Bipartitioning Problems on Graphs with Bounded Tree-Width

N. R. Aravind, Subrahmanyam Kalyanasundaram, and Anjeneya Swami Kare

Department of Computer Science and Engineering,
IIT Hyderabad, Hyderabad, India
{aravind,subruk,cs14resch01002}@iith.ac.in

Abstract. For an undirected graph $G$, we consider the following problems: given a fixed graph $H$, can we partition the vertices of $G$ into two non-empty sets $A$ and $B$ such that neither the induced graph $G[A]$ nor $G[B]$ contain $H$ (i) as a subgraph? (ii) as an induced subgraph? These problems are NP-complete and are expressible in monadic second order logic (MSOL). The MSOL formulation, together with Courcelle’s theorem implies linear time solvability on graphs with bounded tree-width. This approach yields algorithms 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 these problems for graphs $G$ whose tree-width is bounded. We obtain $2^{O(t^r)} \cdot n$ time algorithms when $H$ is any fixed graph of order $r$. In the special case when $H = K_r$, a complete graph on $r$ vertices, we get an $2^{O(t + r \log t)} \cdot n$ time algorithm.

The techniques can be extended to provide FPT algorithms to determine the smallest number $q$ such that $V$ can be partitioned into $q$ parts such that none of the parts have $H$ as a subgraph (induced subgraph).

1 Introduction

Let $G = (V, E)$ be an undirected graph on $n$ vertices. In the classical $k$-coloring problem, we need to color the vertices of the graph using at most $k$ colors such that no pair of adjacent vertices are of the same color. The $k$-coloring problem is NP-complete for $k \geq 3$ and this problem, and its variants, have been studied extensively under various settings. For $k = 2$, this is equivalent to testing whether the graph is bipartite or not, which is of course solvable in polynomial time.

We consider the following generalization of the 2-coloring problem: we need to 2-color the vertices of the graph such that the subgraphs induced by the respective color classes do not have a fixed graph $H$ as a subgraph\footnote{The classical 2-coloring problem is obtained by setting $H = K_2$.}. We call this problem the Bipartitioning without Subgraph $H$ Problem or BWS-$H$ Problem in short.
BWS-\(H\) Problem

**Instance:** An undirected graph \(G = (V, E)\).

**Question:** Can \(V\) be partitioned into two non-empty sets \(A, B\) such that neither of the induced graphs \(G[A]\) and \(G[B]\) have \(H\) as a subgraph?

We also study the variant of the problem where \(H\) does not appear as an induced subgraph. We call this the \(H\)-Free Bipartitioning Problem.

**H-Free Bipartitioning Problem**

**Instance:** An undirected graph \(G = (V, E)\).

**Question:** Can \(V\) be partitioned into two non-empty sets \(A, B\) such that neither of the induced graphs \(G[A]\) and \(G[B]\) have \(H\) as an induced subgraph?

The BWS-\(H\) problem is NP-complete [1] unless \(H = K_2\). Recently, Karpiński [2] gave an alternate proof for the NP-completeness of the problem when \(H = C_r\), a cycle of fixed length \(r\). The \(H\)-Free Bipartitioning Problem is NP-complete [3] as long as \(H\) has 3 or more vertices. For fixed \(H\), both these problems can be expressed in monadic second order logic (MSOL). The well-known Courcelle’s theorem [4,5] states that any graph property that is expressible in MSOL is solvable in linear time for graphs with bounded tree-width. The resulting algorithms have a running time \(f(|\varphi|, t) \cdot n\), where \(|\varphi|\) is the length of the MSOL formula and \(t\) is the tree-width of the graph. Even though the algorithms run in linear time, the dependency of \(f\) on \(|\varphi|\) and \(t\) can be quite bad. Indeed in the worst case \(f(|\varphi|, t)\) can be a tower of exponentials. Considering this, it is preferable to have explicit combinatorial algorithms, since such algorithms are more efficient and are amenable to a precise running time analysis.

In this paper, we give combinatorial algorithms for both BWS-\(H\) and \(H\)-Free Bipartitioning problems. Our main result is the following:

**Theorem 1.** There are \(2^{O(t^r)} \cdot n\) time algorithms that solves the BWS-\(H\) and \(H\)-Free Bipartitioning problems for any arbitrary fixed \(H\) (\(|V(H)| = r\)), on graphs with tree-width at most \(t\).

We also obtain a much faster \(2^{O(t^r \log t)} \cdot n\) time algorithm when \(H = K_r\), a complete graph on \(r\) vertices. Note that in this case, the BWS-\(H\) problem and \(H\)-Free Bipartitioning problem coincide.

Graph bipartitioning with other constraints have been explored in the past. 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 [6] 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 [7,8]. Wu, Yuan and
Zhao [11] 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. A generalization of the \( H \)-Free Bipartitioning problem called \( H \)-Free \( q \)-Coloring has been mentioned in [10].

Farrugia [11] showed the NP-completeness of a general variant of the problem called \((P, Q)\)-coloring problem. Here, \( P \) and \( Q \) 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 \( P \) and \( Q \) respectively.

# Preliminaries

We write \( f(n) = O^*(g(n)) \) if \( f(n) = \Omega(g(n))n^c \) for some constant \( c > 0 \). Let \( G = (V, E) \) be an undirected graph. For \( u \in V \), the set of all neighbors of \( u \) (open neighborhood) is denoted by \( N(u) \). The closed neighborhood of \( u \), 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 \( G[S] \). When there is no ambiguity, we use the simpler notations \( S \setminus x \) to denote \( S \setminus \{x\} \) and \( S \cup x \) to denote \( S \cup \{x\} \). We denote the set of all \( k \) sized subsets of the set \( S \) by \( \binom{S}{k} \). We use \( uv \) to denote the edge \( \{u, v\} \) for convenience. We follow the standard graph theoretic terminology from [11].

A parameterized problem is a language \( L \subseteq \Sigma^* \times \mathbb{N} \), where \( \Sigma \) 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 [12].

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 [13]. 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 \( v \in V \setminus X_j \) such that \( X_j = X_i \cup \{v\} \).
3. Forget Node: A forget node \( i \) has exactly one child \( j \) and there is a vertex \( v \in V \setminus X_i \) 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 [14]. 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.

### 2.1 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 $i$, $G[T_i]$ denote the graph induced by all the vertices in $T_i$.

We use dynamic programming on the nice tree decomposition to solve the problems for different $H$. We process the nodes of 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 $G[B]$ have $H$ as a 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 above types of nodes and consequently, to certify whether a partition is valid or not.

In Section 3, we discuss algorithm for the case $H = K_r$, a complete graph on $r$ vertices. In Section 4, we discuss algorithm for the BWS-$H$ problem when $H = C_4$, a cycle of length 4. In Section 5, the algorithm for the BWS-$H$ problem for a fixed arbitrary graph $H$ is presented. Presenting 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 generalized case. Finally, we explain how the algorithm for the $H$-FREE BIPARTITIONING problem can be obtained by modifying the algorithm for the BWS-$H$ problem in Section 6.

### 3 Bipartitioning without $K_r$

We consider the BWS-$H$ 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 exist 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 $M_i[\Psi] = 1$.

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

**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 $\Psi = (A_i, B_i)$ 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 $\Psi'' = (A_i, B_i \cup v)$.

**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 $V(T_{j_1}) \setminus X_i$ and $V(T_{j_2}) \setminus X_i$, a $K_r$ cannot contain forgotten vertices from both $T_{j_1}$ and $T_{j_2}$.

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

Correctness of the algorithm implied from the correctness of $M_i[\Psi]$ values, which can be proved using bottom up induction on nice tree decomposition. $G$ has a valid bipartitioning if there exists a $\Psi$ 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 $2^t n = O^*(2^{t+n})$. With this we state the following theorem.

**Theorem 2.** There is an $O(2^{t+n} \log^4 n)$ time algorithm that solves the BWS-H problem when $H = K_r$, on graphs with tree-width at most $t$.

4 Bipartitioning without $C_4$

In this section, we describe the combinatorial algorithm for the BWS-H problem for the case when $H = C_4$, a cycle of length 4. As stated, the problem can be expressed in MSOL. An MSOL formulation of the BWS-H problem for the case $H = C_4$ is given below.

$$\exists V_1 \subseteq V : \exists V_2 \subseteq V : (V_1 \cap V_2 = \emptyset) \land (V_1 \cup V_2 = V) \land \neg (V_1 = \emptyset) \land \neg (V_2 = \emptyset) \land$$
$$\neg (u_1 u_2 \in E) \land (u_2 u_3 \in E) \land (u_3 u_4 \in E) \land (u_4 u_1 \in E) \land$$
$$\neg (u_1 u_2 \in E) \land (u_2 u_3 \in E) \land (u_3 u_4 \in E) \land (u_4 u_1 \in E).$$

The predicates $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ and $V_1 = \emptyset$ can be rewritten as follows:

$$V_1 \cap V_2 = \emptyset \iff \neg \exists v \in V : v \in V_1 \land v \in V_2,$$
$$V_1 \cup V_2 = V \iff \forall v \in V : v \in V_1 \lor v \in V_2,$$
$$V_1 = \emptyset \iff \forall v \in V : \neg (v \in V_1).$$
Note that a cycle of length 4 is formed when a pair of (adjacent or non-adjacent) vertices have two or more common neighbors. If a graph has no $C_4$ then any vertex pair can have at most one common neighbor. Let $X_i$ be a bag at the node $i$ of the nice tree decomposition. We guess a partition $(A_i, B_i)$ of the bag $X_i$. For each pair of vertices from $A_i$ (similarly $B_i$), we also guess if the pair has exactly one 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 descendent nodes. Below we formally explain the technique.

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 pairs that have exactly one 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 exactly one common neighbor in $A \setminus A_i$.
3. Every pair in $\binom{A_i}{2} \setminus P_i$ does not have a common neighbor in $A \setminus A_i$.
4. Every pair in $Q_i$ has exactly one common neighbor in $B \setminus B_i$.
5. Every pair in $\binom{B_i}{2} \setminus Q_i$ does not have a common neighbor in $B \setminus B_i$.
6. $G[A]$ and $G[B]$ do not have $C_4$ as a subgraph.

Otherwise, $M_i[\Psi]$ is set to 0. Suppose there exists a 4-tuple $\Psi$ such that $M_i[\Psi] = 1$, 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 $\Psi$ is invalid if one of the below cases occur:

(i) $G[A_i]$ or $G[B_i]$ contains a $C_4$.
(ii) There exists a pair $\{x, y\} \in P_i$ with a common neighbor in $A_i$.
(iii) There exists a pair $\{x, y\} \in Q_i$ with a common neighbor in $B_i$.

Note that it is easy to check if a given $\Psi$ 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 $M_i[\Psi] = 1$.

**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 $\Psi$ 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 $\Psi' = (A_i \setminus v, B_i, P_i, Q_i)$.

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 a $C_4$, 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 $\Psi' = (A_i, B_i \backslash v, P_i, Q_i)$.

**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 $\Psi$ is invalid, we set $M_i[\Psi]$ to 0. Otherwise, $M_i[\Psi]$ is computed as follows:

Case 1, $v \in A_i$: If $\exists x, y \in A_i$ such that $xv, yv \in E$, 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 $R = \{(x, y)|x, y \in A_i \cap N(v)\}$. At node $j$, note that any pair in $R$ with a common forgotten neighbor will form a $C_4$. Hence we consider only those $P_j$’s that are disjoint with $R$. Also there can be new pairs formed with $v$ at the node $j$. Let $S = \{(x, y)|x \in A_i\}$. We have the following equation.

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

Case 2, $v \in B_i$: 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 $S = \{(x, y)|x \in B_i\}$.

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

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

**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 $\Psi$ 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 common neighbors for $x$ and $y$ and hence a $C_4$. For $X \subseteq P_i$ and $Y \subseteq Q_i$, $\Psi_1 = (A_i, B_i, X, Y)$ and $\Psi_2 = (A_i, B_i, P_i \backslash X, Q_i \backslash Y)$.

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

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 $\Psi$ such that $M_r[\Psi] = 1$, where $r$ is the root of the nice tree decomposition.

The time complexity at each of the nodes in the tree decomposition is as follows: constant time at leaf nodes, $O^*(2^{t+\frac{t^2}{2}})$ time at insert nodes, $O^*(2^{t+\frac{t^2}{2}})$ time at forget nodes and $O^*(2^{t+2\frac{t^2}{2}})$ time at join nodes. This gives the following:

**Theorem 3.** There is an $O(2^{O(t^2)})$ time algorithm that solves the BWS-$H$ problem when $H = C_4$ on graphs with tree-width at most $t$. 
Fig. 1. An example graph $H$.

Fig. 2. Forming $H$ at an introduce node. Sequence $s = (v, v_2, v_1, fg, fg, fg)$.

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

5 Bipartitioning without $H$

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 a 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 $G[A_i]$ and $G[B_i]$ do not have $H$ as a subgraph. 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 $H$ are 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 $\Gamma_A$, as follows:
Similarly, we define legal/illegal sequences in $\Gamma$ as follows:

1. If $w_\ell = fg$ then $u_\ell$ is part of $A \setminus A_i$, the forgotten vertices in $A$.
2. If $w_\ell = dc$ then $u_\ell$ need not be part of the subgraph $H'$.
3. If $w_\ell \in A_i$ then the vertex $u_\ell$ corresponds to the vertex $u_\ell$ of $H'$.

$\Gamma_{A_i} = S_{A_i} \setminus I_{A_i}$

Here $fg$ represents a vertex in $A \setminus A_i$, the forgotten vertices in $A$ and $dc$ stands for don’t care. That is we don’t care if the corresponding vertex is part of the subgraph or not. Similarly, we can define $\Gamma_{B_i}$ with respect to the sets $B_i$ and $B$.

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

1. If $w_\ell = fg$ then $u_\ell$ is part of $A \setminus A_i$, the forgotten vertices in $A$.
2. If $w_\ell = dc$ then $u_\ell$ need not be part of the subgraph $H'$.
3. If $w_\ell \in A_i$ then the vertex $u_\ell$ corresponds to the vertex $u_\ell$ of $H'$.

**Definition 1 (Subgraph Legal Sequence in $\Gamma_{A_i}$ 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 subgraph $H'$ of $H$ within $A$ as follows.

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

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

Similarly, we define legal/illegal sequences in $\Gamma_{B_i}$ 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 $\Gamma_{A_i} \setminus P_i$ is illegal with respect to $A$.
5. Every sequence in $\Gamma_{B_i} \setminus Q_i$ is illegal with respect to $B$.
6. Neither $G[A]$ nor $G[B]$ contains $H$ as a subgraph.

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

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

Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple. Here, $(A_i, B_i)$ is a partition of $X_i$,
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 $dc$.

Now we explain how to compute $M_i[\Psi]$ values at the leaf, introduce, forgot 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 $A_i = B_i = \emptyset$, $P_i \subseteq \{[\text{dc}]\}$ and $Q_i \subseteq \{[\text{dc}]\}$.

**Introduce node:** Let $i$ be an introduce node and $j$ be the child node of $i$. Let $\{v\} = X_i \setminus X_j$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node $i$. If $\Psi$ 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 $M_i[\Psi] = 0$ in one of the following $(\ast)$ conditions occurs:

| Condition | Description |
|-----------|-------------|
| 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 $\{v, w\} \notin E(G)$. |
| 2. $\exists \ell_1 \neq \ell_2$, such that $w_{\ell_1} = v, w_{\ell_2} = fg, \{u_{\ell_1}, u_{\ell_2}\} \in E(H)$. |
| 3. Let $s = (w_1, w_2, w_3, \ldots, w_r) \in \Gamma_{A_i \setminus P_i}$. There exists $\ell_1$ such that $w_{\ell_1} = v$ and for all $\ell_2 \neq \ell_1$, $w_{\ell_2} \in A_i \cup \{dc\}$. For all $\ell_1 \neq \ell_2$, $w_{\ell_1}, w_{\ell_2} \in A_i, \{u_{\ell_1}, u_{\ell_2}\} \in E(H) \implies \{w_{\ell_1}, w_{\ell_2}\} \in E(G)$. |

Otherwise we set $M_i[\Psi] = M_j[\Psi']$, where $\Psi' = (A_i \setminus v, B_i, P_j, Q_i)$. Here $P_j$ is computed as follows:

**Definition 2.** $\text{Rep}_{dc}(s, v) = s'$, sequence $s'$ obtained by replacing $v$ (if present) with $dc$ in $s$.

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

$$P_j = \cup_{s \in P_i} \{\text{Rep}_{dc}(s, v)\}.$$ 

**Forget node:** Let $i$ be a forget node and $j$ be the only child of node $i$. Let $\{v\} = X_j \setminus X_i$. Let $\Psi = (A_i, B_i, P_i, Q_i)$ be a 4-tuple at node $i$. If $\Psi$ is invalid we set $M_i[\Psi] = 0$. Otherwise, we set $M_i[\Psi]$ as the extra vertex in $A_j$, there could be many possible $P_j$ at node $j$.

**Definition 3.** $\text{Rep}_{fg}(s, v) = s'$, sequence $s'$ obtained by replacing $v$ (if present) with $fg$ in $s$.

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

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

$$\text{Rep}_{fg}(S, v) = \cup_{s \in S} \{\text{Rep}_{fg}(s, v)\}.$$
Note that, if \( s \) is a legal sequence at the node \( j \) with respect to \( A \), then \( \text{Rep}_{f_3}(s, v) \) is also a legal sequence at node \( i \) with respect to \( A \).

\[
\delta_1 = \max_{P_j \subseteq F_{A_i} \atop \text{Rep}_{f_3}(P_j, v) = P_i} \{ M_j[(A_j, B_i, P_j, Q_i)] \}
\]

**Computing \( \delta_2 \):** \( B_j = B_i \cup v \). It is analogous to computing \( \delta_1 \) but we process on \( B \).

**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 \( V(T_{j_1}) \setminus X_i \) and \( V(T_{j_2}) \setminus X_i \). Let \( \Psi = (A_i, B_i, P_i, Q_i) \) be a 4-tuple at node \( i \). If \( \Psi \) is invalid we set \( M_i[\Psi] = 0 \). Otherwise, we compute \( M_i[\Psi] \) value as follows:

**Definition 4.** 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 \( s = \text{Merge}(s', s'') \) if the following conditions are satisfied.

1. \( \forall \ell \ w_{\ell} \in X_i \implies w'_{\ell} = w''_{\ell} = w_{\ell} \).
2. \( \forall \ell \ w_{\ell} = f g \implies \text{either } (w'_{\ell} = f g \text{ and } w''_{\ell} = dc) \text{ or } (w'_{\ell} = dc \text{ and } w''_{\ell} = f g) \).
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:

\[
\text{Merge}(S_1, S_2) = \{ s \mid \exists s' \in S_1, s'' \in S_2 \text{ such that } s = \text{Merge}(s', s'') \}.
\]

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) \( M_j[A_i, B_i, P_{j_1}, Q_{j_1}] = 1 \), and
(iv) \( M_j[A_i, B_i, P_{j_2}, Q_{j_2}] = 1 \).

The graph has valid bipartitioning if there exists a \( \Psi \) such that \( M_i[\Psi] = 1 \). 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. The time complexity of the algorithm is \( O^*(2^{2r}) \). Thus we get the following:

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

6 **H-Free Bipartitioning Problem**

The techniques described in Section 5 can also be used to solve the **H-Free Bipartitioning Problem**. As we are looking for bipartitioning without \( H \) as an induced subgraph. Definition \( B \) and (\( \ast \)) conditions at the introduced node are modified as below.
Definition 5 (Induced Subgraph Legal Sequence in \( \Gamma_{A_i} \) with respect to \( A \)). A sequence \( s = (w_1, w_2, w_3, \ldots, w_r) \) is legal if the sequence \( s \) corresponds to subgraph \( H' \) of \( H \) within \( A \) as follows.

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

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

\((*)\) conditions at the introduced node:

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

Thus we get the following:

**Theorem 5.** There is an \( 2^{O(t)} \cdot n \) time algorithm that solves the \( H \)-Free Bipartitioning Problem for any arbitrary fixed \( H \mid V(H) = r \), on graphs with tree-width at most \( t \).

**Coloring without subgraph \( H \):** We note that our techniques extend in a straightforward manner to solve the \( q \)-coloring analogues of BWS-\( H \) and \( H \)-Free Bipartitioning problems, where we have to partition the vertices of \( G \) into \( q \) sets such that graphs induced by none of these sets have \( H \) as a subgraph or induced subgraph. In this case, we have to consider tuples \( \Psi \) that have \( 2q \) sets. The operations at the leaf, introduce and forget nodes are very similar to the case of bipartitioning. At the join node we need to define the Merge operation on \( q \) sets instead of 2 sets. The running time of these algorithms are similar to that of the algorithms that solve the bipartitioning problems.

We further consider the optimization problems of finding the smallest \( q \) for which \( V(G) \) can be partitioned into \( q \) sets such that graphs induced by none of these sets have \( H \) as a subgraph or an induced subgraph. Since the chromatic number of \( G \) is at most \( t + 1 \) (where \( t \) is the tree-width of \( G \)), the algorithm needs to search for the smallest \( q \leq t + 1 \). Thus we get the following:

**Theorem 6.** The problem of finding the smallest \( q \) for which \( V(G) \) can be partitioned into \( q \) sets such that the graphs induced by none of these sets have \( H \) as a subgraph (or as an induced subgraph) is FPT when parameterized by tree-width.
1. Farrugia, A.: Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. The Electronic Journal of Combinatorics 11 (08 2004)
2. Karpiński, M.: Vertex 2-coloring without monochromatic cycles of fixed size is NP-complete. Theoretical Computer Science 659(Supplement C) (2017) 88–94
3. Achlioptas, D.: The complexity of G-free colourability. Discrete Mathematics 165-166(Supplement C) (1997) 21–30
4. Courcelle, B.: The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. Information and Computation 85(1) (1990) 12–75
5. Courcelle, B.: The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. Theoretical Informatics and Applications 26 (1992) 257–286
6. Xiao, M., Nagamochi, H.: Complexity and kernels for bipartition into degree-bounded induced graphs. Theoretical Computer Science 659 (2017) 72–82
7. 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
8. Bazgan, C., Tuza, Z., Vanderpoorten, D.: Degree-constrained decompositions of graphs: Bounded treewidth and planarity. Theoretical Computer Science 355(3) (2006) 389 – 395
9. Wu, Y., Yuan, J., Zhao, Y.: Partition a graph into two induced forests. Journal of Mathematical Study 1 (01 1996) 1–6
10. Rao, M.: MSOL partitioning problems on graphs of bounded treewidth and clique-width. Theoretical Computer Science 377(1) (2007) 260–267
11. Diestel, R.: Graph Theory. Springer-Verlag Heidelberg (2005)
12. Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)
13. Robertson, N., Seymour, P.: Graph minors. X. Obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52(2) (1991) 153–190
14. Kloks, T., ed. In: Treewidth: Computations and Approximations. Lecture Notes in Computer Science, Springer (1994)