List coloring in the absence of a linear forest ⋆

Jean-François Couturier[1], Petr A. Golovach[2],
Dieter Kratsch[1], and Daniël Paulusma[2] ⋆⋆

[1]Laboratoire d'Informatique Théorique et Appliquée,
Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
{couturier,kratsch}@univ-metz.fr
[2]School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, United Kingdom
{petr.golovach,daniel.paulusma}@durham.ac.uk

Abstract. The k-Coloring problem is to decide whether a graph can be colored with at most k colors such that no two adjacent vertices receive the same color. The List k-Coloring problem requires in addition that every vertex u must receive a color from some given set $L(u) \subseteq \{1, \ldots, k\}$. Let P_n denote the path on n vertices, and $G + H$ and rH the disjoint union of two graphs G and H and r copies of H, respectively. For any two fixed integers k and r, we show that List k-Coloring can be solved in polynomial time for graphs with no induced $rP_1 + P_5$, hereby extending the result of Hoàng, Kamiński, Lozin, Sawada and Shu for graphs with no induced P_5. Our result is tight; we prove that for any graph H that is a supergraph of $P_1 + P_5$ with at least 5 edges, already List 5-Coloring is NP-complete for graphs with no induced H.

1 Introduction

Graph coloring involves the labeling of the vertices of some given graph by integers called colors such that no two adjacent vertices receive the same color. The corresponding k-Coloring problem is to decide whether a graph can be colored with at most k colors. Due to the fact that k-Coloring is NP-complete for any fixed $k \geq 3$, there has been considerable interest in studying its complexity when restricted to certain graph classes. One of the most well-known results in this respect is due to Grötschel, Lovász, and Schrijver [10] who show that k-Coloring is polynomial-time solvable for perfect graphs. More information on this classic result and on the general motivation, background and related work on coloring problems restricted to special graph classes can be found in several surveys [21, 23] on this topic.

We continue the study of the computational complexity of the k-Coloring problem and related problems, in particular List k-Coloring when restricted

⋆ An extended abstract of this paper has been accepted to the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2011).

⋆⋆ This work has been supported by ANR Blanc AGAPE (ANR-09-BLAN-0159-03) and EPSRC (EP/G043434/1).

to graph classes defined by one or more forbidden induced subgraphs. Such problems have been studied in many papers by different groups of researchers [3–7, 11, 13–17, 20, 24]. Before we summarize these results and explain our new results, we first state the necessary terminology and notations.

1.1 Terminology

We only consider finite undirected graphs $G = (V, E)$ without loops and multiple edges. We sometimes denote the vertex set of G by V_G. The subgraph of $G = (V, E)$ induced by $U \subseteq V$ is denoted by $G[U]$. We refer to the textbook by Bondy and Murty [2] for any undefined graph terminology.

The graph P_n denotes the path on n vertices. The disjoint union of two graphs G and H is denoted $G + H$, and the disjoint union of r copies of G is denoted rG. A *linear forest* is the disjoint union of a collection of paths. Let $\{H_1, \ldots, H_p\}$ be a set of graphs. We say that a graph G is $(H_1, \ldots, H_p)$*-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we sometimes write H_1-free instead of (H_1)-free.

A *(vertex) coloring* of a graph $G = (V, E)$ is a mapping $\phi : V \to \{1, 2, \ldots\}$ such that $\phi(u) \neq \phi(v)$ whenever $uv \in E$. Here, $\phi(u)$ is referred to as the *color* of u. A k*-coloring* of G is a coloring ϕ of G with $\phi(V) \subseteq \{1, \ldots, k\}$. Here, we used the notation $\phi(U) = \{\phi(u) \mid u \in U\}$ for $U \subseteq V$. If G has a k-coloring, then G is called k*-colorable*. Recall that the problem k-Coloring is to decide whether a given graph admits a k-coloring. Here, k is *fixed*, i.e., not part of the input. If k is part of the input then we denote the problem as Coloring. The optimization version of this problem is to determine the *chromatic number* of a graph, i.e., the smallest k such that G has a k-coloring.

A *list assignment* of a graph $G = (V, E)$ is a function L that assigns a list $L(u)$ of so-called *admissible* colors to each $u \in V$. If $L(u) \subseteq \{1, \ldots, k\}$ for $u \in V$, then L is also called a k*-list assignment*. Equivalently, L is a k-list assignment if $|\bigcup_{u \in V} L(u)| \leq k$. We say that a coloring $\phi : V \to \{1, 2, \ldots\}$ *respects* L if $\phi(u) \in L(u)$ for all $u \in V$. For a fixed integer k, the List k-Coloring problem has as input a graph G with a k-list assignment L and asks whether G has a coloring that respects L. If $|L(u)| = 1$ for every vertex u of some subset $W \subseteq V$ and $L(u) = \{1, \ldots, k\}$ for $u \in V \setminus W$, then we obtain the k-Precoloring Extension problem.

1.2 Related work

Král', Kratochvíl, Tuza and Woeginger [15] completely determined the computational complexity of Coloring for graph classes characterized by a forbidden induced subgraph and achieved the following dichotomy.

Theorem 1 ([15]). *Let H be a fixed graph. If H is a (not necessarily proper) induced subgraph of P_4 or of $P_1 + P_3$ then* Coloring *can be solved in polynomial time for H-free graphs; otherwise it is* NP*-complete for H-free graphs.*

Theorem 1 can be extended by considering the computational complexity of Coloring for $\mathcal{H}$-free graphs where $\mathcal{H}$ is a family of two (or more) graphs. Some initial results have been obtained by Král' et al. [15], Schindl [22] and a number of authors studying the $\mathcal{H}$-free graphs, in which one of the two graphs in $\mathcal{H}$ is the triangle [5, 7, 13, 19]. Another extension is to classify the computational complexity of k-Coloring and other variants of coloring for H-free graphs where k is a fixed integer and H is a fixed graph. The complexity classifications in both directions are far from being finished. In this paper we consider the second direction. We focus on the case when H is a linear forest. Below we justify this.

Kamiński and Lozin [13] showed that for any $k \geq 3$, the k-Coloring problem is NP-complete for the class of graphs of girth (the length of a shortest induced cycle) at least p for any fixed $p \geq 3$. Their result implies that for any $k \geq 3$, the k-Coloring problem is NP-complete for the class of H-free graphs if H contains a cycle. Holyer [12] showed that 3-Coloring is NP-complete on line graphs. Later, Leven and Galil [18] extended this result by showing that k-Coloring is also NP-complete on line graphs for $k \geq 4$. Because line graphs are claw-free, i.e., they have no induced $K_{1,3}$, we find that for $k \geq 3$, the k-Coloring problem is NP-complete for the class of H-free graphs if H is a forest that contains a vertex with degree at least 3. Hence, only the case in which H is a linear forest remains.

It is known that 4-Coloring is NP-complete for P_8-free graphs [4] and that 6-Coloring is NP-complete for P_7-free graphs [3]. On the contrary, Randerath and Schiermeyer [20] showed that 3-Coloring can be solved in polynomial time for P_6-free graphs. A result which was generalized by Broersma et al. [3] who showed that 3-Precoloring Extension can be solved in polynomial time for P_6-free graphs. Later, Broersma et al. [4] extended this result by showing that 3-Precoloring Extension can be solved in polynomial time for H-free graphs if H is a linear forest on at most 6 vertices. The proof methods of both papers [3, 4] can directly be applied to show exactly the same results for List 3-Coloring. For P_5-free graphs, Hoàng et al. [11] could show a stronger result; note that Coloring is NP-complete for P_5-free graphs due to Theorem 1.

Theorem 2 ([11]). *For any fixed integer k, the* List k-Coloring *problem can be solved in polynomial time for P_5-free graphs.*

1.3 Our new results

The aim of our paper is to generalize Theorem 2 as much as possible. We prove that for any fixed integers k and r, the List k-Coloring problem is polynomial-time solvable for (rP_1+P_5)-free graphs. In order to prove our result, we show that our input graphs have a dominating set of small size should they be k-colorable. Hence, we search for such a dominating set. If we find it, then we color its vertices in every possible way. Afterwards, we use the technique of "separating the color lists of independent sets" of Hoáng et al. [11] on each resulting instance. They successfully applied this technique for coloring P_5-free graphs, and our result for $(rP_1 + P_5)$-free graphs can be seen as a second example of its usefulness. We

present this technique in Section 2 in a more generic way. In order to obtain our result for $(rP_1 + P_5)$-free graphs we have to prove a number of additional structural results. This is done in Section 3. In Section 4 we show that our result is *tight* by proving that already List 5-Coloring is NP-complete for the class of H-free graphs whenever H has at least 5 edges and contains $P_1 + P_5$ as a subgraph.

2 A generic approach for coloring H-free graphs

We generalize the technique Hoáng et al. [11] used to prove Theorem 2.

Given a graph $G = (V, E)$ with a k-list assignment L, we use the following terminology. Two adjacent vertices u and v are *essential* if $L(u) \cap L(v) \neq \emptyset$; otherwise u and v are *non-essential*. We observe that u is an essential neighbor of v if and only if v is an essential neighbor of u. Two disjoint sets of vertices are *separated* for L if no vertex in one of them has an essential neighbor in the other. Let $\mathcal{L}$ be a set of k-list assignments of G with $L'(u) \subseteq L(u)$ for all $L' \in \mathcal{L}$ and all $u \in V$. Then L and $\mathcal{L}$ are *compatible* if the following holds: G has a coloring respecting L' for some $L' \in \mathcal{L}$ if G has a coloring respecting L. Note that the reverse implication holds by the definition of $\mathcal{L}$.

Assigning an admissible color to a vertex u does not influence the choice of admissible colors for its non-essential neighbors. Hence, in our coloring algorithm, we would like to branch in such a way that we obtain a compatible set of list assignments for which disjoint sets of vertices become separated. Then we can apply the algorithm recursively on smaller graphs induced by these disjoint sets. This idea has been applied more often but usually leads to a huge case analysis. However, Hoáng et al. [11] developed an elegant technique, which works well for P_5-free graphs. We present it in a more generic way below.

A subset $D \subseteq V$ is a *dominating* set of G if every vertex in G belongs to D or is adjacent to a vertex of D. In that case we also say that $G[D]$ is *dominating*. Suppose that we have ordered the vertices of D as $d_1, \ldots, d_p$. Then we can define (possibly empty) sets F_i for $i = 1, \ldots, p$ as follows. Let F_1 be the set of vertices in $V \setminus D$ adjacent to d_1, and for $i = 2, \ldots, p$, let F_i be the set of vertices in $V \setminus D$ adjacent to d_i but not to any d_h with $h \leq i - 1$. The sets $F_1, \ldots, F_p$ are called *fixed* sets for D. By this definition and because D is dominating, every vertex in $V \setminus D$ belongs to exactly one fixed set F_i. We note, however, that D can have several collections of fixed sets, depending on the ordering of the vertices of D. A subset $X \subseteq V$ is *independent* if there is no edge between any two vertices of X.

We call a graph H a *dominator-separator* graph if every connected H-free graph $G = (V, E)$ satisfies the following two properties.

(i) If G is k-colorable for some integer $k \geq 1$, then G has a dominating set D of at most $f(k)$ vertices, where f is a function that only depends on k.

(ii) There exists a polynomial-time algorithm that on input G, two independent sets X and Y that are subsets of two different fixed sets of a dominating set

of G and a k-list assignment L of G outputs a set $\mathcal{L}$ of k-list assignments of G with $L'(u) \subseteq L(u)$ for all $L' \in \mathcal{L}$ and all $u \in V$, such that

1. $\mathcal{L}$ is compatible with L;
2. $|\mathcal{L}| = O(h(k)n^{g(k)})$ for some functions $h(k)$ and $g(k)$ that only depend on k;
3. X and Y are separated for every $L' \in \mathcal{L}$.

By a straightforward translation of the proof of Hoàng et al. [11] one finds that for P_5-free graphs, $f(k) = k$ satisfies property (i), whereas $h(k) = k^k$ and $g(k) = k$ satisfy property (ii). Hence, P_5 is a dominator-separator. The following theorem generalizes their approach. Its proof is a reformulation of their proof in terms of dominator-separator graphs and can be found in Appendix A.

Theorem 3. *Let H be a dominator-separator graph, and let k be a fixed integer. Then* List k-Coloring *can be solved in polynomial time for H-free graphs.*

3 Coloring $(rP_1 + P_5)$-free graphs

In order to apply Theorem 3 we must prove that $rP_1 + P_5$ is a dominator-separator graph for any fixed r. We start with the following lemma.

Lemma 1. *Let G be an $(rP_1 + P_\ell)$-free graph for integers r and ℓ. If G contains an induced P_ℓ, then G contains a dominating induced $sP_1 + P_\ell$ for some $s < r$.*

Proof. Let P be an induced P_ℓ in G. Let U consist of all vertices in G that are neither on P nor a neighbor of a vertex of P. We choose a maximal independent set S in the subgraph of G induced by U. By maximality of S, all vertices in U are dominated by S. This means that $V_P \cup S$ is a dominating set in G. We define $s = |S|$ and observe that $V_P \cup S$ induces an $sP_1 + P_\ell$ in G. Because G is $(rP_1 + P_\ell)$-free, we find that $s < r$. Hence, G contains a dominating induced $sP_1 + P_\ell$ with $s < r$, as desired. □

A vertex subset K in a graph G is called a *clique* of G if there is an edge between any two vertices of K. Just as Hoáng et al. [11], we need the following result of Bacsó and Tuza [1] for the class of connected P_5-free graphs.

Theorem 4 ([1]). *Every connected P_5-free graph G has a dominating P_3 or a dominating clique.*

We are now ready to prove the following two lemmas which together show that $rP_1 + P_5$ is a dominator-separator for any fixed integer r.

Lemma 2. *Every connected $(rP_1 + P_5)$-free graph satisfies property (i).*

Proof. Let G be a connected $(rP_1 + P_5)$-free graph that is k-colorable for some integer $k \geq 1$. We must prove that G has a dominating set of size at most $f(k)$ for some function f that only depends on k. Below we show that G has a dominating set of size at most $\max\{3, k, r+4\}$. Then we may take the function f defined by $f(k) = \max\{3, k, r+4\}$ for all $k \geq 1$. This function only depends on k, because r is fixed.

First suppose that G is P_5-free, then G has a dominating P_3 or a dominating clique due to Theorem 4. Because G is k-colorable, any clique in G has at most k vertices. In the first case we obtain a dominating set of size 3. In the second case we obtain a dominating set of size at most k.

Now suppose that G is not P_5-free. By Lemma 1, G has a dominating induced $sP_1 + P_5$ for some $s < r$. Hence, we obtain a dominating set of size $s+5 \leq r+4$. This completes our proof of Lemma 2. □

Lemma 3. *Every connected $(rP_1 + P_5)$-free graph satisfies property (ii).*

Proof. Let $G = (V, E)$ be a connected $(rP_1 + P_5)$-free graph on n vertices with k-list assignment L. Let $D = \{d_1, \ldots, d_p\}$ be a dominating set of G, and let $F_1, \ldots, F_p$ be the collection of fixed sets for D. For some $1 \leq i < j \leq n$, let $X \subseteq F_i$ and $Y \subseteq F_j$ be two independent sets of G. Note that $i < j$ implies that d_i is not adjacent to any vertex in F_j, whereas d_j might be adjacent to one or more vertices of F_i.

Let the set C consist of every color c for which there exist two adjacent vertices $x \in X$ and $y \in Y$ such that $c \in L(x) \cap L(y)$. By definition, such x and y are essential neighbors of each other. If $C = \emptyset$, then X and Y are separated.

Suppose that $C \neq \emptyset$. We define a set X' as the set of all vertices in X that have an essential neighbor in Y, and a set Y' as the set of all vertices in Y that have an essential neighbor in X'. Because $C \neq \emptyset$, both X' and Y' are nonempty. Our goal is to reduce the size of X'. The reason is that when X' becomes empty, then C' will be empty, and consequently, X and Y will be separated.

Before we present our algorithm we first prove a useful claim that requires some extra terminology. We say that $x \in X'$ is *maximal* if there is no vertex in X' that has more neighbors in Y' than x has. We say that a vertex $z \in X'$ is an *associate* of x if at least $|Y'| - r + 1$ vertices in Y' are adjacent to x or z.

Claim 1. Let $x \in X'$ be maximal. Then either x is adjacent to all vertices of Y', or every vertex in X' that is adjacent to a non-neighbor of x in Y' is an associate of x.

We prove Claim 1 as follows. Suppose that $x \in X'$ is maximal and that x is not adjacent to all vertices of Y'. Let z be adjacent to a vertex $y \in Y'$ with $xy \notin E$. In order to derive at a contradiction, suppose that Y' contains r vertices $y_1, \ldots, y_r$ neither adjacent to x nor to z. Because x is maximal and z is adjacent to a vertex in Y', namely y, that is not adjacent to x, there exists a vertex $y' \in Y'$ adjacent to x but not to z. Recall that d_i is not adjacent to any vertex of Y'. Hence, we have found an induced $P_5 = y'xd_izy$ that together with $y_1, \ldots, y_r$ forms an induced $rP_1 + P_5$ in G. This is not possible, because G is $(rP_1 + P_5)$-free. Hence, we have proven Claim 1.

We are now ready to describe our algorithm that we use to prove property (ii). Recall that our goal is to reduce the size of X'. Hence, we branch on vertices of X'. Because X' may have a large size, we cannot branch by arbitrarily assigning colors to vertices of X'. Therefore, we do as follows as long as $X' \neq \emptyset$.

Determine a maximal vertex $x \in X'$ and start to branch on x.

Our algorithm either assigns to x a specific color c from C, creating a number of branches, or no color from C at all, yet another branch. In a branch of the first type we cannot only remove x from X' but we will also show that we may remove c from C; this is crucial for the running time analysis which we do afterwards. If x is not adjacent to every vertex in Y', then we may need to refine the branching by involving the associates of x. In a branch of the second type we remove every color in C from the list of x. Consequently, x can be removed from X' as desired (but we might not have decreased the size of C in this case).

The procedure `Reduce-to-empty-set` explains our approach in detail; see Pseudocode 2. Here, *updating* a list assignment after a vertex gets a color means removing this color from the list of every neighbor of that vertex. Further, for $x \in X'$, the set A_c^x denotes the set of associates of x that have color c in their list and that are adjacent to a vertex in Y' that is no neighbor of x. Finally, we note that at some places in this procedure we could also reduce the set Y'. However, for simplicity, we refrain from doing this, except in line 14 where it is necessary for the correctness. We will use the `Reduce-to-empty-set` procedure as a subroutine inside our separation algorithm called `Separator`; see Pseudocode 1. The output of `Separator` is a set $\mathcal{L}$ of k-list assignments of G; at the start we set $\mathcal{L} = \emptyset$.

`Separator`

input : sets X and Y
output : a set $\mathcal{L}$ of k-list assignments

1. determine the sets X', Y' and C
2. set $\mathcal{L} := \emptyset$
3. `Reduce-to-empty-set`$(X', Y', C, \mathcal{L})$
4. return $\mathcal{L}$

Pseudocode 1. Separating the two sets X and Y.

Having completed the overall description of our branching algorithm we prove that G satisfies property (ii) as follows. From the description of the procedure `Reduce-to-empty-set`, we conclude that each time we process a maximal vertex $x \in X'$, the size of X' reduces by at least one vertex. Hence, this procedure will always terminate, and when it does X' will be empty. Consequently, our algorithm `Separator` will terminate as well. When it does, it will return as output a set $\mathcal{L}$ of k-list assignments of G. The sets X and Y are separated for each k-list assignment of $\mathcal{L}$, because X', and consequently, C are empty for each

such list assignment. In other words, condition 3 of property (ii) is satisfied. We now show that conditions 1 and 2 are also satisfied.

The procedure `Reduce-to-empty-set` only reduces lists of vertices of G. As a consequence, every list assignment $L' \in \mathcal{L}$ has the property that $L'(u) \subseteq L(u)$ for all $u \in V$. We will show that L and $\mathcal{L}$ are compatible. In order to show this suppose that G has a coloring ϕ respecting L. Let $x \in X'$ be the maximal vertex that is under consideration. We show that in the search tree that represents our recursive procedure, there exist a branch that we can follow in order to prove the existence of a list assignment $L' \in \mathcal{L}$ that is respected by ϕ. The line numbers in our proof refer to lines in the `Reduce-to-empty-set` procedure.

If $\phi(x) \in C$, then we follow the branch that assigns color c to x in one of the executions of line 4. Afterwards, we may update the list assignment. If $A_c^x = \emptyset$, then Claim 1 tells us that there is no vertex in X' left that has color c in its list and that is adjacent to a vertex in Y' with c in its list; if there were such vertices they would have been associates of x. Hence, we may remove c from C and x from X', as is done in line 7. If $A_c^x \neq \emptyset$, then there are two cases to consider.

`Reduce-to-empty-set`$(X', Y', C, \mathcal{L})$

1. **while** $X' \neq \emptyset$
2. determine a maximal vertex $x \in X'$
3. **for** every color $c \in C$ that is in the list of x **do**
4. color x by c and update the list assignment
5. determine the set A_c^x
6. **if** $A_c^x = \emptyset$ **then**
7. `Reduce-to-empty-set`$(X' \setminus \{x\}, Y', C \setminus \{c\}, \mathcal{L})$
8. **else**
9. **for** every $z \in A_c^x$ **do**
10. color z by c and update the list assignment
11. determine the set $Y'' \subseteq Y'$ of vertices that have c in their list
12. **for** every coloring ϕ of Y'' that respects the lists **do**
13. color Y'' according to ϕ and update the list assignment
14. `Reduce-to-empty-set`$(X' \setminus \{x, z\}, Y' \setminus Y'', C' \setminus \{c\}, \mathcal{L})$
15. **end for**
16. remove c from the lists of every vertex in A_c^x
17. `Reduce-to-empty-set`$(X' \setminus \{x\}, Y', C' \setminus \{c\}, \mathcal{L})$
18. **end for**
19. **end if**
20. remove every color in C from the list of x
21. `Reduce-to-empty-set` $(X' \setminus \{x\}, Y', C, \mathcal{L})$
22. **end for**
23. **end while**
24. put the obtained list assignment in $\mathcal{L}$

Pseudocode 2. Reducing the set X' to the empty set.

Case 1. At least one vertex $z \in A_c^x$ has color $\phi(z) = c$.
We will detect this case in one of the execution of line 10. If after updating the list assignment there is still a set Y'' of vertices in Y' left, then we will consider the coloring according to ϕ in one of the executions of line 13. We follow the corresponding branch that colors the vertices of Y'' according to ϕ. Afterwards, we may remove the vertices of Y'' from Y' as is done in line 14. Consequently, the lists of the remaining vertices of Y' do not contain c anymore. Hence, we may remove c from C in line 14. Because x and z received a color, we may remove x and z from X'; this is done in line 14 as well.

Case 2. None of the vertices in A_c^x has color c according to ϕ.
In this case we follow the branch that removes c from the lists of every vertices in A_c^x; see line 16. We claim that c is not in C anymore. This can be seen as follows. In order to obtain a contradiction suppose that $c \in C$. Then there are two adjacent vertices $x^* \in X'$ and $y^* \in Y'$ that each have c in their list. Because x received color c and we removed c from the lists of its neighbors, we find that y^* is no neighbor of x. However, then x^* must be in A_c^x by the definition of this set and Claim 1. This is not possible either, because we removed c from the list of every vertex in A_c^x. We conclude that $c \notin C$. Hence, we may remove c from C in line 17, and as before, we may also remove x from X', which is done in line 17 as well.

Finally, we consider the case in which $\phi(x) \notin C$. In this case, we follow the branch that removes every color in C from the list of x; see line 20. Afterwards, we may remove x from X', as is done in line 21. We conclude that for every maximal vertex x, there exists a branch that assigns color $\phi(x)$ to x and that the adjustments in the sets X', Y' and C in lines 7, 14, 17 and 21 are permitted. Following these branches leads to a k-list assignment $L' \in \mathcal{L}$ that is respected by ϕ, as desired. This completes our proof of condition 1 of property (ii).

We are left to prove condition 2 of property (ii), namely that our algorithm `Separator` runs in polynomial time and that $|\mathcal{L}| = O(h(k)n^{g(k)})$ for some functions $h(k)$ and $g(k)$ that only depend on k. We note that the sets X', Y' and C can be computed in polynomial time. By the construction of the `Reduce-to-empty` procedure, each k-list assignment in $\mathcal{L}$ is the output of exactly one leaf of the search tree T. This means that the number of leaves of T is an upper bound for the number of the k-list assignments of $\mathcal{L}$. Also, finding a maximal vertex, assigning it a color and updating its list and the lists of its neighbors takes polynomial time. Hence our algorithm runs in polynomial-time if the number of leaves in T is $O(h(k)n^{g(k)})$ for some functions $h(k)$ and $g(k)$ that only depend on k. We will show this latter statement below.

Let ℓ be a leaf of T. Then there exists a sequence of vertices of X', on which we branched in order to arrive at ℓ. Each of these vertices was a maximal vertex at the moment it was considered. We call these vertices the *ℓ-vertices*.

The procedure `Reduce-to-Empty` only assigns a color from C to a vertex in X' if it can remove this color from C afterwards. Maintaining this property has the following two consequences. First, the number of ℓ-vertices that received a

color from C is at most $|C|$; all other ℓ-vertices got their list reduced by removing the colors of C. Second, no two ℓ-vertices received the same color from C. Recall that every vertex in every nonempty set A_c^x determined in line 5 is an associate of the minimal vertex x under consideration. Then, by definition, every set Y'' determined in line 11 has size at most $r-1$.

For a leaf ℓ of T, we let C_ℓ denote the set of colors from C used on the ℓ-vertices. Using the above observations, we can determine a bound on the number of leaves of T as follows.

1. We fix a set $C_\ell \subseteq C$. There are at most $2^{|C|} \leq 2^k$ such sets.
2. We fix a set X'_ℓ of $|C_\ell|$ vertices in X', which correspond to the ℓ-vertices. There are at most $|X'|^{|C_\ell|} \leq n^{|C_\ell|} \leq n^{|C|} \leq n^k$ such sets.
3. We fix the order in which we assign the colors of C_ℓ to the vertices of X'_ℓ during the branching. There are at most $|C_\ell|! \leq |C|! \leq k!$ such orderings.
4. For each $x_i \in X'_\ell$ we fix an associate z_i from X'. We allow that $z_i = x_i$ in order to take into account that x_i might be adjacent to all vertices of Y', or that none of its associates get the same color as x_i. This leads to at most $|X'|^{|X'_\ell|} \leq n^{|X'_\ell|} = n^{|C_\ell|} \leq n^k$ sets of associates.
5. For each $x_i \in X'_\ell$ we choose a set Y''_i of at most $r-1$ vertices from Y'. We color every vertex of Y''_i with a color from its list. Because there are at most $(r-1)|Y'|^{r-1} \leq rn^r$ choices for each Y''_i, this leads to at most $(rn^r)^{|X'_\ell|} = (rn^r)^{|C_\ell|} \leq (rn^r)^{|C|} \leq (rn^r)^k = r^k n^{rk}$ collections of such sets, and each such set can be colored in at most k^r ways.

From the above, we find that the number of leaves, and consequently the number of k-list assignments of $\mathcal{L}$ is at most $2^k \cdot n^k \cdot k! \cdot n^k \cdot r^k n^{rk} \cdot k^r$. Hence, we can set $h(k) = 2^k k! r^k k^r$ and $g(k) = 2k + rk$. This completes the proof of Lemma 3. □

The proof of Lemma 3 differs from the proof of Hoàng et al. [11] in the following way. They define $C = L(Y')$ and show that X' contains a dominating vertex; this suffices for the case $H = P_5$ but does not work for the case $H = rP_1 + P_5$ with $r \geq 1$.

Due to Lemmas 2 and 3, the graph $rP_1 + P_5$ is a dominator-separator for every fixed integer r. Hence we can apply Theorem 3 and obtain the main result of this section.

Theorem 5. *For any fixed integers k and r, the* List k-Coloring *problem can be solved in polynomial time for $(rP_1 + P_5)$-free graphs.*

4 Tightness

In this section we show that Theorem 5 is best possible in the sense that List k-Coloring becomes NP-complete for some integer k on H-free graphs, whenever H is a supergraph of $P_1 + P_5$ with at least 5 edges. In order to prove this we need the following two results. The first result is due to Broersma et al. [4].

Theorem 6 ([4]). *The* 5-Precoloring Extension *problem is* NP*-complete for P_6-free graphs.*

Theorem 7. *The* List 5-Coloring *problem is* NP*-complete for $(P_2 + P_4)$-free graphs.*

Proof. Because we can check in polynomial time whether a coloring of a graph is a coloring that respects a given list assignment, List 5-Coloring is in NP. In order to prove NP-completeness we reduce from the Not-All-Equal 3-Satisfiability (NAE 3SATPL) problem with positive literals only. This NP-complete problem [8], also known as Hypergraph 2-Colorability and Set Splitting, is defined as follows. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of logical variables, and a set $C = \{C_1, C_2, \ldots, C_m\}$ of three-literal clauses over X in which all literals are positive, does there exist a truth assignment for X such that each clause contains at least one true literal and at least one false literal?

From an arbitrary instance I of NAE 3SATPL we define a graph G with a 5-list assignment L. In Claim 1 we show that G is $(P_2 + P_4)$-free. In Claim 2 we show that G has a coloring respecting L if I has a satisfying truth assignment in which each clause contains at least one true literal and at least one false literal. In Claim 3 we prove the converse. Together these three claims form the proof of Theorem 7.

Construction of G.
Let $\{x_1, x_2, \ldots, x_n\}$ and clauses $\{C_1, C_2, \ldots, C_m\}$ be the variables and clauses of I. Then we define G as follows.

- We let each clause C_j correspond to a vertex C_j with $L(C_j) = \{1, 2, 3\}$. We say that such a vertex is of *C-type*.
- We let each variable x_i correspond to a vertex x_i with $L(x_i) = \{4, 5\}$. We say that such a vertex is of *x-type*.
- For each clause C_j, we fix an arbitrary order of its variables x_i, x_k, and x_r, and we introduce three pairs of new vertices $\{a_{i,j}, b_{i,j}\}$, $\{a_{k,j}, b_{k,j}\}$, $\{a_{r,j}, b_{r,j}\}$. We set as lists of admissible colors for these three pairs, respectively: $\{\{1, 4\}, \{2, 5\}\}$, $\{\{2, 4\}, \{3, 5\}\}$, $\{\{3, 4\}, \{1, 5\}\}$. We say that these vertices are *auxiliary*.
- We add edges between x-type and auxiliary vertices whenever the first index of the auxiliary vertex is the same as of the x-type vertex.
- We add edges between C-type and auxiliary vertices whenever the second index of the auxiliary vertex is the same as the index of the C-type vertex.
- We add edges between all C-type vertices and all x-type vertices.

Note that the subgraph of G induced by the C-type and x-type vertices is a complete bipartite graph with nm edges. Furthermore, each clause with its three variables is represented by three 4-cycles that have one C-type vertex in common. This is illustrated in Fig. 1 where we omitted the edges between C_j and its x-type vertices.

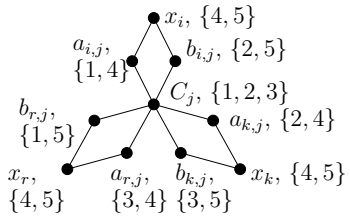

Fig. 1. Representation of a clause and its three variables in G.

Claim 1. The graph G is $(P_2 + P_4)$-free.

We prove Claim 1 as follows. Consider an arbitrary edge uv of G. Let H denote the graph obtained from G after removing of u, v and all the vertices adjacent with them. To prove the claim, it is sufficient to show that H is P_4-free. Observe that the set $\{u, v\}$ contains at least an x-type vertex or a C-type vertex. In the first case H has no C-type vertices. Because the graph obtained from G after removing all C-type vertices is a disjoint union of stars, H is P_4-free. In the second case H has no x-type vertices, and we can use the same arguments. This completes the proof of Claim 1.

Claim 2. If I has a truth assignment in which each clause contains at least one true and at least one false literal, then G has a coloring that respects L.

We prove Claim 2 as follows. Suppose that I has a satisfying truth assignment in which each clause contains at least one true and at least one false literal. We use color 4 for the x-type vertices representing the true literals and color 5 for all the x-type vertices representing the false literals. We give the auxiliary vertices color 4 or 5 if their color is not forced to be from $\{1, 2, 3\}$. Consider a clause C_j with variables x_i, x_k, x_r. By our assumption, the vertices in $\{x_i, x_k, x_r\}$ get colors $\{4, 4, 5\}$, $\{4, 5, 4\}$,$\{5, 4, 4\}$, $\{5, 5, 4\}$, $\{5, 4, 5\}$, or $\{4, 5, 5\}$. Hence, we can color C_j with color $3, 2, 1, 1, 3, 2$, respectively. This completes the proof of Claim 2.

Claim 3. If G has a coloring that respects L, then I has a satisfying truth assignment in which each clause contains at least one true and at least one false literal.

We prove Claim 3 as follows. Suppose that G has a coloring that respects L. Then each of the x-type vertices has color 4 or 5, and each of the C-type vertices has color 1, 2 or 3. We define a truth assignment that sets a variable to `true` if the corresponding x-type vertex has color 4, and to `false` otherwise. Suppose that I contains a clause C_j with literals x_i, x_k, x_r that are all set to true. Then x_i, x_k, x_r all have color 4. Consequently, $a_{i,j}$, $a_{k,j}$, $a_{r,j}$ must have color $1, 2, 3$, respectively. However, this is not possible, because C_j has a color from $\{1, 2, 3\}$. Hence, every clause contains at least one false literal. In the same way we can show that every clause contains at least one true literal. This completes the proof of Claim 3, and consequently, the proof of Theorem 7. □

As explained in Section 1, the 5-Coloring problem is NP-complete for H-free graphs whenever H is not a linear forest, due to results of Kamiński and Lozin [13] and Leven and Galil [18]. Consequently, List 5-Coloring is NP-complete for such graph classes. This means that P_6 and P_2+P_5 are the two remaining supergraphs of P_1+P_5 with exactly 5 edges. By Theorems 6 and 7, List 5-Coloring is NP-complete for P_6-free graphs and for (P_2+P_5)-free graphs, respectively. This yields our desired result.

Theorem 8. *Let H be a supergraph of P_1+P_5 with at least 5 edges. Then* List 5-Coloring *is* NP*-complete for H-free graphs.*

5 Future Work

Theorem 5 implies that for any fixed integer k and any fixed graph H on at most 5 vertices, the List k-Coloring problem is polynomial-time solvable, except when $H=P_2+P_3$.

Is List k-Coloring polynomial-time solvable on (P_2+P_3)-free graphs for any fixed k?

Due to the aforementioned polynomial-time result on List 3-Coloring for sP_3-free graphs [4], the first open case is $k=4$. We note that the same question is also open with respect to k-Coloring. For this problem, the first open case is $k=5$, as it is known that 4-Coloring is polynomial-time solvable on (P_2+P_3)-free graphs [9]. A possible solution strategy would be to prove that P_2+P_3 is a dominator-separator graph but this seems to be difficult.

References

1. G. Bacsó and Zs. Tuza, Dominating cliques in P_5-free graphs, Periodica Mathematica Hungarica 21, 303–308 (1990).
2. J.A. Bondy and U.S.R. Murty, Graph Theory, Springer Graduate Texts in Mathematics 244 (2008).
3. H.J. Broersma, F.V. Fomin, P.A. Golovach and D. Paulusma, Three complexity results on coloring P_k-free graphs, Proceedings of IWOCA 2009, LNCS 5874, 95–104 (2009).
4. H.J. Broersma, P.A. Golovach, D. Paulusma and J. Song, Updating the complexity status of coloring graphs without a fixed induced linear forest, manuscript.
5. H.J. Broersma, P.A. Golovach, D. Paulusma and J. Song, Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time, manuscript.
6. D. Bruce, C.T. Hoàng, and J. Sawada, A certifying algorithm for 3-colorability of P_5-free graphs, Proceedings of ISAAC 2009, LNCS 5878, 594-604 (2009).
7. K. Dabrowski, V. Lozin, R. Raman and B. Ries, Colouring vertices of triangle-free graphs, Proceedings of WG 2010, LNCS 6410, 184-195 (2010).
8. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco (1979).
9. P.A. Golovach, D. Paulusma and J. Song, 4-Coloring H-free graphs when H is small, manuscript.

10. M. Grötschel, L. Lovász, and A. Schrijver, Polynomial algorithms for perfect graphs, Ann. Discrete Math., Topics on Perfect Graphs 21, 325–356 (1984).
11. C.T. Hoàng, M. Kamiński, V. Lozin, J. Sawada, and X. Shu, Deciding k-colorability of P_5-free graphs in polynomial time, Algorithmica 57, 74–81 (2010).
12. I. Holyer, The NP-completeness of edge-coloring, SIAM J. Comput. 10, 718–720 (1981).
13. M. Kamiński and V.V. Lozin, Coloring edges and vertices of graphs without short or long cycles, Contributions to Discrete Math. 2, 61–66 (2007).
14. M. Kamiński and V.V. Lozin, Vertex 3-colorability of Claw-free Graphs. Algorithmic Operations Research 21, (2007).
15. D. Král', J. Kratochvíl, Zs. Tuza, and G.J. Woeginger, Complexity of coloring graphs without forbidden induced subgraphs, Proceedings of WG 2001, LNCS 2204, 254–262 (2001).
16. J. Kratochvíl, Precoloring extension with fixed color bound, Acta Math. Univ. Comen. 62, 139–153 (1993).
17. V.B. Le, B. Randerath and I. Schiermeyer, On the complexity of 4-coloring graphs without long induced paths, Theoret. Comput. Sci. 389, 330–335 (2007).
18. D. Leven and Z. Galil, NP completeness of finding the chromatic index of regular graphs, Journal of Algorithms 4, 35—44 (1983).
19. F. Maffray and M. Preissmann, On the NP-completeness of the k-colorability problem for triangle-free graphs, Discrete Math. 162, 313–317 (1996).
20. B. Randerath and I. Schiermeyer, 3-Colorability $\in$ P for P_6-free graphs, Discrete Appl. Math. 136, 299–313 (2004).
21. B. Randerath and I. Schiermeyer, Vertex colouring and forbidden subgraphs - a survey, Graphs Combin. 20, 1–40 (2004).
22. D. Schindl, Some new hereditary classes where graph coloring remains NP-hard, Discrete Math. 295, 197–202 (2005).
23. Zs. Tuza, Graph colorings with local restrictions - a survey, Discuss. Math. Graph Theory 17, 161–228 (1997).
24. G.J. Woeginger and J. Sgall, The complexity of coloring graphs without long induced paths, Acta Cybernet. 15, 107–117 (2001).

A The proof of Theorem 3

Theorem 3. *Let H be a dominator-separator graph, and let k be a fixed integer. Then* List k-Coloring *can be solved in polynomial time for H-free graphs.*

Proof. Suppose that H is a dominator-separator and that k is some fixed integer. Let G be an H-free graph with k-list assignment L. Let $|V_G| = n$. If G is not connected, we apply the algorithm on each connected component of G. Hence, we may assume that G is connected.

First we check if G has a dominating set of at most $f(k)$ vertices. Recall that f is a function that only depends on k and that is given to us, because G satisfies property (i). Property (i) also tells us that G is not k-colorable if we do not find such a dominating set. A brute force search for this dominating set takes $O(n^{f(k)})$ time, which is polynomial because k is fixed.

Suppose that we find a dominating set D with $|D| \leq f(k)$. Then we fix an order $d_1, \ldots, d_{|D|}$ of the vertices of D and compute the corresponding fixed sets F_i for $i = 1, \ldots, |D|$ in polynomial time. Let G_i denote the subgraph of G induced by F_i for $i = 1, \ldots, |D|$. By definition, every vertex of each F_i is adjacent to d_i for $i = 1, \ldots, |D|$. Consequently, each G_i must have a coloring using at most $k-1$ colors should G have a coloring respecting L. For $i = 1, \ldots, |D|$, we define a $(k-1)$-list assignment L_i by assigning the list $L_i(u) = \{1, \ldots, k-1\}$ to every $u \in F_i$. Then we apply the algorithm on every G_i with list assignment L_i in order to determine if G_i allows a $(k-1)$-coloring. If there is a subgraph G_i that has no $(k-1)$-coloring, then G has no coloring respecting L. Otherwise, we have found a $(k-1)$-coloring ϕ_i for every G_i. The color classes of each ϕ_i form a partition $\mathcal{X}_i$ of F_i in at most $k-1$ independent sets. Obtaining this partition was exactly the purpose of constructing these colorings. Afterwards they does not play a role anymore.

We now color the vertices of D in every possible way, while respecting L. In other words, the new lists of the vertices in D have size 1 in every such coloring of D. If $u \in D$ got colored by color i, then we remove i from the list of every neighbor of u that is not in D. This gives us a set $\mathcal{L}_D$ of k-list assignments of G that is compatible with L and that has cardinality $|\mathcal{L}_D| \leq k^{|D|} \leq k^{f(k)}$; the latter is a constant because k is fixed.

We consider each k-list assignment $L' \in \mathcal{L}_D$. We apply property (ii) on two independent sets $X \in \mathcal{X}_i$ and $Y \in \mathcal{X}_j$ for some $1 \leq i < j \leq |D|$. This yields a set of k-list assignments that is compatible with L', and for which X and Y are separated. Property (ii) also guarantees that the size of this set is at most $O(h(k)n^{g(k)})$ for some functions $h(k)$ and $g(k)$ that only depend on k. Starting with each newly created list assignment from this set, we repeatedly apply property (ii) until all other pairs of independent sets that consist of one set from $\mathcal{X}_i$ and one from $\mathcal{X}_j$ are separated as well. Note that previously separated sets will indeed remain separated, because the lists of every newly generated list assignment are subsets of the lists of an earlier created list assignment. The resulting set of k-list assignments $\mathcal{L}^*$ is compatible with L'. Because the number of pairs X, Y with $X \in \mathcal{X}_i$ and $Y \in \mathcal{X}_j$ is $|\mathcal{X}_i| \cdot |\mathcal{X}_j| \leq (k-1) \cdot (k-1) \leq k^2$,

the set $\mathcal{L}^*$ is obtained in polynomial time and contains $O((h(k)n^{g(k)})^{k^2})$ list assignments. We find that F_i and F_j are separated for each list assignment $L^* \in \mathcal{L}^*$, because every pair (X, Y) with $X \in \mathcal{X}_i$ and $Y \in \mathcal{X}_j$ is separated for L^*, and $\mathcal{X}_i$ and $\mathcal{X}_j$ are partitions of F_i and F_j, respectively.

Starting with each list assignment of $\mathcal{L}^*$, we repeatedly apply the above procedure until all other pairs of fixed sets are separated as well. Because there are $\frac{1}{2}|D| \cdot (|D|-1) \leq \frac{1}{2}f(k) \cdot (f(k)-1) \leq f(k)^2$ such pairs, the total time that we use is polynomial and the total number of k-list assignments that we create in this way form a set $\mathcal{L}'$ that is compatible with the list assignment $L' \in \mathcal{L}_D$ and that has size $O(((h(k)n^{g(k)})^{k^2})^{f(k)^2}) = O(h'(k)n^{g'(k)})$ where $h'(k) = h(k)^{k^2 f(k)^2}$ and $g'(k) = g(k)^{k^2 f(k)^2}$ are functions that only depend on k.

Recall that $|\mathcal{L}_D| \leq k^{f(k)}$. After processing all list assignments of $\mathcal{L}_D$ as described above, we obtain a set $\mathcal{L}$ that is compatible with L and that has size $O(k^{f(k)}h'(k)n^{g'(k)})$; this number is polynomial in n because k is fixed.

We consider each k-list assignment $L'' \in \mathcal{L}$. For $i = 1, \ldots, |D|$, let L''_i denote the restriction of L'' to F_i. By construction, every two fixed sets are separated for L''. Recall that every vertex in D already received a color. Then, G has a coloring respecting L'' if and only if every G_i has a coloring respecting L''_i. Hence, we can apply the algorithm on each G_i. Because we removed the color of every vertex of D from the lists of its neighbors and D is dominating, we find that $L''_i(F_i)$ contains at most $k-1$ colors for $i = 1, \ldots, |D|$, i.e., the number of colors decreases. This means that the algorithm runs in polynomial time. This completes the proof of Theorem 3. □