Parameterized Algorithms for Recognizing Monopolar and 2-Subcolorable Graphs*

Iyad Kanj[1], Christian Komusiewicz[2], Manuel Sorge[3], and Erik Jan van Leeuwen[4]

1 **School of Computing, DePaul University, Chicago, USA**
ikanj@cs.depaul.edu
2 **Institut für Informatik, Friedrich-Schiller-Universität, Jena, Germany**
christian.komusiewicz@uni-jena.de
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Berlin, Germany**
manuel.sorge@tu-berlin.de
4 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
erikjan@mpi-inf.mpg.de

Abstract

We consider the recognition problem for two graph classes that generalize split and unipolar graphs, respectively. First, we consider the recognizability of graphs that admit a *monopolar partition*: a partition of the vertex set into sets A, B such that $G[A]$ is a disjoint union of cliques and $G[B]$ an independent set. If in such a partition $G[A]$ is a single clique, then G is a split graph. We show that in $\mathcal{O}(2^k \cdot k^3 \cdot (|V(G)| + |E(G)|))$ time we can decide whether G admits a monopolar partition (A, B) where $G[A]$ has at most k cliques. This generalizes the linear-time algorithm for recognizing split graphs corresponding to the case when $k = 1$.

Second, we consider the recognizability of graphs that admit a *2-subcoloring*: a partition of the vertex set into sets A, B such that each of $G[A]$ and $G[B]$ is a disjoint union of cliques. If in such a partition $G[A]$ is a single clique, then G is a unipolar graph. We show that in $\mathcal{O}(k^{2k+2} \cdot (|V(G)|^2 + |V(G)| \cdot |E(G)|))$ time we can decide whether G admits a 2-subcoloring (A, B) where $G[A]$ has at most k cliques. This generalizes the polynomial-time algorithm for recognizing unipolar graphs corresponding to the case when $k = 1$.

We also show that in $\mathcal{O}^*(4^k)$ time we can decide whether G admits a 2-subcoloring (A, B) where $G[A]$ and $G[B]$ have at most k cliques in total.

To obtain the first two results above, we formalize a technique, which we dub inductive recognition, that can be viewed as an adaptation of iterative compression to recognition problems. We believe that the formalization of this technique will prove useful in general for designing parameterized algorithms for recognition problems. Finally, we show that, unless the Exponential Time Hypothesis fails, no subexponential-time algorithms for the above recognition problems exist, and that, unless P=NP, no generic fixed-parameter algorithm exists for the recognizability of graphs whose vertex set can be bipartitioned such that one part is a disjoint union of k cliques.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity, G.2.2 Graph Theory

Keywords and phrases vertex-partition problems, monopolar graphs, subcolorings, split graphs, unipolar graphs, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.14

* Iyad Kanj, Christian Komusiewicz, and Manuel Sorge gratefully acknowledge the support by the DFG, projects DAPA, NI 369/12 and MAGZ, KO 3669/4-1.

1 Introduction

A (Π_A, Π_B)-*graph*, for graph properties Π_A, Π_B, is a graph $G = (V, E)$ for which V admits a partition into two sets A, B such that $G[A]$ satisfies Π_A and $G[B]$ satisfies Π_B. There is an abundance of (Π_A, Π_B)-graph classes, and important ones include, in addition to bipartite graphs (*i.e.*, 2-colorable graphs), well-known graph classes such as *split graphs* (which admit a bipartition into a clique and an independent set), and *unipolar graphs* (which admit a bipartition into a clique and a cluster graph). Here a *cluster graph* is a disjoint union of cliques.

The problem of recognizing whether a given graph belongs to a particular class of (Π_A, Π_B)-graphs is called (Π_A, Π_B)-Recognition, and is known as a *vertex-partition* problem. In general, most recognition problems for (Π_A, Π_B)-graphs are NP-hard [11], but bipartite, split, and unipolar graphs can all be recognized in polynomial time [17, 13, 16, 10, 20]. With the aim of generalizing these polynomial-time algorithms, we study the complexity of recognizing two classes of (Π_A, Π_B)-graphs that generalize split and unipolar graphs.

First, we consider *monopolar graphs*; these are graphs in which the vertex set admits a bipartition into a cluster graph and an independent set, and thus generalize split graphs. Monopolar graphs have applications in the analysis of protein-interaction networks [3]. The recognition problem of monopolar graphs can be formulated as follows:

Monopolar Recognition
Input: A graph $G = (V, E)$.
Question: Does G have a *monopolar partition* (A, B), that is, can V be partitioned into sets A and B such that $G[A]$ is a cluster graph and $G[B]$ is an independent set?

Second, we study *2-subcolorable graphs*; these are graphs in which the vertex set admits a bipartition into two cluster graphs [2], and thus generalize unipolar graphs. The recognition problem of 2-subcolorable graphs can be formulated as follows:

2-Subcoloring
Input: A graph $G = (V, E)$.
Question: Does G have a *2-subcoloring* (A, B), that is, can V be partitioned into sets A and B such that each of $G[A]$ and $G[B]$ is a cluster graph?

Monopolar Recognition and 2-Subcoloring are both NP-hard [1, 11]. This is a stark contrast to the variants of both problems where $G[A]$ is required to consist of a single cluster, which correspond to the recognition of split graphs and unipolar graphs, respectively, and admit polynomial-time algorithms [13, 16, 10, 20]. This has left the complexity of Monopolar Recognition and 2-Subcoloring parameterized by the number of clusters in $G[A]$ as intriguing open questions.

Our Results. We show that both Monopolar Recognition and 2-Subcoloring are fixed-parameter tractable parameterized by the number of clusters in $G[A]$. More formally, let $G = (V, E)$ be a graph and k a nonnegative integer. We prove the following:

▶ **Theorem 1.1.** *In $\mathcal{O}(2^k \cdot k^3 \cdot (|V| + |E|))$ time, we can decide whether G admits a monopolar partition (A, B) such that $G[A]$ is a cluster graph with at most k clusters.*

Observe that the algorithm runs in linear time for any fixed k. In particular, the algorithm recognizes split graphs (the case $k = 1$) in linear time, matching the running time of the existing algorithm for this problem [13].

▶ **Theorem 1.2.** *In $\mathcal{O}(k^{2k+2} \cdot (|V|^2 + |V| \cdot |E|))$ time, we can decide whether G admits a 2-subcoloring (A, B) such that $G[A]$ is a cluster graph with at most k clusters.*

For both problems, one faces various technical difficulties when designing recognition algorithms due to the following: First, we parameterize by the number of clusters in $G[A]$ and not by the number of vertices. Second, the number of clusters or vertices of $G[B]$ can be arbitrary. In particular, 2-Subcoloring does not seem to yield to standard approaches in parameterized algorithms. To overcome these difficulties, we propose a technique, which we dub *inductive recognition*, that is an adaptation of iterative compression to recognition problems. We believe that the applications of this technique extend beyond the problems under consideration in this paper, and will have general use for designing parameterized algorithms for recognition problems.

Inductive recognition on a graph $G = (V, E)$ works as follows. Start with an empty graph G_0 that trivially belongs to the graph class. In iteration i, we recognize whether the subgraph G_i of G induced by the first i vertices of V still belongs to the graph class, given that G_{i-1} belongs to the graph class. This technique is very similar to the well-known iterative compression technique [18]. The crucial difference, however, is that in iterative compression we can always add the i-th vertex v_i to the solution from the previous iteration to obtain a new solution (which we compress if it is too large). However, in our problems, we cannot simply add v_i to one part and witness that G_i is still a member of the graph class with possibly too many clusters. For example, if we consider v_i with respect to a monopolar partition (A, B) of G_{i-1}, then potentially v_i could neighbor a vertex in B and vertices of two clusters in $G[A]$. Therefore, we cannot add v_i to A or B to obtain another monopolar partition, even if G_i is monopolar, and hence, we also cannot perform a "compression step". Instead, we must "repair" the solution by rearranging vertices. This idea is formalized in the inductive recognition framework in Section 3.

In the case of 2-Subcoloring, we also consider the weaker parameter of the total number of clusters in $G[A]$ and $G[B]$. This parameterization makes the problem amenable to a branching strategy that branches on the placement of the endpoints of suitably-chosen edges and nonedges of the graph. This way, we create partial 2-subcolorings (A', B') where each vertex in $V \setminus (A' \cup B')$ is adjacent to the vertices of exactly two partial clusters, one in each of $G[A']$ and $G[B']$. Then we show that whether such a partial 2-subcoloring extends to an actual 2-subcoloring of G can be tested in polynomial time via a reduction to 2-CNF-Sat.

▶ **Theorem 1.3** ($\star$[1])**.** *In $\mathcal{O}^*(4^k)$ time, we can decide whether G admits a 2-subcoloring (A, B) such that $G[A]$ and $G[B]$ are cluster graphs with at most k clusters in total.*

Finally, we consider the parameter consisting of the total number of vertices in $G[A]$. We observe that a straightforward branching strategy yields a generic fixed-parameter algorithm for many (Π_A, Π_B)-Recognition problems.

▶ **Proposition 1.4** ($\star$)**.** *Let Π_A and Π_B be two graph properties such that membership of Π_A can be decided in polynomial time and Π_B can be characterized by a finite set of forbidden induced subgraphs. Then we can decide in $\mathcal{O}^*(2^{\mathcal{O}(k)})$ time whether V can be partitioned into sets A and B such that $G[A] \in \Pi_A$, $G[B] \in \Pi_B$, and $|A| \leq k$.*

We observe that several possible improvements or generalizations of our results are unlikely. First, we notice that subexponential-time fixed-parameter algorithms for both Monopolar Recognition and 2-Subcoloring are unlikely.

[1] The proofs of the results marked with a "$\star$" are omitted due to the lack of space.

▶ **Proposition 1.5** (⋆). MONOPOLAR RECOGNITION *parameterized by the number k of clusters in $G[A]$ and* 2-SUBCOLORING *parameterized by the total number k of clusters in $G[A]$ and $G[B]$ cannot be solved in $\mathcal{O}^*(2^{o(k)})$ time, unless the Exponential Time Hypothesis fails.*

Second, observe that Theorems 1.1 and 1.2 give fixed-parameter algorithms for two (Π_A, Π_B)-RECOGNITION problems, in both of which Π_A defines the set of all cluster graphs, parameterized by the number of clusters in $G[A]$. Hence, one might hope for a generic fixed-parameter algorithm for such problems, irrespective of Π_B. However, polar graphs stand in our way. A graph $G = (V, E)$ has a *polar partition* if V can be partitioned into sets A and B such that $G[A]$ is a cluster graph and $G[B]$ is the complement of a cluster graph [21].

▶ **Proposition 1.6** (⋆). *It is NP-hard to decide whether G has a polar partition (A, B) such that $G[A]$ is a cluster graph with one cluster or $G[B]$ is a co-cluster graph with one co-cluster.*

Related Work. To the best of the authors' knowledge, the parameterized complexity of MONOPOLAR RECOGNITION and 2-SUBCOLORING has not been studied before. The known algorithms for both problems are not parameterized, and assume that the input graph belongs to a structured graph class; see [4, 5, 9, 15] and [2, 12, 19], respectively. Recently, Kolay and Panolan [14] considered the problem of deleting k vertices or edges to obtain an (r, ℓ)-graph. For integers r, ℓ, a graph $G = (V, E)$ is an *(r, ℓ)-graph* if V can be partitioned into r independent sets and ℓ cliques. For example, $(2, 0)$-graphs are precisely bipartite graphs and $(1, 1)$-graphs are precisely split graphs. However, observe that $(1, \cdot)$-graphs are *not* monopolar graphs, because monopolar graphs do not allow edges between the cliques (as $G[A]$ is a cluster graph), whereas such edges are allowed in $(1, \cdot)$-graphs. These differences lead to substantially different algorithmic techniques. For example, since Kolay and Panolan consider the deletion problem, they can use iterative compression in their work. Moreover, they consider $r, \ell < 3$, which makes even $n^{f(r,\ell)}$-time algorithms polynomial. Neither of these ideas works for the problems in this paper. We were, however, inspired by their Observation 2, which we adapt to our setting to help bound the running time of our algorithms.

2 Preliminaries

For the relevant notions from parameterized algorithms, we refer to the literature [8, 6]. We follow standard graph-theoretic notation [7]. Let G be a graph. By $V(G)$ and $E(G)$ we denote the vertex-set and the edge-set of G, respectively. For $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by X. For a vertex $v \in G$, $N(v)$ and $N[v]$ denote the open neighborhood and the closed neighborhood of v, respectively. For $X \subseteq V(G)$, we define $N(X) := (\bigcup_{v \in X} N(v)) \setminus X$ and $N[X] := \bigcup_{v \in X} N[v]$. We say that a vertex v is *adjacent to a subset $X \subseteq V(G)$ of vertices* if v is adjacent to at least one vertex in X. A P_3 is a path on 3 vertices. We repeatedly use the following well-known characterization of cluster graphs:

▶ **Fact 2.1.** *A graph is a cluster graph if and only if it contains no P_3 as an induced subgraph.*

The asymptotic notation $\mathcal{O}^*()$ suppresses a polynomial factor in the input length. For $\ell \in \mathbb{N}$, by $[\ell]$ we denote the set $\{1, \ldots, \ell\}$.

3 Foundations for Inductive Recognition

We describe the foundations of the general technique that we use to recognize monopolar and 2-subcolorable graphs. The technique works in a similar way to the iterative compression

technique by Reed *et al.* [18]. Let $\mathcal{G}$ be an arbitrary hereditary graph class (*i.e.* if $G \in \mathcal{G}$, then $G' \in \mathcal{G}$ for every induced subgraph G' of G). We call an algorithm $\mathcal{A}$ an *inductive recognizer* for $\mathcal{G}$ if given a graph $G = (V, E)$, a vertex $v \in V$ such that $G - v \in \mathcal{G}$, and a certificate for $G - v \in \mathcal{G}$, algorithm $\mathcal{A}$ correctly decides whether $G \in \mathcal{G}$ and gives a membership certificate if $G \in \mathcal{G}$.

▶ **Theorem 3.1** (⋆). *Given an inductive recognizer $\mathcal{A}$ for $\mathcal{G}$, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}$ in time $\mathcal{O}(|V| + |E|) + \sum_{i=1}^{|V|} T(i)$, where $T(i) > 0$ is the worst-case running time of $\mathcal{A}$ on a graph of size at most i.*

For the purpose of this paper, we consider *parameterized inductive recognizers*. In addition to G and v, these recognizers take a nonnegative integer k as input. The above general theorem can then be instantiated as follows.

▶ **Corollary 3.2** (⋆). *Let k be a nonnegative integer, and let Π_A and Π_B be two graph properties. Let $\mathcal{G}_k$ be a hereditary class of (Π_A, Π_B)-graphs with an arbitrary additional property that may depend on k.*

- *Given a parameterized inductive recognizer $\mathcal{A}$ for $\mathcal{G}_k$, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}_k$ in time $\mathcal{O}(|V| + |E|) + \sum_{i=1}^{|V|} T(i, k)$, where $T(i, k)$ is the worst-case running time of $\mathcal{A}$ with parameter k on a graph of size at most i.*
- *Given a parameterized inductive recognizer $\mathcal{A}$ for $\mathcal{G}_k$ that runs in time $f(k) \cdot \Delta$, where Δ is the maximum degree of the input graph and f is an arbitrary computable function, we can recognize whether a given graph $G = (V, E)$ is a member of $\mathcal{G}_k$ in time $f(k) \cdot (|V| + |E|)$.*

4 An FPT algorithm for Monopolar Recognition

In this section, we give an FPT algorithm for Monopolar Recognition parameterized by the number of clusters. Throughout, given a graph $G = (V, E)$ and a nonnegative integer k, we say that a monopolar partition (A, B) of G is *valid* if $G[A]$ is a cluster graph with at most k clusters. Using Corollary 3.2, it suffices to give a parameterized inductive recognizer for graphs with a valid monopolar partition. That is, we need to solve the following problem in time $f(k) \cdot \Delta$, where f is some computable function and Δ the maximum degree of G:

Inductive Monopolar Recognition
Input: A graph $G = (V, E)$, a vertex $v \in V$, and a valid monopolar partition (A', B') of $G' = G - v$.
Question: Does G have a valid monopolar partition (A, B)?

Fix an instance of Inductive Monopolar Recognition with a graph $G = (V, E)$, a vertex $v \in V$, and a valid monopolar partition (A', B') of $G' = G - v$.

To find a valid monopolar partition (A, B) of G, we try the two possibilities of placing v in A or placing v in B. More precisely, in one case, we start a search from the bipartition $(A' \cup \{v\}, B')$, and in the other case, we start a search from the bipartition $(A', B' \cup \{v\})$. Neither of these two partitions is necessarily a valid monopolar partition of G. The search strategy is to try to "repair" a candidate partition by moving few vertices from one part of the partition to the other part. During this process, if a vertex is moved from one part to the other, then it will never be moved back. To formalize this approach, we introduce the notion of constraints.

▶ **Definition 4.1.** A *constraint* $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ is a four-partition of V such that $A^{\mathcal{C}}_* \subseteq A'$ and $B^{\mathcal{C}}_* \subseteq B'$. The vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ are called *permanent* vertices of the constraint. A constraint $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ is *fulfilled* by a vertex bipartition (A, B) of G if (A, B) is a valid monopolar partition of G such that:

1. $A_P^{\mathcal{C}} \subseteq A$; and
2. $B_P^{\mathcal{C}} \subseteq B$.

The permanent vertices in $A_P^{\mathcal{C}}$ and $B_P^{\mathcal{C}}$ in the above definition will correspond to those vertices that were moved during the search from one part to the other part. Note that:

▶ **Fact 4.2.** *Each valid monopolar partition (A, B) of G fulfills either $(A', \{v\}, B', \emptyset)$ or $(A', \emptyset, B', \{v\})$.*

We call the two constraints in Fact 4.2 the *initial constraints* of the search. We solve Inductive Monopolar Recognition by giving a search-tree algorithm that determines for each of the two initial constraints whether there is a partition fulfilling it. The root of the search tree is a dummy node that has two children, associated with the two initial constraints. Each non-root node in the search tree is associated with a constraint $\mathcal{C}$, and the algorithm searches for a solution that fulfills $\mathcal{C}$. To this end, the algorithm applies reduction and branching rules that find vertices that in *every* valid monopolar partition (A, B) fulfilling $\mathcal{C}$ are in $A_*^{\mathcal{C}} \cap B$ or $B_*^{\mathcal{C}} \cap A$; that is, these vertices must "switch sides".

Formally, a *reduction rule* that is applied to a constraint $\mathcal{C}$ associated with a node α in the search tree associates α with a new constraint $\mathcal{C}'$ or rejects $\mathcal{C}$; the reduction rule is *correct* either if $\mathcal{C}$ has a fulfilling partition if and only if $\mathcal{C}'$ does, or if the rule rejects $\mathcal{C}$, then no valid monopolar partition of G fulfills $\mathcal{C}$. A *branching rule* applied to a constraint $\mathcal{C}$ associated with a node α in the search tree produces more than one child node of α, each associated with a constraint; the branching rule is *correct* if $\mathcal{C}$ has a fulfilling partition if and only if at least one of the child nodes of α is associated with a constraint $\mathcal{C}'$ that has a fulfilling partition.

The algorithm first performs the reduction rules exhaustively, in order, and then performs the branching rules, in order. That is, Reduction Rule i may only be applied if Reduction Rule i' for all $i' < i$ cannot be applied. In particular, after Reduction Rule i is applied, we start over and apply Reduction Rule 1, etc. The same principle applies to the branching rules; moreover, branching rules are only applied if no reduction rule can be applied.

Let $\mathcal{C} = (A_*^{\mathcal{C}}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}})$ be a constraint. We now describe the reduction rules. Bear in mind that cluster graphs contain no P_3 as an induced subgraph (Fact 2.1). The first reduction rule identifies obvious cases in which a constraint cannot be fulfilled.

▶ **Reduction Rule 4.3** ($\star$). *If $G[A_P^{\mathcal{C}}]$ is not a cluster graph with at most k clusters, or if $G[B_P^{\mathcal{C}}]$ is not an independent set, then reject the current constraint.*

The second reduction rule finds vertices that must be moved from $B_*^{\mathcal{C}}$ to $A_P^{\mathcal{C}}$.

▶ **Reduction Rule 4.4** ($\star$). *If there is a vertex $u \in B_*^{\mathcal{C}}$ that has a neighbor in $B_P^{\mathcal{C}}$, then set $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$ and $B_*^{\mathcal{C}} \leftarrow B_*^{\mathcal{C}} \setminus \{u\}$; that is, replace $\mathcal{C}$ with the constraint $(A_*^{\mathcal{C}}, A_P^{\mathcal{C}} \cup \{u\}, B_*^{\mathcal{C}} \setminus \{u\}, B_P^{\mathcal{C}})$.*

The third reduction rule finds vertices that must be moved from $A_*^{\mathcal{C}}$ to $B_P^{\mathcal{C}}$.

▶ **Reduction Rule 4.5** ($\star$). *If there is a vertex $u \in A_*^{\mathcal{C}}$ and two vertices $w, x \in A_P^{\mathcal{C}}$ such that $G[\{u, w, x\}]$ is a P_3, set $A_*^{\mathcal{C}} \leftarrow A_*^{\mathcal{C}} \setminus \{u\}$ and $B_P^{\mathcal{C}} \leftarrow B_P^{\mathcal{C}} \cup \{u\}$.*

The first branching rule identifies pairs of vertices from $A_*^{\mathcal{C}}$ such that at least one of them must be moved to $B_P^{\mathcal{C}}$ because they form a P_3 with a vertex in $A_P^{\mathcal{C}}$.

▶ **Branching Rule 4.6** ($\star$). *If there are two vertices $u, w \in A_*^{\mathcal{C}}$ and a vertex $x \in A_P^{\mathcal{C}}$ such that $G[\{u, w, x\}]$ is a P_3, then branch into two branches, one associated with the constraint $(A_*^{\mathcal{C}} \setminus \{u\}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{u\})$ and one with constraint $(A_*^{\mathcal{C}} \setminus \{w\}, A_P^{\mathcal{C}}, B_*^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{w\})$.*

It is important to observe that if none of the previous rules applies, then $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition (we prove this rigorously in Lemma 4.8). However, $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$ may consist of too many clusters for this to be a *valid* monopolar partition. To check whether it is possible to reduce the number of clusters in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$, we apply a second branching rule that deals with singleton clusters in $G[A']$.

▶ **Branching Rule 4.7 ($\star$).** *If there is a vertex $u \in A^{\mathcal{C}}_*$ such that $\{u\}$ is a cluster in $G[A']$, then branch into two branches: the first is associated with the constraint $(A^{\mathcal{C}}_* \setminus \{u\}, A^{\mathcal{C}}_P \cup \{u\}, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$, and the second is associated with the constraint $(A^{\mathcal{C}}_* \setminus \{u\}, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P \cup \{u\})$.*

If no more rules apply to a constraint $\mathcal{C}$, then we can determine whether $\mathcal{C}$ can be fulfilled:

▶ **Lemma 4.8.** *Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint to which Reduction Rules 4.3, 4.4, and 4.5, and Branching Rules 4.6 and 4.7 do not apply. Then $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition. Moreover, there is a valid monopolar partition (A, B) fulfilling $\mathcal{C}$ if and only if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is valid.*

Proof. First, we show that $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition. There are no induced P_3's in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$, because Reduction Rules 4.3 and 4.5 and Branching Rule 4.6 do not apply, and because there are no induced P_3's in G containing three vertices from $A^{\mathcal{C}}_* \subseteq A'$. Similarly, there are no edges in $G[B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P]$, because Reduction Rules 4.3 and 4.4 do not apply, and because there are no edges in $G[B']$ and $B^{\mathcal{C}}_* \subseteq B'$.

To show the second statement in the lemma, observe that, if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is valid, then $\mathcal{C}$ is fulfilled by $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$. It remains to show that, if $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is not valid, then each monopolar partition (A, B) of G fulfilling $\mathcal{C}$ is not valid. For the sake of contradiction, assume that this is not the case and let (A, B) be a valid monopolar partition fulfilling $\mathcal{C}$. Since $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$ is a monopolar partition of G that is not valid, there are more than k clusters in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$. Thus, there is a cluster Q in $G[A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P]$ such that $Q \subseteq B$. Note that $|Q| = 1$, because $G[B]$ is an independent set and $Q \subseteq B$. Because (A, B) fulfills $\mathcal{C}$, $Q \cap A^{\mathcal{C}}_P = \emptyset$ and thus $Q \subseteq A^{\mathcal{C}}_*$. Hence, Q is a subset of a cluster Q' of $G[A']$, as $Q \subseteq A^{\mathcal{C}}_* \subseteq A'$. However, $|Q'| \geq 2$, because Branching Rule 4.7 does not apply even though $Q \subseteq A^{\mathcal{C}}_*$. Hence, any rule that moved the vertices of $Q' \setminus Q$ was not Branching Rule 4.7. Then the description of the other rules implies that $Q' \setminus Q \subseteq B^{\mathcal{C}}_P$. Note that $B^{\mathcal{C}}_P \subseteq B$, because (A, B) fulfills $\mathcal{C}$. Hence, $Q' \subseteq B$ and thus $G[B]$ is not an independent set. Therefore, (A, B) is not a monopolar partition, a contradiction to our choice of (A, B). ◀

The following lemmas will be used to upper bound the depth of the search tree, and the number of applications of each rule along each root-leaf path in this tree. Herein a *leaf* of the search tree is a node associated either with a constraint that Reduction Rule 4.3 rejects, or with a constraint to which no rule applies.

▶ **Lemma 4.9.** *Along any root-leaf path in the search tree of the algorithm, Reduction Rule 4.4 is applied at most $k + 1$ times.*

Proof. Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint obtained from an initial constraint via $k+1$ applications of Reduction Rule 4.4 and an arbitrary number of applications of Reduction Rules 4.3 and 4.5, and Branching Rules 4.6 and 4.7. Each application of Reduction Rule 4.4 adds a vertex of B' to $A^{\mathcal{C}}_P$. Since $G[B']$ is an independent set, any monopolar partition (A, B) with $A^{\mathcal{C}}_P \subseteq A$ has at least $k + 1$ clusters in $G[A]$ and, therefore, is not valid. Reduction Rule 4.3 will then be applied before any further application of Reduction Rule 4.4, and the constraint $\mathcal{C}$ will be rejected. ◀

▶ **Lemma 4.10.** *Along any root-leaf path in the search tree of the algorithm, Reduction Rule 4.5 and Branching Rules 4.6 and 4.7 are applied at most $k+1$ times in total.*

Proof. Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint obtained from an initial constraint via $k+1$ applications of Reduction Rule 4.5 and Branching Rules 4.6 and 4.7, and an arbitrary number of applications of the other rules. Let k_s denote the number of singleton clusters in $G[A']$. Observe that each application of Reduction Rule 4.5 or Branching Rules 4.6 and 4.7 makes a vertex of $A^{\mathcal{C}}_* \subseteq A'$ permanent by placing it in $A^{\mathcal{C}}_P$ or $B^{\mathcal{C}}_P$. By the description of all rules, a vertex will never be made permanent twice. Hence, out of the $k+1$ applications of Reduction Rule 4.5 and Branching Rules 4.6 and 4.7, at most k_s make the vertex from a singleton cluster of $G[A']$ permanent. Observe that Branching Rule 4.7 cannot make a vertex from a non-singleton cluster in $G[A']$ permanent. Thus, Reduction Rule 4.5 and Branching Rule 4.6 make at least $k-k_s+1$ vertices in the $k-k_s$ non-singleton clusters of $G[A']$ permanent. Since $k-k_s+1 \geq 1$, this also implies that a non-singleton cluster exists. By the pigeonhole principle, out of the $k-k_s+1$ vertices that are made permanent by Reduction Rule 4.5 and Branching Rule 4.6, two are from the same non-singleton cluster in $G[A']$. Since both Reduction Rule 4.5 and Branching Rule 4.6 only move vertices from $A^{\mathcal{C}}_*$ to $B^{\mathcal{C}}_P$, it follows that $B^{\mathcal{C}}_P$ contains two adjacent vertices. Then the constraint $\mathcal{C}$ will be rejected by Reduction Rule 4.3, which is applied before any further rule is applied. ◀

▶ **Theorem 4.11.** Inductive Monopolar Recognition *can be solved in $\mathcal{O}(2^k \cdot k^3 \cdot \Delta)$ time, where Δ is the maximum degree of G.*

Proof. We call a leaf of the search tree associated with a constraint to which no rule applies an *exhausted leaf*. By Lemma 4.8 and the correctness of the rules, G has a valid monopolar partition if and only if for at least one exhausted leaf node, the partition $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$, induced by the constraint $\mathcal{C}$ associated with that node, is a valid monopolar partition. Hence, if the search tree has an exhausted leaf for which the partition $(A^{\mathcal{C}}_* \cup A^{\mathcal{C}}_P, B^{\mathcal{C}}_* \cup B^{\mathcal{C}}_P)$, induced by the constraint $\mathcal{C}$ associated with that node, is a valid monopolar partition, the algorithm answers "yes"; otherwise, it answers "no". Therefore, the described search-tree algorithm correctly decides an instance of Inductive Monopolar Recognition.

To upper bound the running time, let $\mathcal{T}$ denote the search tree of the algorithm. By Lemma 4.10, Branching Rules 4.6 and 4.7 are applied at most $k+1$ times in total along any root-leaf path in $\mathcal{T}$. It follows that the depth of $\mathcal{T}$ is at most $k+2$. As each of the branching rules is a two-way branch, $\mathcal{T}$ is a binary tree, and thus the number of leaves in $\mathcal{T}$ is $\mathcal{O}(2^k)$.

The running time along any root-leaf path in $\mathcal{T}$ is dominated by the overall time taken along the path to test the applicability of the reduction and branching rules, and to apply them. By Lemma 4.9 and Lemma 4.10, along any root-leaf path in $\mathcal{T}$ the total number of applications of Reduction Rules 4.4 and 4.5 and Branching Rules 4.6 and 4.7 is $\mathcal{O}(k)$. Reduction Rule 4.3 is applied once before the application of each of the aforementioned rules. It follows that the total number of applications of all rules along any root-leaf path in $\mathcal{T}$ is $\mathcal{O}(k)$. Moreover, $\mathcal{T}$ has $\mathcal{O}(2^k)$ leaves as argued before. Therefore, we test for the applicability of the rules and apply them, or use the check of Lemma 4.8, at most $\mathcal{O}(2^k \cdot k)$ times. We next upper bound the time to test the applicability of the rules and to apply them by $\mathcal{O}(k^2 \cdot \Delta)$.

Let $\mathcal{C} = (A^{\mathcal{C}}_*, A^{\mathcal{C}}_P, B^{\mathcal{C}}_*, B^{\mathcal{C}}_P)$ be a constraint associated with a node in $\mathcal{T}$. Observe that each cluster in $G[A^{\mathcal{C}}_*]$ has size $\mathcal{O}(\Delta)$. Since $G[A^{\mathcal{C}}_*]$ has at most k clusters, this implies that $|A^{\mathcal{C}}_*| \leq k \cdot \Delta$. Thus, in $\mathcal{O}(k \cdot \Delta)$ time, we can compute a list of all clusters in $G[A^{\mathcal{C}}_*]$ and the size of each cluster. The same holds for $G[A']$. Observe that we can always check in $\mathcal{O}(1)$ time, for a given vertex v, whether v is contained in A', $A^{\mathcal{C}}_*$, $A^{\mathcal{C}}_P$, $B^{\mathcal{C}}_*$, or $B^{\mathcal{C}}_P$ and, in case v is contained in A' or $A^{\mathcal{C}}_*$, we can find the index and the size of the cluster that contains v.

Moreover, by Lemma 4.9 and 4.10, we can assume that $|A_P^{\mathcal{C}}| = \mathcal{O}(k)$, and by Lemma 4.10, we can assume that $|B_P^{\mathcal{C}}| = \mathcal{O}(k)$.

To test the applicability of Reduction Rules 4.3 and 4.4, we check whether $G[A_P^{\mathcal{C}}]$ is a cluster graph with at most k clusters, whether $G[B_P^{\mathcal{C}}]$ is an independent set, and whether there is an edge with one endpoint in $B_P^{\mathcal{C}}$ and the other endpoint in $B_*^{\mathcal{C}}$. This can be done in $\mathcal{O}(k \cdot \Delta)$ time since $|A_P^{\mathcal{C}}| = \mathcal{O}(k)$ and $|B_P^{\mathcal{C}}| = \mathcal{O}(k)$.

To test the applicability of Reduction Rule 4.5, we consider each pair v, w of vertices in $A_P^{\mathcal{C}}$. If v and w are adjacent, then in $\mathcal{O}(\Delta)$ time we can check whether there is a vertex $u \in A_*^{\mathcal{C}}$ such that u is adjacent to exactly one of v and w. If v and w are not adjacent, then in $\mathcal{O}(\Delta)$ time we can check whether they have a common neighbor in $A_*^{\mathcal{C}}$. If neither condition applies to any pair v, w, then Reduction Rule 4.5 does not apply. Overall, this test takes $\mathcal{O}(k^2 \cdot \Delta)$ time.

To test the applicability of Branching Rule 4.6, we can check for each vertex v of the at most k vertices of $A_P^{\mathcal{C}}$ in $\mathcal{O}(\Delta)$ time whether v has neighbors in two different clusters of $A_*^{\mathcal{C}}$, or whether there are two vertices u, w in the same cluster of $A_*^{\mathcal{C}}$ such that v is adjacent to u but not adjacent to w. If one of the two cases applies to some vertex $v \in A_P^{\mathcal{C}}$, then Branching Rule 4.6 applies to v. Otherwise, there is no P_3 containing exactly one vertex from $A_P^{\mathcal{C}}$ and exactly two vertices from $A_*^{\mathcal{C}}$, and Branching Rule 4.6 does not apply. Hence, the applicability of Branching Rule 4.6 can be tested in $\mathcal{O}(k \cdot \Delta)$ time.

To test the applicability of Branching Rule 4.7, we can check in $\mathcal{O}(k)$ time, whether $G[A_*^{\mathcal{C}}]$ contains a singleton cluster that is also a singleton cluster of $G[A']$.

All rules can trivially be applied in $\mathcal{O}(1)$ time if they were found to be applicable. Hence, the running time to test and apply any of the rules is $\mathcal{O}(k^2 \cdot \Delta)$.

Finally, if none of the rules applies, then we can check in $\mathcal{O}(k \cdot \Delta)$ time whether the number of clusters in $G[A_*^{\mathcal{C}} \cup A_P^{\mathcal{C}}]$ is at most k. Hence, the algorithm runs in $\mathcal{O}(2^k \cdot k^3 \cdot \Delta)$ time in total. ◀

Given the above theorem, Corollary 3.2 immediately implies Theorem 1.1.

5 An FPT algorithm for 2-Subcoloring

In this section, we give an FPT algorithm for 2-Subcoloring parameterized by the smallest number of clusters in the two parts. Although the general approach is similar to the approach used for Monopolar Recognition, in that it relies on the inductive recognition technique and the notion of constraints, the algorithm is substantially more complex. In particular, the notion of constraints and the reduction and branching rules are more involved, mainly due to the much more complicated structure of 2-subcolorable graphs.

Throughout, given a graph G and a nonnegative integer k, we call a 2-subcoloring (A, B) of G *valid* if $G[A]$ has at most k cliques. Using the inductive recognition approach, we need a parameterized inductive recognizer for the following problem:

Inductive 2-Subcoloring
Input: A graph $G = (V, E)$, a vertex $v \in V$, and a valid 2-subcoloring (A', B') of $G' = G - v$.
Question: Does G have a valid 2-subcoloring (A, B)?

Fix an instance of Inductive 2-Subcoloring with a graph $G = (V, E)$, a vertex $v \in V$, and a valid 2-subcoloring (A', B') of $G' = G - v$. Let $n = |V|$. We again apply a search-tree algorithm that starts with initial partitions $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$ of V, derived from (A', B'), that are

not necessarily 2-subcolorings of G. Then, we try to "repair" those partitions by moving vertices between $A^{\mathcal{C}}_*$ and $B^{\mathcal{C}}_*$ to form a valid 2-subcoloring (A, B) of G. As before, each node in the search tree is associated with one constraint.

▶ **Definition 5.1.** A *constraint* $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ consists of a partition $(A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n)$ of V and two vertex sets $A^{\mathcal{C}}_P \subseteq A^{\mathcal{C}}_*$ and $B^{\mathcal{C}}_P \subseteq B^{\mathcal{C}}_*$, where $A^{\mathcal{C}}_* = \bigcup_{i=1}^{k} A^{\mathcal{C}}_i$ and $B^{\mathcal{C}}_* = \bigcup_{i=1}^{n} B^{\mathcal{C}}_i$, such that for any $i \neq j$:
- u and w are not adjacent for any $u \in A^{\mathcal{C}}_i \setminus A^{\mathcal{C}}_P$ and $w \in A^{\mathcal{C}}_j \setminus A^{\mathcal{C}}_P$, and
- u and w are not adjacent for any $u \in B^{\mathcal{C}}_i \setminus B^{\mathcal{C}}_P$ and $w \in B^{\mathcal{C}}_j \setminus B^{\mathcal{C}}_P$.

We explicitly allow (some of) the sets of the partition $(A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n)$ of V to be empty. The vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ are called *permanent* vertices of the constraint.

The permanent vertices in $A^{\mathcal{C}}_P$ and $B^{\mathcal{C}}_P$ in the definition will correspond precisely to those vertices that have switched sides during the algorithm. We refer to the sets $A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k$ and $B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n$ as *groups*; during the algorithm, $G[A^{\mathcal{C}}_*]$ and $G[B^{\mathcal{C}}_*]$ are not necessarily cluster graphs and, thus, we avoid using the term clusters.

We now define the notion of a valid 2-subcoloring fulfilling a constraint. Intuitively speaking, a constraint $\mathcal{C}$ is fulfilled by a bipartition (A, B) if (A, B) respects the assignment of the permanent vertices stipulated by $\mathcal{C}$, and if all vertices that do not switch sides stay in the bipartition (A, B) in the same groups they belong to in $\mathcal{C}$. This notion is formalized as follows.

▶ **Definition 5.2.** A constraint $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ is *fulfilled* by a bipartition (A, B) of V if $G[A]$ is a cluster graph with k clusters $A_1, \ldots, A_k$ and $G[B]$ is a cluster graph with n clusters $B_1, \ldots, B_n$ (some of the clusters may be empty) such that:
1. for each $i \in [k]$, $A_i \cap A^{\mathcal{C}}_* \subseteq A^{\mathcal{C}}_i$;
2. for each $i \in [n]$, $B_i \cap B^{\mathcal{C}}_* \subseteq B^{\mathcal{C}}_i$;
3. $A^{\mathcal{C}}_P \subseteq A$; and
4. $B^{\mathcal{C}}_P \subseteq B$.

We now need a set of initial constraints to jumpstart the search-tree algorithm.

▶ **Lemma 5.3** (⋆). *Let $A'_1, \ldots, A'_k$ denote the clusters of $G'[A']$ and let $B'_1, \ldots, B'_n$ denote the clusters of $G'[B']$. Herein, if there are less than k clusters in $G[A']$ or less than n clusters in $G[B']$, we add an appropriate number of empty sets. By relabeling, we may assume that only $B'_1, \ldots, B'_i$ contain neighbors of v, and $B'_{i+1} = \emptyset$. Each valid 2-subcoloring (A, B) of G fulfills either:*
- *$(A'_1, \ldots, A'_j \cup \{v\}, \ldots, A'_k, B'_1, \ldots, B'_n, \{v\}, \emptyset)$ for some $j \in [k]$, or*
- *$(A'_1, \ldots, A'_k, B'_1, \ldots, B'_j \cup \{v\}, \ldots, B'_n, \emptyset, \{v\})$ for some $j \in [i+1]$.*

Now that we have identified the initial constraints, we turn to the search-tree algorithm and its reduction and branching rules. A crucial ingredient to the rules and the analysis of the running time is the following lemma. A consequence of the lemma is that if the number of initial constraints is too large, then most of them should be rejected immediately.

▶ **Lemma 5.4** (⋆). *Let $\mathcal{C} = (A^{\mathcal{C}}_1, \ldots, A^{\mathcal{C}}_k, B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n, A^{\mathcal{C}}_P, B^{\mathcal{C}}_P)$ be a constraint and let (A, B) be any valid 2-subcoloring of G fulfilling $\mathcal{C}$. If $u \in V$ has neighbors in more than $k+1$ groups among $B^{\mathcal{C}}_1, \ldots, B^{\mathcal{C}}_n$, then $u \in A$.*

Lemma 5.4 implies that if v has neighbors in more than $k+1$ clusters of A', then we should immediately reject the initial constraints generated by Lemma 5.3 that place v in $B^{\mathcal{C}}_P$. Hence, we obtain the following corollary of Lemmas 5.3 and 5.4.

▶ **Corollary 5.5** (⋆). *Lemma 5.3 generates at most* $2k+2$ *constraints that are not immediately rejected.*

As before, each non-root node of the search tree is associated with a constraint. The root of the search tree is a dummy node with children associated with the constraints generated by Lemma 5.3 that are not immediately rejected due to Lemma 5.4. We now give two reduction rules, which are applied exhaustively to each search-tree node, in the order they are presented.

Let $\mathcal{C} = (A_1^{\mathcal{C}}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}})$ be a constraint. The first reduction rule identifies some obvious cases in which the constraint cannot be fulfilled.

▶ **Reduction Rule 5.6** (⋆). *If* $G[A_P^{\mathcal{C}}]$ *or* $G[B_P^{\mathcal{C}}]$ *is not a cluster graph, or if there are* $i \neq j$ *such that there is an edge between* $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$ *and* $A_j^{\mathcal{C}} \cap A_P^{\mathcal{C}}$ *or an edge between* $B_i^{\mathcal{C}} \cap B_P^{\mathcal{C}}$ *and* $B_j^{\mathcal{C}} \cap B_P^{\mathcal{C}}$*, then reject* $\mathcal{C}$*.*

The second reduction rule is the natural consequence of Lemma 5.4.

▶ **Reduction Rule 5.7** (⋆). *If there is a vertex* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *that has neighbors in more than* $k+1$ *groups of* $B_*^{\mathcal{C}}$*, then set* $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$*.*

The algorithm contains a single branching rule. This rule, called switch(u), uses branching to fix a vertex u in one of the clusters in one of the parts of the 2-subcoloring. The vertices to which switch() must be applied are identified by *switching rules*. We say that a switching rule that calls for applying switch(u) is *correct* if for all valid 2-subcolorings (A, B) of G fulfilling $\mathcal{C}$, we have $u \in A_*^{\mathcal{C}} \cap B$ or $u \in B_*^{\mathcal{C}} \cap A$. We first describe the switching rules, and then describe switch(u). Recall from Fact 2.1 that cluster graphs do not contain induced P_3's.

The first switching rule identifies vertices that are not adjacent to some permanent vertices of their group.

▶ **Switching Rule 5.8.** *If there is a vertex* u *such that* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *is not adjacent to some vertex in* $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$*, or* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *is not adjacent to some vertex in* $B_i^{\mathcal{C}} \cap B_P^{\mathcal{C}}$*, then call* switch$(u)$*.*

The second switching rule finds vertices that have permanent neighbors in another group.

▶ **Switching Rule 5.9.** *If there is a vertex* u *such that* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has a neighbor in* $A_P^{\mathcal{C}} \setminus A_i^{\mathcal{C}}$*, or* $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ *and* u *has a neighbor in* $B_P^{\mathcal{C}} \setminus B_i^{\mathcal{C}}$*, then call* switch$(u)$*.*

Now, we describe switch(u), which is a combination of a reduction rule and a branching rule. There are two main scenarios that we distinguish. If u has permanent neighbors in the other part, then there is only one choice for assigning u to a group. Otherwise, we branch into all (up to symmetry when a group is empty) possibilities to place u into a group. It is important to note that the switching rules never apply switch(u) to a permanent vertex.

▶ **Branching Rule 5.10** (switch(u))**.**

- *If* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has a permanent neighbor in some* $B_j^{\mathcal{C}}$*, then set* $A_i^{\mathcal{C}} \leftarrow A_i^{\mathcal{C}} \setminus \{u\}$*,* $B_j^{\mathcal{C}} \leftarrow B_j^{\mathcal{C}} \cup \{u\}$*,* $B_P^{\mathcal{C}} \leftarrow B_P^{\mathcal{C}} \cup \{u\}$*.*
- *If* $u \in A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ *and* u *has only nonpermanent neighbors in* $B_*^{\mathcal{C}}$*, then, for each* $B_j^{\mathcal{C}}$ *such that* $N(u) \cap B_j^{\mathcal{C}} \neq \emptyset$ *and* $B_j^{\mathcal{C}} \cap B_P^{\mathcal{C}} = \emptyset$*, and for one* $B_j^{\mathcal{C}}$ *such that* $B_j^{\mathcal{C}} = \emptyset$ *(chosen arbitrarily), branch into a branch associated with the constraint* $(A_1^{\mathcal{C}}, \ldots, A_i^{\mathcal{C}} \setminus \{u\}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_j^{\mathcal{C}} \cup \{u\}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}} \cup \{u\})$*.*

- *If $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ and u has a permanent neighbor in some $A_j^{\mathcal{C}}$, then set $B_i^{\mathcal{C}} \leftarrow B_i^{\mathcal{C}} \setminus \{u\}$, $A_j^{\mathcal{C}} \leftarrow A_j^{\mathcal{C}} \cup \{u\}$, $A_P^{\mathcal{C}} \leftarrow A_P^{\mathcal{C}} \cup \{u\}$.*
- *If $u \in B_i^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ and u has only nonpermanent neighbors in $A_*^{\mathcal{C}}$, then for each $A_j^{\mathcal{C}}$ with $A_j^{\mathcal{C}} \cap A_P^{\mathcal{C}} = \emptyset$, branch into a branch associated with the constraint $(A_1^{\mathcal{C}}, \ldots, A_j^{\mathcal{C}} \cup \{u\}, \ldots, A_k^{\mathcal{C}}, B_1^{\mathcal{C}}, \ldots, B_i^{\mathcal{C}} \setminus \{u\}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}} \cup \{u\}, B_P^{\mathcal{C}})$; if no such $A_j^{\mathcal{C}}$ exists, reject $\mathcal{C}$.*

If none of the previous rules applies, then the constraint directly gives a solution:

▶ **Lemma 5.11.** *Let $\mathcal{C} = (A_1^{\mathcal{C}}, \ldots, B_n^{\mathcal{C}}, A_P^{\mathcal{C}}, B_P^{\mathcal{C}})$ be a constraint such that none of the rules applies. Then $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$ is a valid 2-subcoloring.*

Proof. We need to show that $G[A_*^{\mathcal{C}}]$ and $G[B_*^{\mathcal{C}}]$ are cluster graphs and that $G[A_*^{\mathcal{C}}]$ has at most k clusters. First, we claim that $G[A_i^{\mathcal{C}}]$ is a clique for every $i = 1, \ldots, k$. Every vertex in $A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ is adjacent to every vertex in $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$; otherwise, Switching Rule 5.8 applies. Any two vertices in $A_i^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ are also adjacent, because they are in the same cluster of A'. It remains to show that $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ is a clique. By the description of switch(u), if a vertex x is placed into $A_i^{\mathcal{C}}$ and $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}} \neq \emptyset$, then x is adjacent to a vertex of $A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}$. Hence, $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ is connected. Since Reduction Rule 5.6 does not apply, $G[A_i^{\mathcal{C}} \cap A_P^{\mathcal{C}}]$ does not contain an induced P_3 and, thus, it is a clique. Hence, $G[A_i^{\mathcal{C}}]$ is a clique, as claimed.

Second, we claim that there are no edges between $A_i^{\mathcal{C}}$ and $A_j^{\mathcal{C}}$, where $i \neq j$. Suppose for the sake of a contradiction that e is such an edge. Since Reduction Rule 5.6 does not apply, e is incident with at least one nonpermanent vertex. Since Switching Rule 5.9 does not apply, e is in fact incident with two nonpermanent vertices. Then e cannot exist by the definition of a constraint. The claim follows.

The combination of the above claims shows that $G[A_*^{\mathcal{C}}]$ is a cluster graph with the clusters $A_i^{\mathcal{C}}$ (some of which may be empty) and, thus, has at most k clusters. Similar arguments show that $G[B_*^{\mathcal{C}}]$ is a cluster graph: in the above argument, we used only Reduction Rule 5.6 and Switching Rules 5.8 and 5.9, which apply to vertices in $A_*^{\mathcal{C}}$ and $B_*^{\mathcal{C}}$ symmetrically. ◀

▶ **Theorem 5.12.** Inductive 2-Subcoloring *can be solved in $\mathcal{O}(k^{2k+2} \cdot (|V| + |E|))$ time.*

Proof. Given the valid 2-subcoloring (A', B') of G', we use Lemma 5.3 to generate a set of initial constraints, and reject those which cannot be fulfilled due to Lemma 5.4. By Corollary 5.5, at most $2k+2$ initial constraints remain, which are associated with the children of the (dummy) root node. For each node of the search tree, we first exhaustively apply the reduction rules on the associated constraint. Afterwards, if there exists a vertex u to which a switching rule applies, then we apply switch(u). If switch(u) does not branch but instead reduces to a new constraint, then we apply the reduction rules exhaustively again, etc.

A *leaf* of the search tree is a node associated either with a constraint that is rejected, or with a constraint to which no rule applies. The latter is called an *exhausted leaf*. If the search tree has an exhausted leaf, then the algorithm answers "yes"; otherwise, it answers "no". By the correctness of the reduction, branching, and switching rules, and by Lemma 5.11, graph G has a valid 2-subcoloring if and only if the search tree has at least one exhausted leaf node. Therefore, the described search-tree algorithm correctly decides an instance of Inductive 2-Subcoloring.

We now bound the running time of the algorithm. Observe that each described reduction rule and the branching rule switch() either rejects the constraint or makes a vertex permanent. Hence, along each root-leaf path, $\mathcal{O}(n)$ rules are applied. Each rule can trivially be tested for applicability and applied in polynomial time. Hence, it remains to bound the number of leaves of the search tree.

As mentioned, at the root of the search tree, we create at most $\mathcal{O}(n)$ constraints, out of which at most $2k+2$ constraints do not correspond to leaf nodes by Lemma 5.3, Corollary 5.5

and Reduction Rule 5.7. The only branches are created by a call to switch(u) for a vertex u that has only non-permanent neighbors in the other part of the bipartition $(A_*^{\mathcal{C}}, B_*^{\mathcal{C}})$. Observe that if such a vertex $u \in B_*^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$, then in each constraint $\mathcal{C}'$ constructed by switch(u) the number of groups in $A_*^{\mathcal{C}'}$ that have at least one permanent vertex increases by one compared to $\mathcal{C}$. Since each constraint has k groups in $A_*^{\mathcal{C}}$, this branch can be applied at most k times along each root-leaf path in the search tree.

Similarly, if $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$, then in each constraint $\mathcal{C}'$ constructed by switch(u) the number of groups in $B_*^{\mathcal{C}}$ that have at least one permanent vertex increases by one compared to $\mathcal{C}$. We claim that, if $B_*^{\mathcal{C}}$ has k groups with a permanent vertex, then u has a neighbor in $B_P^{\mathcal{C}}$. First, each permanent vertex in $B_*^{\mathcal{C}}$ is part of A' by the description of the rules. Moreover, the permanent vertices of the k groups in $B_*^{\mathcal{C}}$ with a permanent vertex stem from k different clusters in $G[A']$, because switch() places a vertex of $A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ that has neighbors in $B_P^{\mathcal{C}}$ in the same group as its neighbors in $B_P^{\mathcal{C}}$. This implies that one of the clusters in $G[A']$ that the permanent vertices stem from contains u. Hence, u is adjacent to a vertex in $B_P^{\mathcal{C}}$, as claimed. The claim implies that if $B_*^{\mathcal{C}}$ has k groups with a permanent vertex, then switch(u) applied to a vertex $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ does not branch. Hence, also the branch of switch(u) in which $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ is performed at most k times along each root-leaf path in the search tree.

In summary, the branchings of switch(u) in which $u \in B_*^{\mathcal{C}} \setminus B_P^{\mathcal{C}}$ branch into at most k cases, and the branchings in which $u \in A_*^{\mathcal{C}} \setminus A_P^{\mathcal{C}}$ branch into at most $k+2$ cases, since Reduction Rule 5.7 does not apply. Observe that k of the initial constraints have already one group in $A_*^{\mathcal{C}}$ with a permanent vertex, and the other $k+1$ initial constraints have one group in B with a permanent vertex. Thus, if the initial constraint $\mathcal{C}$ places v in $A_P^{\mathcal{C}}$, then the overall number of constraints from $\mathcal{C}$ by branching is at most $k^{k-1} \cdot (k+2)^k$. If the initial constraint $\mathcal{C}$ places v in $B_P^{\mathcal{C}}$, then the overall number of constraints created from $\mathcal{C}$ by branching is at most $k^k \cdot (k+2)^{k-1}$. Altogether, the number of constraints created by branching is thus

$$(2k+1) \cdot k^k \cdot (k+2)^k = (2k+1) \cdot k^k \cdot k^k \cdot [(1+1/(k/2))^{k/2}]^2 = \mathcal{O}(k^{2k+1})$$

after noting that $[(1+1/(k/2))^{k/2}]^2 = \mathcal{O}(1)$. This provides the claimed bound on the number of leaves of the search tree. By performing an analysis similar to that in the proof of Theorem 4.11, we can show that the time spent along each root-leaf path of the search tree is $\mathcal{O}(k \cdot (|V| + |E|))$, which yields an overall running time of $\mathcal{O}(k^{2k+2} \cdot (|V| + |E|))$ for Inductive 2-Subcoloring. ◀

Given the above theorem, Corollary 3.2 immediately implies Theorem 1.2.

References

1. Demetrios Achlioptas. The complexity of G-free colourability. *Discrete Mathematics*, 165–166(0):21–30, 1997.
2. Hajo Broersma, Fedor V. Fomin, Jaroslav Nešetřil, and Gerhard J. Woeginger. More about subcolorings. *Computing*, 69(3):187–203, 2002.
3. Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10:16, 2015.
4. Ross Churchley and Jing Huang. List monopolar partitions of claw-free graphs. *Discrete Mathematics*, 312(17):2545–2549, 2012.
5. Ross Churchley and Jing Huang. Solving partition problems with colour-bipartitions. *Graphs and Combinatorics*, 30(2):353–364, 2014.
6. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

7 Reinhard Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Berlin, Heidelberg, 2013.

9 Tınaz Ekim, Pavol Hell, Juraj Stacho, and Dominique de Werra. Polarity of chordal graphs. *Discrete Applied Mathematics*, 156(13):2469–2479, 2008.

10 E. M. Eschen and X. Wang. Algorithms for unipolar and generalized split graphs. *Discrete Applied Mathematics*, 162:195–201, 2014.

11 Alastair Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *The Electronic Journal of Combinatorics*, 11(1):R46, 2004.

12 Jirí Fiala, Klaus Jansen, Van Bang Le, and Eike Seidel. Graph subcolorings: Complexity and algorithms. *SIAM Journal on Discrete Mathematics*, 16(4):635–650, 2003.

13 Peter L. Hammer and Bruno Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

14 Sudeshna Kolay and Fahad Panolan. Parameterized Algorithms for Deletion to (r, ℓ)-Graphs. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 45 of *LIPIcs*, pages 420–433. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

15 Van Bang Le and Ragnar Nevries. Complexity and algorithms for recognizing polar and monopolar graphs. *Theoretical Computer Science*, 528:1–11, 2014.

16 Colin McDiarmid and Nikola Yolov. Recognition of unipolar and generalised split graphs. *Algorithms*, 8(1):46–59, 2015.

17 C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

18 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

19 Juraj Stacho. On 2-subcolourings of chordal graphs. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium*, volume 4957 of *LNCS*, pages 544–554. Springer, 2008.

20 Regina I. Tyshkevich and Arkady A. Chernyak. Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR, ser. Fiz. Mat. Nauk*, 6:16–23, 1985. In Russian.

21 Regina I. Tyshkevich and Arkady A. Chernyak. Decomposition of graphs. *Cybernetics and Systems Analysis*, 21(2):231–242, 1985.