List 3-Coloring on Comb-Convex and Caterpillar-Convex Bipartite Graphs

Banu Baklan Şen[1][0000−0003−4545−5044], Öznur Yaşar Diner[1][0000−0002−9271−2691], and Thomas Erlebach[2][0000−0002−4470−5868]

[1] Computer Engineering Department, Kadir Has University, Istanbul, Turkey
banubak@hotmail.com, oznur.yasar@khas.edu.tr
[2] Department of Computer Science, Durham University, Durham, United Kingdom
thomas.erlebach@durham.ac.uk

Abstract. Given a graph $G = (V, E)$ and a list of available colors $L(v)$ for each vertex $v \in V$, where $L(v) \subseteq \{1, 2, \ldots, k\}$, List k-Coloring refers to the problem of assigning colors to the vertices of G so that each vertex receives a color from its own list and no two neighboring vertices receive the same color. The decision version of the problem List 3-Coloring is NP-complete even for bipartite graphs, and its complexity on comb-convex bipartite graphs has been an open problem. We give a polynomial-time algorithm to solve List 3-Coloring for caterpillar-convex bipartite graphs, a superclass of comb-convex bipartite graphs. We also give a polynomial-time recognition algorithm for the class of caterpillar-convex bipartite graphs.

1 Introduction

Graph coloring is the problem of assigning colors to the vertices of a given graph in such a way that no two adjacent vertices have the same color. *List coloring* [27, 16] is a generalization of graph coloring in which each vertex must receive a color from its own list of allowed colors. In this paper, we study the list coloring problem with a fixed number of colors in subclasses of bipartite graphs. We give a polynomial-time algorithm for the list 3-coloring problem for caterpillar-convex bipartite graphs, a superclass of comb-convex bipartite graphs. We also give a polynomial-time recognition algorithm for the class of caterpillar-convex bipartite graphs. Our results resolve the open question regarding the complexity of list 3-coloring for comb-convex bipartite graphs stated in [5, 6].

We consider finite simple undirected graphs $G = (V, E)$ with vertex set V and edge set E. By $N_G(v)$ (or by $N(v)$ if the graph is clear from the context) we denote the neighborhood of v in G, i.e., the set of vertices that are adjacent to v. A *k-coloring* of G is a labeling that assigns colors to the vertices of G from the set $[k] = \{1, 2, \ldots, k\}$. A coloring is *proper* if no two adjacent vertices have the same color. A *list assignment* of a graph $G = (V, E)$ is a mapping $\mathcal{L}$ that assigns each vertex $v \in V$ a list $\mathcal{L}(v) \subseteq \{1, 2, \ldots\}$ of admissible colors for v. When $\mathcal{L}(v) \subseteq [k] = \{1, 2, \ldots k\}$ for every $v \in V$ we say that $\mathcal{L}$ is a k-list assignment of G. The total number of available colors is bounded by k in a k-list assignment.

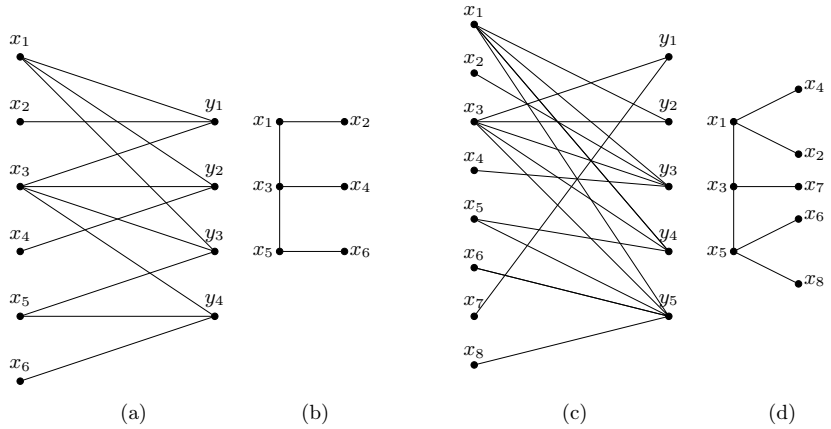

Fig. 1. (a) A comb-convex bipartite graph G_1, (b) a comb representation of G_1, (c) a caterpillar-convex bipartite graph G_2, and (d) a caterpillar representation for G_2.

On the other hand, when the only restriction is that $|\mathcal{L}(v)| \leq k$ for every $v \in V$, then we say that $\mathcal{L}$ is a list k-assignment of G. *List coloring* is the problem of deciding, for a given graph $G = (V, E)$ and list assignment $\mathcal{L}$, whether G has a proper coloring where each vertex v receives a color from its list $\mathcal{L}(v)$. If $\mathcal{L}$ is a k-list assignment for a fixed value of k, the problem becomes the list k-coloring problem:

List k-Coloring (Li k-Col)
Instance: A graph $G = (V, E)$ and a k-list assignment $\mathcal{L}$.
Question: Does G have a proper coloring where each vertex v receives a color from its list $\mathcal{L}(v)$?

If $\mathcal{L}$ is a list k-assignment instead of a k-list assignment, the problem is called k-List Coloring.

The classes of bipartite graphs of interest to us are defined via a convexity condition for the neighborhoods of the vertices on one side of the graph with respect to a tree defined on the vertices of the other side. The following types of trees are relevant here: A *star* is a tree of diameter 2. A *comb* is a tree that consists of a chordless path P, called the *backbone*, with a single leaf neighbor attached to each backbone vertex [10]. A *caterpillar* is a tree that consists of a chordless path P, called the *backbone*, with an arbitrary number (possibly zero) of leaf vertices attached to each vertex on P. Note that if a caterpillar has exactly one leaf vertex attached to each vertex on P, then that caterpillar is a comb.

A bipartite graph $G = (X \cup Y, E)$ is called a *star-convex* (or *comb-convex*, or *caterpillar-convex*) bipartite graph if a star (or comb, or caterpillar) $T = (X, F)$ can be defined on X such that for each vertex $y \in Y$, its neighborhood $N_G(y)$ induces a subtree of T. The star (or comb, or caterpillar) $T = (X, F)$ is then called a *star representation* (or *comb representation*, or *caterpillar representation*) of G.

Figure 1 shows an example of a comb-convex bipartite graph and its comb representation, and a caterpillar-convex bipartite graph and its caterpillar representation. Both the comb (in part (b)) and the caterpillar (in part (d)) have the path $P = x_1x_3x_5$ as backbone.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we give a polynomial-time algorithm for Li 3-col for caterpillar-convex bipartite graphs (and thus also for comb-convex bipartite graphs). In Section 4, we give a polynomial-time recognition algorithm for caterpillar-convex bipartite graphs. In Section 5, we give concluding remarks. Proofs omitted due to space restrictions can be found in the full version [1].

2 Related Work

Deciding whether a graph has a proper coloring with k colors is polynomial-time solvable when $k = 1$ or 2 [26] and NP-complete for $k \geq 3$ [27]. As Li k-col generalizes this problem, it is also NP-complete for $k \geq 3$. When the list coloring problem is restricted to perfect graphs and their subclasses, it is still NP-complete in many cases such as for bipartite graphs [25] and interval graphs [3]. On the other hand, it is polynomially solvable for trees and graphs of bounded treewidth [22]. The problems Li k-col and k-List Coloring are polynomial-time solvable if $k \leq 2$ and NP-complete if $k \geq 3$ [26, 27]. k–list coloring has been shown to be NP-complete for small values of k for complete bipartite graphs and cographs by Jansen and Scheffler [22], as observed in [17]. The 3-List Coloring problem is NP-complete even if each color occurs in the lists of at most three vertices, as shown by Kratochvil and Tuza [24].

We use the following standard notation for specific graphs: P_t denotes a path with t vertices; K_t denotes a clique with t vertices; $K_{\ell,r}$ denotes a complete bipartite subgraph with parts of sizes ℓ and r; $K^1_{1,s}$ denotes the 1-subdivision of $K_{1,s}$ (i.e., every edge $e = \{u, v\}$ of $K_{1,s}$ is replaced by two edges $\{u, w_e\}$ and $\{w_e, v\}$, where w_e is a new vertex); and $sP_1 + P_5$ is the disjoint union of s isolated vertices and a P_5. Li k-col is known to be NP-complete even for $k = 3$ within the class of 3–regular planar bipartite graphs [23]. On the positive side, for fixed $k \geq 3$, Li k-col is polynomially solvable for P_5-free graphs [19]. Li 3-col is polynomial for P_6-free graphs [8] and for P_7-free graphs [4]. Li 3-col is polynomial-time solvable for $(K^1_{1,s}, P_t)$-free graphs for every $s \geq 1$ and $t \geq 1$ [11]. Li k-col is polynomial-time solvable for $(sP_1 + P_5)$-free graphs, which was proven for $s = 0$ by Hoàng et al. [19] and for every $s \geq 1$ by Couturier et al. [12].

An overview of complexity results for Li k-col in some subclasses of bipartite graphs is shown in Fig. 2. The computational complexity of Li 3-col for chordal bipartite graphs has been stated as an open problem in 2015 [21] and has been of interest since then [13]. In [13] a partial answer is given to this question by showing that Li 3-col is polynomial in the class of biconvex bipartite graphs and convex bipartite graphs. Li 3-col is solvable in polynomial time when it is restricted to graphs with all connected induced subgraphs having a multichain ordering [15]. This result can be applied to permutation graphs and interval

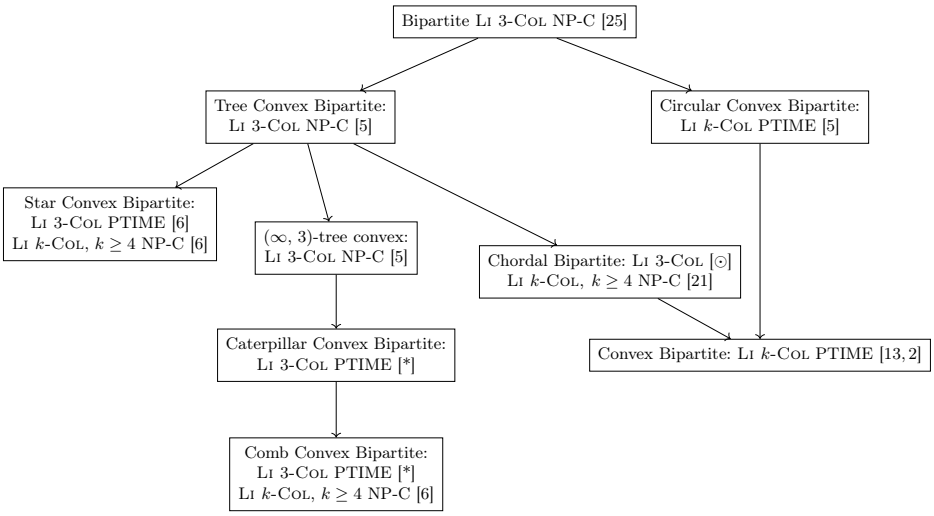

Fig. 2. Computational complexity results for Li k-Col on subclasses of bipartite graphs. [*] refers to this paper, [⊙] refers to an open problem, NP-C denotes NP-complete problem, PTIME denotes polynomial-time solvable problem.

graphs. In [13], it is shown that connected biconvex bipartite graphs have a multichain ordering, implying a polynomial time algorithm for Li 3-col on this graph class. They also provide a dynamic programming algorithm to solve Li 3-col in the class of convex bipartite graphs and show how to modify the algorithm to solve the more general Li H-col problem on convex bipartite graphs. The computational complexity of Li 3-col for P_8-free bipartite graphs is open [3]. Even the restricted case of Li 3-col for P_8-free chordal bipartite graphs is open. Golovach et al. [17] survey results for Li k-col on H-free graphs in terms of the structure of H.

So-called *width parameters* play a crucial role in algorithmic complexity. For various combinatorial problems, it is possible to find a polynomial-time solution by exploiting bounded width parameters such as mim-width, sim-width and clique-width. Given a graph class, it is known that when mim-width is bounded, then Li k-col is polynomial-time solvable [7]. Brettell et al. [7] proved that for every $r \geq 1, s \geq 1$ and $t \geq 1$, the mim-width is bounded and quickly computable for $(K_r, K^1_{1,s}, P_t)$-free graphs. This result further implies that for every $k \geq 1, s \geq 1$ and $t \geq 1$, Li k-col is polynomial-time solvable for $(K^1_{1,s}, P_t)$-free graphs. Most recently, Bonomo-Braberman et al. [5] showed that mim-width is unbounded for star-convex and comb-convex bipartite graphs. On the other hand, Li 3-col is polynomial-time solvable for star-convex bipartite graphs whereas Li k-col is NP-complete for $k \geq 4$ [6]. Furthermore, Bonomo-Braberman et al. [6] show that for comb-convex bipartite graphs, Li k-col remains NP-complete for $k \geq 4$ and leave open the computational complexity of Li 3-col for this graph class. In

this paper, we resolve this problem by showing that Li 3-col is polynomial-time solvable even for caterpillar-convex bipartite graphs.

As for the recognition of graph classes, Bonomo-Braberman et al. [6] provide an algorithm for the recognition of (t, Δ)-tree convex bipartite graphs by using a result from Buchin et al. [9]. Here, a tree is a (t, Δ)-tree if the maximum degree is bounded by Δ and the tree contains at most t vertices of degree at least 3. This result for the recognition of (t, Δ)-tree convex bipartite graphs, however, does not apply to caterpillar-convex bipartite graphs. Therefore, we give a novel algorithm for the recognition of caterpillar-convex bipartite graphs.

3 List 3-Coloring Caterpillar-Convex Bipartite Graphs

In this section we give a polynomial-time algorithm for solving Li 3-col in caterpillar-convex bipartite graphs. Let a caterpillar-convex bipartite graph $G = (X \cup Y, E)$ be given, together with a 3-list assignment $\mathcal{L}$. We assume that a caterpillar $T = (X, F)$ is also given, where $N_G(y)$ induces a subtree of T for each $y \in Y$. If the caterpillar is not provided as part of the input, we can compute one in polynomial time using the recognition algorithm that we present in Section 4.

Let T consist of a backbone B with vertices $b_1, b_2, \ldots, b_n$ (in that order) and a set of leaves $L(b_i)$, possibly empty, attached to each $b_i \in B$. We use L to denote the set of all leaves, i.e., $L = \bigcup_{i=1}^{n} L(b_i)$. Furthermore, for any $1 \leq i \leq j \leq n$, we let $B_{i,j} = \{b_i, b_{i+1}, \ldots, b_j\}$ and $L_{i,j} = \bigcup_{k=i}^{j} L(b_k)$.

The idea of the algorithm is to define suitable subproblems that can be solved in polynomial time, and to obtain the overall coloring as a combination of solutions to subproblems. Roughly speaking, the subproblems consider stretches of the backbone in which all backbone vertices are assumed to be assigned the same color in a proper list 3-coloring. More precisely, a subproblem $SP(i, j, c_1, c_2, c_3)$ is specified via two values i, j with $1 \leq i \leq j \leq n$ and three colors c_1, c_2, c_3 with $c_1 \neq c_2$ and $c_2 \neq c_3$ where $c_i \in [3]$, for $i = 1, 2, 3$. Hence, there are $O(n^2)$ subproblems.

The subproblem $S = SP(i, j, c_1, c_2, c_3)$ is concerned with the subgraph G_S of G induced by $B_{i-1,j+1} \cup L_{i,j} \cup \{y \in Y \mid N(y) \cap (B_{i,j} \cup L_{i,j}) \neq \emptyset\}$. It assumes that color c_1 is assigned to b_{i-1}, color c_2 to the backbone vertices from b_i to b_j, and color c_3 to b_{j+1}. See Fig. 3 for an illustration of $SP(i, j, 2, 1, 2)$. Solving the subproblem S means determining whether this coloring of $B_{i-1,j+1}$ can be extended to a proper list 3-coloring of G_S. The result of the subproblem is False if this is not possible, or True (and a suitable proper list 3-coloring of G_S) otherwise. If $c_1 \notin \mathcal{L}(b_{i-1})$, or $c_3 \notin \mathcal{L}(b_{j+1})$, or $c_2 \notin \mathcal{L}(b_k)$ for some $i \leq k \leq j$, then the result of the subproblem is trivially False.

We will show that this subproblem can be solved in polynomial time as it can be reduced to the 2-list coloring problem, which is known to be solvable in linear time [14, 18]. Furthermore, solutions to consecutive 'compatible' subproblems can be combined, and a proper list 3-coloring of G exists if and only if there is a collection of subproblems whose solutions can be combined into a list 3-coloring of G. For example, the colorings of two subproblems $SP(i, j, c_1, c_2, c_3)$

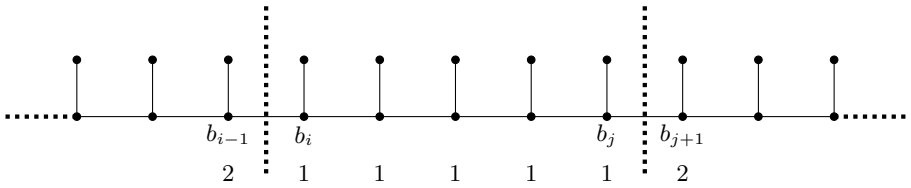

Fig. 3. Illustration of subproblem $SP(i, j, c_1, c_2, c_3)$ for the case $SP(i, j, 2, 1, 2)$.

and $SP(j + 1, k, c_2, c_3, c_4)$ can be combined because they agree on the colors of backbone vertices that are in both subproblems, they do not share any leaf vertices, and the vertices $y \in Y$ that have neighbors in both $B_{i,j} \cup L_{i,j}$ and $B_{j+1,k} \cup L_{j+1,k}$ must be adjacent to b_j and b_{j+1}, which are colored with colors c_2 and c_3 (where $c_2 \neq c_3$) in the colorings of both subproblems, and hence must have received the same color (the only color in $\{1, 2, 3\} \setminus \{c_2, c_3\}$) in both colorings. To check whether there is a collection of compatible subproblems whose solutions can be combined into a list 3-coloring of G, we will show that it suffices to search for a directed path between two vertices in an auxiliary directed acyclic graph (DAG) on the subproblems whose result is True.

For a subproblem $S = SP(i, j, c_1, c_2, c_3)$, if $i = 1$, there is no vertex b_{i-1}, and we write $*$ for c_1; similarly, if $j = n$, there is no vertex b_{j+1}, and we write $*$ for c_3. The graph G_S considered when solving such a subproblem does not contain b_{i-1} or b_{j+1}, respectively, but is otherwise defined analogously. If $i = 1$ and $j = n$, then G_S contains neither b_{i-1} nor b_{j+1}.

Lemma 1. *There is a linear-time algorithm for solving any subproblem of the form $SP(i, j, c_1, c_2, c_3)$.*

Proof. Consider the subproblem $S = SP(i, j, c_1, c_2, c_3)$. Let G_S be the subgraph of G defined by S, and let $X_S \subseteq X$, $Y_S \subseteq Y$ be such that the vertex set of G_S is $X_S \cup Y_S$. First, we check whether $c_1 \in \mathcal{L}(b_{i-1})$ (only if $i > 1$), $c_3 \in \mathcal{L}(b_{j+1})$ (only if $j < n$), and $c_2 \in \mathcal{L}(b_k)$ for all $i \leq k \leq j$. If one of these checks fails, we return False. Otherwise, we assign color c_1 to b_{i-1}, color c_2 to all vertices in $B_{i,j}$, and color c_3 to b_{j+1}.

For every vertex $y \in Y_S$, we check if $N(y)$ contains any vertices of $B_{i-1,j+1}$ and, if so, remove the colors of those vertices from $\mathcal{L}(y)$ (if they were contained in $\mathcal{L}(y)$). If the list of any vertex $y \in Y_S$ becomes empty in this process, we return False.

Let B_S denote the backbone vertices in X_S and L_S the leaf vertices in X_S (with respect to the caterpillar T). If there is a vertex in L_S or Y_S with a list of size 1, assign the color in that list to that vertex and remove that color from the lists of its neighbors (if it is contained in their lists). Repeat this operation until there is no uncolored vertex with a list of size 1. (If an uncolored vertex with a list of size 1 is created later on in the algorithm, the same operation is applied to that vertex.) If the list of any vertex becomes empty in this process, return

False. Otherwise, we must arrive at a state where all uncolored vertices in G_S have lists of size 2 or 3.

If there is a vertex $y \in Y_S$ with a list of size 3, that vertex must be adjacent to a single leaf ℓ in L_S (as it cannot be adjacent to a backbone vertex). In this case we remove an arbitrary color from $\mathcal{L}(y)$: This is admissible as, no matter what color ℓ receives in a coloring, vertex y can always be colored with one of the two colors that have remained in its list.

If there is a vertex $\ell \in L_S$ with a list of length 3, assign color c_2 to ℓ (and remove color c_2 from the lists of vertices in $N(\ell)$). This color assignment does not affect the existence of a proper list 3-coloring for the following reasons (where we let b_k denote the backbone vertex with $\ell \in L(b_k)$):

- If a vertex $y \in N(\ell)$ is adjacent to more than one vertex, it must be adjacent to b_k, which has been colored with c_2, and hence it cannot receive color c_2 in any case.
- If a vertex $y \in N(\ell)$ is adjacent only to ℓ and no other vertex, then y can still be colored after ℓ is assigned color c_2, because we cannot have $\mathcal{L}(y) = \{c_2\}$; this is because, if y had the list $\mathcal{L}(y) = \{c_2\}$, it would have been colored c_2 and the color c_2 would have been removed from $\mathcal{L}(\ell)$.

If at any step of this process, an uncolored vertex with an empty list is created, return False. Otherwise, we arrive at an instance I of Li 3-col where all uncolored vertices have lists of size 2. Such an instance can be solved in linear time [14, 18] (via reduction to a 2SAT problem). If I admits a proper list 3-coloring, that coloring gives a proper list 3-coloring of G_S, and we return True and that coloring. Otherwise, we return False.

Correctness of the algorithm follows from its description, and the algorithm can be implemented to run in linear time using standard techniques. □

Call a subproblem $S = SP(i, j, c_1, c_2, c_3)$ *valid* if its answer is True (and a proper list 3-coloring of G_S has been produced), and *invalid* otherwise. To check whether the colorings obtained from valid subproblems can be combined into a list 3-coloring of G, we make use of an auxiliary DAG H constructed as follows. The existence of a proper list 3-coloring of G can then be determined by checking whether H contains a directed path from s to t.

Definition 1. *The auxiliary DAG $H = (V_H, A)$ has vertices s, t, and a vertex for each valid subproblem $SP(i, j, c_1, c_2, c_3)$. Its arc set A contains the following arcs: An arc $(s, SP(1, i, *, c_2, c_3))$ for each $i < n$ and $c_2, c_3 \in [3]$ such that $SP(1, i, *, c_2, c_3)$ is valid; an arc $(SP(i, n, c_1, c_2, *), t)$ for each $i > 1$ and $c_1, c_2 \in [3]$ such that $SP(i, n, c_1, c_2, *)$ is valid; arcs $(s, SP(1, n, *, c_2, *))$ and $(SP(1, n, *, c_2, *), t)$ if $SP(1, n, *, c_2, *)$ is valid; an arc $(SP(i, j, c_1, c_2, c_3), SP(j+1, k, c_2, c_3, c_4)$ for each $i \leq j \leq k-1$ and each $c_1, c_2, c_3, c_4 \in [3]$ (or $c_1 = *$ or $c_4 = *$ if $i = 1$ or $k = n$, respectively) such that $SP(i, j, c_1, c_2, c_3)$ and $SP(j+1, k, c_2, c_3, c_4)$ are both valid.*

Theorem 1. Li 3-col *can be solved in polynomial time for caterpillar-convex bipartite graphs.*

Algorithm 1 List-3-Coloring Algorithm for Caterpillar-Convex Bipartite Graphs

Require: A caterpillar-convex bipartite graph $G = (X \cup Y, E)$ (with caterpillar $T = (X, F)$) and a list assignment $\mathcal{L}$.
Ensure: A proper coloring that obeys $\mathcal{L}$, or False if no such coloring exists.
▷ Compute solutions to all subproblems
for $i = 1$ to n **do**
  for $j = i$ to n **do**
    for $c_i \in [3], i = 1, 2, 3$ with $c_1 \neq c_2$ and $c_2 \neq c_3$ **do**
      ▷ let $c_1 = *$ if $i = 1$ and $c_3 = *$ if $j = n$
      Solve $SP(i, j, c_1, c_2, c_3)$ (Lemma 1)
    end for
  end for
end for
▷ Check if solutions of subproblems can be combined into a list 3-coloring of G
Build a DAG H whose vertices are s, t, and a vertex $SP(i, j, c_1, c_2, c_3)$ for each subproblem with answer True (Definition 1)
if H contains a directed path P from s to t **then**
  Return the coloring obtained as union of the colorings of the subproblems on P
else
  Return False
end if

The resulting algorithm is shown in Algorithm 1. As comb-convex bipartite graphs are a subclass of caterpillar-convex bipartite graphs, we obtain:

Corollary 1. Li 3-col *can be solved in polynomial time for comb-convex bipartite graphs.*

Combining Corollary 1 with Theorem 4 in [6] and the polynomial-time solvability of Li k-col for $k \leq 2$ [16, 27] yields a complexity dichotomy: Li k-col is polynomial-time solvable on comb-convex bipartite graphs when $k \leq 3$; otherwise, it is NP-complete.

4 Recognition of Caterpillar-Convex Bipartite Graphs

We give a polynomial-time recognition algorithm for caterpillar-convex bipartite graphs. We are given a bipartite graph $G = (X \cup Y, E)$ and want to decide whether it is caterpillar-convex and, if so, construct a caterpillar representation $T = (X, F)$. First, we assume that a specific partition of the vertex set into independent sets X and Y is given as part of the input, and we want to decide whether there is a caterpillar representation $T = (X, F)$ with respect to that given bipartition (i.e., the vertex set of the caterpillar is the independent set X that was specified in the input). At the end of this section, we will discuss how to handle the case that the bipartite graph is given without a specific bipartition

of the vertex set and we want to decide whether the vertex set can be partitioned into independent sets X and Y in such a way that there is a caterpillar representation with respect to that bipartition.

The main idea of the algorithm for recognizing caterpillar-convex bipartite graphs is to construct an auxiliary DAG D on vertex set X in such a way that the sinks in D can be used as the backbone vertices of T. To make this work, it turns out that we first need to remove some vertices from G that have no effect on whether G is caterpillar-convex. First, we show that we can remove isolated vertices from X and vertices of degree 0 or 1 from Y.

Lemma 2. *Let $x \in X$ be a vertex with degree 0, and let G' be the graph obtained from G by removing x. Then G' is caterpillar-convex if and only if G is caterpillar-convex. Furthermore, a caterpillar representation of G can be constructed from a caterpillar representation of G' by adding x in a suitable location.*

Lemma 3. *Let $y \in Y$ be a vertex with degree 0 or 1, and let G' be the graph obtained from G by removing y. Then G' is caterpillar-convex if and only if G is caterpillar-convex. Any caterpillar representation of G' is also a caterpillar representation of G.*

We call a pair of vertices x_i and x_j *twins* if $N(x_i) = N(x_j)$. The twin relation on X partitions X into equivalence classes, such that $x_1, x_2 \in X$ are twins if and only if they are in the same class. We say that two twins x, x' are *special twins* if $\{x, x'\}$ is an equivalence class of the twin relation on X and if there is $y \in Y$ with $N_G(y) = \{x, x'\}$. Now, we show that removing a twin from X (with some additional modification in the case of special twins) has no effect on whether the graph is caterpillar-convex or not.

Lemma 4. *Let $x, x' \in X$ be twins of non-zero degree, and let $G' = (X' \cup Y', E')$ be the graph obtained from G by deleting x. If x, x' are special twins in G, then modify G' by adding a new vertex $\bar{x}$ to X', a new vertex $\bar{y}$ to Y', and the edges $\{x', \bar{y}\}$ and $\{\bar{x}, \bar{y}\}$ to E'. Then G is caterpillar-convex if and only if G' is caterpillar-convex. Furthermore, a caterpillar representation of G can be constructed from a caterpillar representation of G' by adding x in a suitable location (and removing $\bar{x}$ if it has been added to G').*

We remark that the special treatment of special twins in Lemma 4 seems necessary because there is a graph $G = (X \cup Y, E)$ with special twins that does not have a caterpillar representation $T = (X, F)$, while simply removing one of the two special twins (without adding the extra vertices $\bar{x}$ and $\bar{y}$) would produce a graph $G' = (X' \cup Y', E')$ that has a caterpillar representation $T' = (X', F')$. An example of such a graph is the graph with $X = \{a, b, c, f, g, x, x'\}$ where the neighborhoods of the vertices in Y are $\{a, f\}, \{a, b\}, \{b, x, x'\}, \{x, x'\}, \{b, c\}, \{c, g\}$. Here, the vertices x and x' are special twins, and the graph obtained after removing x has the caterpillar representation with backbone path abc and leaf f attached to a, leaf x' attached to b, and leaf g attached to c.

Let $G_1 = (X_1 \cup Y_1, E_1)$ be the graph obtained from $G = (X \cup Y, E)$ by repeatedly removing vertices of degree 0 from X, vertices of degree 0 or 1 from

Y, and twins from X (with the extra modification detailed in Lemma 4 in case of special twins) as long as such vertices exist. Lemmas 2–4 imply:

Corollary 2. *G_1 is caterpillar-convex if and only if G is caterpillar-convex.*

We now define a directed graph $D = (X_1, A)$ based on G_1: For every pair of distinct vertices $x, x' \in X_1$, we let D contain the arc (x, x') if and only if $N_{G_1}(x) \subseteq N_{G_1}(x')$, i.e., we add the arc (x, x') if and only if every vertex in y that is adjacent to x in G_1 is also adjacent to x' in G_1. Note that D is transitive: If it contains two arcs (x, x') and (x', x''), it must also contain (x, x'').

Lemma 5. *D is a directed acyclic graph.*

Proof. Assume there is a cycle on vertices $x_i, x_{i+1}, \ldots, x_j$ in D. Then, $N(x_i) \subseteq N(x_{i+1}) \subseteq \cdots \subseteq N(x_j) \subseteq N(x_i)$. Thus $N(x_i) = N(x_{i+1}) = \cdots = N(x_j)$, and so $x_i, x_{i+1}, \ldots, x_j$ are twins, a contradiction because there are no twins in X_1. Thus D is acyclic. □

The following lemma can be proved along the following lines: If there is a caterpillar representation in which two backbone vertices are connected by an arc in D, then there must be two adjacent backbone vertices u and v with an arc (u, v) in D. The caterpillar can then be modified by attaching u and the vertices in $L(u)$ as leaves to v, giving another valid caterpillar representation with fewer arcs between backbone vertices.

Lemma 6. *If $G_1 = (X_1 \cup Y_1, E_1)$ is caterpillar-convex, there is a caterpillar representation $T_1 = (X_1, F)$ in which no two backbone vertices are connected by an arc in D.*

Lemma 7. *If $G_1 = (X_1 \cup Y_1, E_1)$ is caterpillar-convex, there is a caterpillar representation $T_1 = (X_1, F)$ such that the set of backbone vertices is exactly the set of sinks in D.*

Proof. By Lemma 6, there exists a caterpillar representation T_1 of G_1 in which no two backbone vertices are connected by an arc in D. Furthermore, every leaf attached to a backbone vertex (in T_1) has an arc (in D) to that backbone vertex (because every $y \in Y$ has degree at least 2). A backbone vertex cannot have an arc (in D) to a leaf attached to it (in T_1), as D is acyclic (Lemma 5). Finally, a backbone vertex b cannot have an arc (in D) to a leaf vertex ℓ attached to a different backbone vertex b' because that would imply that b has an arc to b' (since every vertex in y that is adjacent to b is also adjacent to ℓ and hence, as $N(y)$ induces a tree in T_1, also to b'). Therefore, the backbone vertices of T_1 are exactly the sinks (vertices without outgoing edges) of D. □

Theorem 2. *Algorithm 2 decides in polynomial time whether a given bipartite graph $G = (X \cup Y, E)$ is caterpillar-convex and, if so, outputs a caterpillar representation $T = (X, F)$.*

Algorithm 2 Recognition Algorithm for Caterpillar-Convex Bipartite Graphs

Require: A bipartite graph $G = (X \cup Y, E)$

Ensure: Either return a caterpillar representation $T = (X, F)$ of G, or decide that G is not caterpillar-convex and return 'fail'

1: Obtain $G_1 = (X_1 \cup Y_1, E_1)$ from G by removing vertices of degree 0 from X, vertices of degree 0 or 1 from Y, and twins from X (with the extra modification stated in Lemma 4 in case of special twins), as long as any such vertex exists (Lemmas 2–4)
2: Create a directed graph $D = (X_1, A)$ that contains the arc (x, x') if and only if $N_{G_1}(x) \subseteq N_{G_1}(x')$
3: B = the set of sinks in D, L = all other vertices in D
4: Use an algorithm for consecutive ones [20] to order B. If it fails, return 'fail'.
5: Form caterpillar T_1 by taking the ordered backbone B and attaching each vertex in L as leaf to an arbitrary vertex in B to which it has an arc in D
6: Obtain T from T_1 by adding the vertices that were deleted from X in Step 1 (and removing vertices that have been added when special twins were processed) (Lemmas 2 and 4).

Proof. Let $G = (X \cup Y, E)$ be a bipartite graph and $n = |X \cup Y|$. First, the algorithm removes vertices of degree 0 from X, vertices of degree 0 or 1 from Y, and twins from X (with the extra modification of Lemma 4 in case of special twins) as long as such vertices exist. The resulting graph is $G_1 = (X_1 \cup Y_1, E_1)$. By Corollary 2, G_1 is caterpillar-convex if and only if G is caterpillar-convex.

Next, the algorithm constructs the directed graph $D = (X_1, A)$ from $G_1 = (X_1 \cup Y_1, E_1)$ by adding an arc (x_i, x_j) for $x_i, x_j \in X_1$ if $N_{G_1}(x_i) \subseteq N_{G_1}(x_j)$. Once D has been constructed, the set B of sinks and the set L of remaining vertices are determined. Then, we create a set system $\mathcal{S}$ containing for every $y \in Y$ the set $N(y) \cap B$ and apply an algorithm for checking the consecutive ones property [20] to check if B can be ordered in such a way that every set in $\mathcal{S}$ consists of consecutive vertices. If so, the resulting order is used to determine the order in which B forms the backbone path. Otherwise, G_1 (and hence G) cannot be caterpillar-convex (cf. Lemma 7), and the algorithm returns 'fail'.

Next, the algorithm attaches each vertex $\ell \in L$ as a leaf to an arbitrary vertex $b \in B$ to which it has an arc in D. Every $\ell \in L$ must indeed have at least one arc to a vertex in B: As D is acyclic, every vertex ℓ that is not a sink must have a directed path leading to some sink b, and as D is transitive, the arc (ℓ, b) must exist. Attaching ℓ to b yields a valid caterpillar representation for the following reason: As every neighbor y of ℓ is also adjacent to b, and as $N(y) \cap B$ is a contiguous segment of B, it is clear that $N(y)$ induces a tree in the resulting caterpillar T_1. Hence, T_1 is a caterpillar representation of G_1.

Finally, the vertices that have been deleted in the first step are added back (and vertices that have been added when special twins were processed are removed) in order to extend the caterpillar T_1 to a caterpillar representation T of G (Lemmas 2 and 4). By Corollary 2, T is a caterpillar representation of G.

It can be shown that the algorithm can be implemented to run in $O(n^3)$ time (see the full version [1] for details). □

Finally, we discuss the case that the bipartition of the vertex set V of the input graph $G = (V, E)$ into independent sets X and Y is not provided as part of the input. First, if $G = (V, E)$ is a connected bipartite graph, note that there is a unique partition of V into two independent sets Q and R. We can then run the recognition algorithm twice, once with $X = Q$ and $Y = R$ and once with $X = R$ and $Y = Q$. G is caterpillar-convex if and only if at least one of the two runs of the algorithm produces a caterpillar representation. If $G = (V, E)$ is not connected, let $H_1, \ldots, H_r$ for some $r > 1$ be its connected components. As just discussed, we can check in polynomial time whether each connected component H_j, $1 \leq j \leq r$, is a caterpillar-convex bipartite graph. If all r connected components are caterpillar-convex, the whole graph G is caterpillar-convex, and a caterpillar representation can be obtained by concatenating the backbones of the caterpillar representations of the connected components in arbitrary order. If at least one of the connected components, say, the component H_j, is not caterpillar-convex, then G is not caterpillar-convex either. This can be seen as follows: Assume for a contradiction that G is caterpillar-convex while H_j is not caterpillar-convex. Then let $T = (X, F)$ be a caterpillar representation of G. Observe that the subgraph of T induced by $V(H_j) \cap X$, where $V(H_j)$ denotes the vertex set of H_j, must be connected. Therefore, that subgraph of T provides a caterpillar representation of H_j, a contradiction to our assumption. Thus we get:

Corollary 3. *There is a polynomial-time algorithm that decides whether a given bipartite graph $G = (V, E)$ is caterpillar-convex, i.e., whether it admits a bipartition of V into independent sets X and Y such that there is a caterpillar representation $T = (X, F)$.*

5 Conclusion

Determining the computational complexity of Li k-col for $k \geq 3$ when restricted to comb-convex bipartite graphs was stated as an open problem by Bonomo-Braberman et al. [5]. Subsequently, the same authors proved that the problem is NP-complete for $k \geq 4$ [6], but the complexity for $k = 3$ was still left open. In this paper, we resolve this question by showing that Li 3-col is solvable in polynomial time even for caterpillar-convex bipartite graphs, a superclass of comb-convex bipartite graphs.

Recall that if mim-width is bounded for a graph class $\mathcal{G}$, then Li k-col is polynomially solvable when it is restricted to $\mathcal{G}$. Polynomial-time solvability of Li k-col on circular convex graphs is shown by demonstrating that mim-width is bounded for this graph class [5]. On the other hand, there are graph classes for which Li 3-col is tractable but mim-width is unbounded, such as star-convex bipartite graphs [6]. By combining our result with Theorem 3 in [5], we conclude that caterpillar-convex bipartite graphs and comb-convex bipartite graphs also belong to this type of graph classes. On a much larger graph class, chordal bipartite graphs, the computational complexity of Li 3-col is still open [21].

Finally, as for future work, it would be interesting to see whether one can modify and extend Algorithm 2 to recognize comb-convex bipartite graphs.

References

1. Baklan Şen, B., Diner, Ö.Y., Erlebach, T.: List 3-coloring on comb-convex and caterpillar-convex bipartite graphs. CoRR **abs/2305.10108** (2023). https://doi.org/10.48550/arXiv.2305.10108
2. Belmonte, R., Vatshelle, M.: Graph classes with structured neighborhoods and algorithmic applications. Theor. Comput. Sci. **511**, 54–65 (2013). https://doi.org/10.1016/j.tcs.2013.01.011
3. Biró, M., Hujter, M., Tuza, Z.: Precoloring extension. I. Interval graphs. Discret. Math. **100**(1-3), 267–279 (1992). https://doi.org/10.1016/0012-365X(92)90646-W
4. Bonomo, F., Chudnovsky, M., Maceli, P., Schaudt, O., Stein, M., Zhong, M.: Three-coloring and list three-coloring of graphs without induced paths on seven vertices. Comb. **38**(4), 779–801 (2018). https://doi.org/10.1007/s00493-017-3553-8
5. Bonomo-Braberman, F., Brettell, N., Munaro, A., Paulusma, D.: Solving problems on generalized convex graphs via mim-width. In: WADS 2021. Lecture Notes in Computer Science, vol. 12808, pp. 200–214. Springer (2021). https://doi.org/10.1007/978-3-030-83508-8_15
6. Bonomo-Braberman, F., Brettell, N., Munaro, A., Paulusma, D.: Solving problems on generalized convex graphs via mim-width. CoRR **abs/2008.09004** (September 2022), https://arxiv.org/abs/2008.09004
7. Brettell, N., Horsfield, J., Munaro, A., Paulusma, D.: List k-colouring P_t-free graphs: A mim-width perspective. Inf. Process. Lett. **173**, 106168 (2022). https://doi.org/10.1016/j.ipl.2021.106168
8. Broersma, H., Fomin, F.V., Golovach, P.A., Paulusma, D.: Three complexity results on coloring P_k-free graphs. Eur. J. Comb. **34**(3), 609–619 (2013). https://doi.org/10.1016/j.ejc.2011.12.008
9. Buchin, K., van Kreveld, M.J., Meijer, H., Speckmann, B., Verbeek, K.: On planar supports for hypergraphs. J. Graph Algorithms Appl. **15**(4), 533–549 (2011). https://doi.org/10.7155/jgaa.00237
10. Chen, H., Lei, Z., Liu, T., Tang, Z., Wang, C., Xu, K.: Complexity of domination, hamiltonicity and treewidth for tree convex bipartite graphs. J. Comb. Optim. **32**(1), 95–110 (2016). https://doi.org/10.1007/s10878-015-9917-3
11. Chudnovsky, M., Spirkl, S., Zhong, M.: List 3-coloring P_t-free graphs with no induced 1-subdivision of $K_{1,s}$. Discret. Math. **343**(11), 112086 (2020). https://doi.org/10.1016/j.disc.2020.112086
12. Couturier, J.F., Golovach, P.A., Kratsch, D., Paulusma, D.: List coloring in the absence of a linear forest. Algorithmica **71**(1), 21–35 (2015). https://doi.org/10.1007/s00453-013-9777-0
13. Díaz, J., Diner, Ö., Serna, M., Serra, O.: On list k-coloring convex bipartite graphs. Graphs and Combinatorial Optimization: from Theory to Applications **5**, 15–26 (2021)
14. Edwards, K.: The complexity of colouring problems on dense graphs. Theor. Comput. Sci. **43**, 337–343 (1986). https://doi.org/10.1016/0304-3975(86)90184-2
15. Enright, J.A., Stewart, L., Tardos, G.: On list coloring and list homomorphism of permutation and interval graphs. SIAM J. Discret. Math. **28**(4), 1675–1685 (2014). https://doi.org/10.1137/13090465X
16. Erdős, P., Rubin, A.L., Taylor, H.: Choosability in graphs. In: Proceedings of the West Coast Conference on Combinatorics, Graph Theory and Computing. Congressus Numerantium, vol. 26, pp. 125–157 (1979)

17. Golovach, P.A., Paulusma, D.: List coloring in the absence of two subgraphs. Discret. Appl. Math. **166**, 123–130 (2014). https://doi.org/10.1016/j.dam.2013.10.010
18. Gravier, S., Kobler, D., Kubiak, W.: Complexity of list coloring problems with a fixed total number of colors. Discret. Appl. Math. **117**(1-3), 65–79 (2002). https://doi.org/10.1016/S0166-218X(01)00179-2
19. Hoàng, C.T., Kaminski, M., Lozin, V.V., Sawada, J., Shu, X.: Deciding k-colorability of P_5-free graphs in polynomial time. Algorithmica **57**(1), 74–81 (2010). https://doi.org/10.1007/s00453-008-9197-8
20. Hsu, W.L.: A simple test for the consecutive ones property. J. Algorithms **43**(1), 1–16 (2002). https://doi.org/10.1006/jagm.2001.1205
21. Huang, S., Johnson, M., Paulusma, D.: Narrowing the complexity gap for coloring (C_s, P_t)–free graphs. The Computer Journal **58**(11), 3074–3088 (2015)
22. Jansen, K., Scheffler, P.: Generalized coloring for tree-like graphs. Discret. Appl. Math. **75**(2), 135–155 (1997). https://doi.org/10.1016/S0166-218X(96)00085-6
23. Kratochvil, J.: Precoloring extension with fixed color bound. Acta Math. Univ. Comen. **62**, 139–153 (1993)
24. Kratochvíl, J., Tuza, Z.: Algorithmic complexity of list colorings. Discret. Appl. Math. **50**(3), 297–302 (1994). https://doi.org/10.1016/0166-218X(94)90150-3
25. Kubale, M.: Some results concerning the complexity of restricted colorings of graphs. Discret. Appl. Math. **36**(1), 35–46 (1992). https://doi.org/10.1016/0166-218X(92)90202-L
26. Lovász, L.: Coverings and coloring of hypergraphs. Proc. 4th Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Utilitas Math. pp. 3–12 (1973)
27. Vizing, V.: Coloring the vertices of a graph in prescribed colors. Diskret. Analiz **101**(29), 3–10 (1976)