. In: Treewidth: Computations and Approximations. Lecture Notes in Computer Science, Springer (1994)

[31] Lampis, M.: Algorithmic meta-theorems for restrictions of treewidth. Algorithmica **64**(1) (2012) 19–37

[32] Ganian, R.: Using neighborhood diversity to solve hard problems. CoRR **abs/1201.3091** (2012)