

Algorithmica (2014) 70:92–111
DOI 10.1007/s00453-013-9819-7

Online Coloring of Bipartite Graphs with and without Advice

Maria Paola Bianchi ·
Hans-Joachim Böckenhauer · Juraj Hromkovič ·
Lucia Keller

Received: 15 October 2012 / Accepted: 25 July 2013 / Published online: 14 August 2013
© Springer Science+Business Media New York 2013

Abstract In the online version of the well-known graph coloring problem, the vertices appear one after the other together with the edges to the already known vertices and have to be irrevocably colored immediately after their appearance. We consider this problem on bipartite, i.e., two-colorable graphs. We prove that at least $\lfloor 1.13746 \cdot \log_2(n) - 0.49887 \rfloor$ colors are necessary for any deterministic online algorithm to be able to color any given bipartite graph on n vertices, thus improving on the previously known lower bound of $\lfloor \log_2 n \rfloor + 1$ for sufficiently large n .

Recently, the advice complexity was introduced as a method for a fine-grained analysis of the hardness of online problems. We apply this method to the online coloring problem and prove (almost) tight linear upper and lower bounds on the advice complexity of coloring a bipartite graph online optimally or using 3 colors. Moreover, we prove that $O(\sqrt{n})$ advice bits are sufficient for coloring any bipartite graph on n vertices with at most $\lceil \log_2 n \rceil$ colors.

Keywords Online coloring · Lower bounds · Advice complexity · Bipartite graph

A preliminary version of this paper has been presented at the 18th Annual International Computing and Combinatorics Conference (COCOON 2012).

M.P. Bianchi
Dipartimento di Informatica, Università degli Studi di Milano, Milan, Italy
e-mail: maria.bianchi@unimi.it

H.-J. Böckenhauer (✉) · J. Hromkovič · L. Keller
Department of Computer Science, ETH Zurich, Zurich, Switzerland
e-mail: hjb@inf.ethz.ch

J. Hromkovič
e-mail: juraj.hromkovic@inf.ethz.ch

L. Keller
e-mail: lucia.keller@inf.ethz.ch

1 Introduction

In an online problem, the input is revealed piecewise in consecutive time steps and an irrevocable part of the output has to be produced at each time step. For a detailed introduction and an overview of online problems and algorithms, see, e.g., [4]. One of the most studied online scenarios is the problem of coloring a graph online. Here, the vertices of the graph are revealed one after the other, together with the edges connecting them to the already present vertices. The goal is to assign the minimum number of colors to these vertices in such a way that no two adjacent vertices get the same color. As usual in an online setting, each vertex has to be colored before the next one arrives. The quality of an online algorithm for this problem is usually measured by the so-called *competitive ratio*, i.e., the ratio between the number of colors used by this algorithm and an optimal coloring for the resulting graph as it could be computed by an offline algorithm with unlimited computing power, knowing the whole graph in advance.

It turns out that online coloring is a very hard online problem for which no constant competitive ratio is possible [10]. For an overview of results on the online graph coloring problem, see, e.g., [12, 13]. In particular, some bounds on the chromatic number of the class $\Gamma(k, n)$ of k -colorable graphs on n vertices have been proven: For all k and infinitely many n , there exists a $G \in \Gamma(k, n)$ such that any online coloring algorithm for G needs at least $\Omega(((\log_2 n)/(4k))^{k-1})$ colors [17]. On the other hand, there exists an online algorithm for coloring any graph $G \in \Gamma(k, n)$ with $O(n \cdot \log_2^{(2k-3)} n / \log_2^{(2k-4)} n)$ colors [15], where $\log_2^{(k)}$ is the log-function iterated k times. Even for the very restricted class of bipartite, i.e., two-colorable, graphs, any online algorithm can be forced to use at least $\lfloor \log_2 n \rfloor + 1$ colors for coloring some bipartite graph on n vertices [1]. On the other hand, an online algorithm coloring every bipartite graph with at most $2 \log_2 n$ colors is known [15]. In the first part of this paper, we improve the lower bound for bipartite graphs to $\lfloor 1.13746 \cdot \log_2(n) - 0.49887 \rfloor$.

The main drawback in the competitive analysis of online algorithms is that an online algorithm has a huge disadvantage compared to an offline algorithm by not knowing the future parts of the input. This seems to be a rather unfair comparison since there is no way to use an offline algorithm in an online setting. Recently, the model of advice complexity of online problems has been introduced to enable a more fine-grained analysis of the hardness of online problems. The idea here is to measure what amount of information about the yet unknown parts of the input is necessary to compute an optimal (or near-optimal) solution online [3, 5, 7, 11]. For this, we analyze online algorithms that have access to an arbitrary prefix of an infinite tape with *advice bits* that was computed by some oracle knowing the whole input in advance. The *advice complexity* of such an algorithm measures how many of these advice bits the algorithm reads during its computation. As usual, the advice complexity of an online problem is defined as the minimum amount of advice needed by some algorithm solving the problem. We are especially interested in *lower bounds* on the advice complexity. Such lower bounds do not only tell us something about the information content [11] of online problems, but they also carry over to a randomized setting where they imply lower bounds on the number of random decisions needed to compute a good solution [14].

It turns out that, for some problems, very little advice can drastically improve the competitive ratio of an online algorithm, e.g., for the simple knapsack problem, i.e., where the value and weight of each item are equal, a single bit of advice is sufficient to jump from being non-competitive at all to 2-competitiveness [2]. On the other hand, many problems require a linear (or even higher) amount of advice bits for computing an optimal solution [2, 3, 7]. The advice complexity of a coloring problem was first investigated in [9] where linear upper and lower bounds for coloring a path were shown.

In the second part of this paper, we investigate the advice complexity of the online coloring problem on bipartite graphs. We prove almost tight upper and lower bounds on the advice complexity of computing an optimal solution, more precisely, for a graph on n vertices, $n - 2$ advice bits are sufficient and $n - 3$ advice bits are necessary for this. Moreover, we prove almost matching linear upper and lower bounds on the advice complexity of computing a 3-coloring, namely an upper bound of $n/2$ and a lower bound of $\frac{n}{2} - 4$. We complement these results by an algorithm that uses less than $n/\sqrt{2^{k-1}}$ advice bits for coloring a bipartite graph online with k colors.

The paper is organized as follows. In Sect. 2, we formally define the online coloring problem and fix our notation. In Sect. 3, we consider online algorithms without advice and present the improved lower bound on the number of necessary colors for deterministic online coloring algorithms. The proof of this lower bound is contained in Sect. 4, while Sect. 5 is devoted to the advice complexity of the online coloring of bipartite graphs.

2 Preliminaries

In this section, we fix our notation and formally define the problem we are dealing with in this paper.

Definition 1 (Coloring) Let $G = (V, E)$ be an undirected and unweighted graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set E . A (proper) coloring of a graph G is a function $\text{col} : V \rightarrow S$ which assigns to every vertex $v_i \in V$ a color $\text{col}(v_i) \in S$ and has the property that $\text{col}(v_i) \neq \text{col}(v_j)$, for all $i, j \in \{1, 2, \dots, n\}$ with $\{v_i, v_j\} \in E$.

Usually, we consider the set $S = \{1, 2, \dots, n\} \subset \mathbb{N}^+$. Let $V' \subseteq V$, then we denote by $\text{col}(V')$ the set of colors assigned to the vertices in V' . To distinguish the coloring functions used by different algorithms, we denote, for an algorithm A , its coloring function by col_A . We denote the subgraph of $G = (V, E)$ induced by a vertex subset $V' \subseteq V$ by $G[V']$, i.e., $G[V'] = (V', E')$, where $E' = \{\{v, w\} \in E \mid v, w \in V'\}$.

Definition 2 (Online Graph) An instance $G^\prec = (G, \prec)$ for the online coloring problem consists of a graph $G = (V, E)$ and a linear ordering \prec on the vertex set $V = \{v_1, v_2, \dots, v_n\}$ with $v_i \prec v_j$ for $i < j$. In the online representation G^\prec of the graph G , the vertices of V appear in the order determined by \prec .

For $V_i = \{v_1, v_2, \dots, v_i\}$ we denote by $G^\prec[V_i]$ the online subgraph of G^\prec induced by V_i . Note that $G^\prec[V_n] = G^\prec$.

Informally speaking, $G^{\prec}[V_i]$ is derived from $G^{\prec}[V_{i-1}]$ by adding the vertex v_i together with its edges incident to vertices from V_{i-1} . Let \mathcal{G}_n denote the set of all online graph instances on n vertices. Then, \mathcal{G} is the set of all possible online graph instances for the online coloring problem for all $n \in \mathbb{N}^+$, i.e., $\mathcal{G} = \bigcup_{n \in \mathbb{N}^+} \mathcal{G}_n$.

With this, we can formally define the online coloring problem.

Definition 3 (Online Coloring Problem)

Input: $G^{\prec} = (G, \prec) \in \mathcal{G}$ with $G = (V, E)$ and $V = \{v_1, v_2, \dots, v_n\}$.

Output: $(c_1, c_2, \dots, c_n) \in (\mathbb{N}^+)^n$ such that $\text{col}(v_i) = c_i$ and $\text{col}: V \rightarrow \mathbb{N}^+$ is a coloring, for all $i \in \{1, 2, \dots, n\}$

Cost: Number of colors used by the coloring.

Goal: Minimum.

The coloring of online graphs G^{\prec} depends on the used algorithm and the ordering of the vertices. Therefore, we denote by $\text{col}_{A, G^{\prec}}(V')$ the coloring function of a given algorithm A and an online graph $G^{\prec} \in \mathcal{G}_n$ for the vertex set $V' \subseteq V_n$. We will omit the subscript whenever A and G^{\prec} are clear from the context.

In the following, we will restrict our attention to the class of bipartite, i.e., two-colorable graphs. We denote the subproblem of the online coloring problem restricted to bipartite input graphs by BIPCOL. In a bipartite graph $G = (V(G), E(G))$, the vertex set $V(G)$ can be partitioned into two subsets, called *shores* and denoted by $S_1(G)$ and $S_2(G)$, with the property that the edges in $E(G)$ connect only vertices from different shores. If the graph is clear from the context, we write V, E, S_1, S_2 instead of $V(G), E(G), S_1(G)$ and $S_2(G)$. We say that a color α is *common* in a bipartite graph if it appears on both shores of the bipartition.

Given two vertices v_i and v_j and a time step t , we write $v_i \leftrightarrow_t v_j$ iff there exists a path in $G^{\prec}[V_t]$ from v_i to v_j . It is always possible to partition V_t into connected components according to the equivalence relation \leftrightarrow_t , and we call such components $C_t(v_i) = [v_i]_{\leftrightarrow_t}$. In each connected component, the shore partition is unique up to swapping the shores.

We want to analyze BIPCOL giving bounds on the number of colors used in the online coloring process. These bounds will always depend on the number n of vertices in the final graph $G^{\prec} = G^{\prec}[V_n]$. Let A be an online coloring algorithm. We denote by $F_A(G^{\prec}) = |\text{col}_{A, G^{\prec}}(V_n)|$ the number of colors used by A to color the graph G . Then, $F_A(n) = \max_{G \in \mathcal{G}_n} F_A(G^{\prec})$ is the maximum number of colors A uses to color any online graph instance with n vertices in the final graph G^{\prec} . We say that $U : \mathbb{N} \rightarrow \mathbb{N}$ is an *upper bound* on the number of colors sufficient for online coloring, if there exists an online algorithm A such that, for all $n \in \mathbb{N}$, we have $F_A(n) \leq U(n)$. Hence, to get an upper bound U on the number of used colors, it is sufficient to find a deterministic online algorithm A coloring each graph from \mathcal{G}_n using $U(n)$ colors. Similarly, a function L is a *lower bound* on the number of colors necessary for online coloring any graph if, for any online algorithm A , there exists an infinite subset $X \subseteq \mathbb{N}$ such that $L : X \rightarrow \mathbb{N}$ and, for all $n \in X$, we have $L(n) \leq F_A(n)$, i.e., if, for every algorithm A , for infinitely many n , there is an online graph $G_A^{\prec}(n) \in \mathcal{G}_n$ for which

A needs at least $L(n)$ colors. Observe that a lower bound $L: X \rightarrow \mathbb{N}$ also implies a lower bound $\tilde{L}: \mathbb{N} \rightarrow \mathbb{N}$, where $\tilde{L}(n) = L(m)$ with $m = \max\{k \in X \mid k \leq n\}$.

In the second part of this paper, we study the advice complexity of online coloring of bipartite graphs. The idea is to consider an oracle that sees the whole input in advance and writes information about the input onto a binary infinite *advice tape* [3, 11]. The algorithm may access an arbitrary prefix of this tape at runtime, and the amount of advice bits read by the algorithm can serve as a fine-grained measure of the hardness of online problems. Formally, online algorithms with advice can be defined as follows.

Definition 4 (Online Algorithm with Advice [3, 11]) Let $I = (x_1, \dots, x_n)$ be an input of an online minimization problem. An *online algorithm A with advice* computes the output sequence $A^\varphi(I) = (y_1, \dots, y_n)$ such that y_i is computed from φ, x_1, \dots, x_i , where φ is the content of the advice tape, i.e., an infinite binary sequence. For some output sequence o , $\text{cost}(o)$ denotes the cost of o . An algorithm A is *c -competitive with advice complexity $b(n)$* if there exists some non-negative constant α such that, for every n and for each input sequence I of length at most n , there exists some φ such that $\text{cost}(A^\varphi(I)) \leq c \cdot \text{cost}(\text{Opt}(I)) + \alpha$ and at most the first $b(n)$ bits of φ have been accessed during the computation of $A^\varphi(I)$. Here, $\text{Opt}(I)$ denotes an optimal (offline) solution for I . If $\alpha = 0$, then A is called *strictly c -competitive*. A is optimal if it is strictly 1-competitive.

In this paper, we restrict our attention to strict competitiveness. For simplicity, we call a strictly c -competitive algorithm c -competitive from now on.

3 Online Coloring Without Advice

In this section, we deal with the competitive ratio of deterministic online algorithms without advice. The following upper bound is well known.

Theorem 1 (Lovász, Saks, and Trotter [15]) *There is an online algorithm using at most $2 \log_2 n$ colors for coloring any bipartite graph of n vertices.*

Proof Let $G^<$ be the input instance. We describe the algorithm A that works as follows: at each step t , consider the component $C_t(v_t)$ containing the last revealed vertex. If $C_t(v_t)$ contains only v_t , i.e., if v_t was revealed isolated, A outputs $\text{col}_A(v_t) = 1$, otherwise it assigns to v_t the smallest color not present on the opposite shore of this component. More formally, assuming $v_t \in S_1(C_t(v_t))$, A outputs

$$\text{col}_A(v_t) = \min \{c \geq 1 \mid c \neq \text{col}_A(v) \text{ for all } v \in S_2(C_t(v_t))\}.$$

By calling $B(k)$ the minimum number of vertices required for A to output color k , we have that $B(2) = 2$ and $B(3) = 4$. We inductively show that $B(k) \geq 2^{\frac{k}{2}}$, which implies that, on an instance of n vertices, A uses at most $2 \log_2(n)$ colors.

If $\text{col}_A(v_t) = k$ and $v_t \in S_1(C_t(v_t))$, it means that on the shore $S_2(C_t(v_t))$ all colors from 1 to $k - 1$ are present. Similarly, since color $k - 1$ was assigned to

some vertex in $S_2(C_t(v_t))$, then on the shore $S_1(C_t(v_t))$ all colors from 1 to $k - 2$ are present. Since there are two vertices $v_p \in S_1(C_t(v_t))$ and $v_q \in S_2(C_t(v_t))$ such that $\text{col}_A(v_p) = \text{col}_A(v_q) = k - 2$, the only way A would assign that color is if v_p and v_q were on two different components of G_r , where $r = \max\{p, q\}$. By induction hypothesis, each of these components must have at least $2^{\frac{k-2}{2}}$ vertices, therefore $B(k) \geq 2 \cdot 2^{\frac{k-2}{2}} = 2^{\frac{k}{2}}$. \square

There is also a well-known lower bound which even holds for trees.

Theorem 2 (Bean [1]) *For every $k \in \mathbb{N}^+$, there exists a tree $T_k^<$ on 2^{k-1} vertices such that, for every deterministic online coloring algorithm A , $\text{col}_A(T_k^<) \geq k$.*

Theorem 2 immediately implies that there exists an infinite number of trees (and thus of bipartite graphs) forcing any online algorithm to use at least $\log_2 n + 1$ colors on any graph on n vertices from this class. In the remainder of this section, we improve on this result by describing a graph class, which forces every coloring algorithm A to use even more colors on infinitely many graphs of the class. This class is built recursively. In the proof, we will focus, for a fixed deterministic online coloring algorithm A , only on those $G^<[V_i]$'s in an instance $G^< \in \mathcal{G}_n$ in which the new vertex v_i gets a new color with respect to the previous graph $G^<[V_{i-1}]$.

Lemma 1 *For every $k \in \mathbb{N}^+$ and every online coloring algorithm A , there exists an online graph $G_A^<(k)$ such that:*

1. $F_A(G_A^<(k)) \geq k$,
2. $F_A(S_1(G_A^<(k))) \geq k - 2$,
3. $F_A(S_2(G_A^<(k))) \geq k - 1$,
4. $|V(G_A^<(k))| \leq W(k) := W(k - 1) + W(k - 2) + W(k - 3) + 1$, for $k \geq 3$, and $W(0) = 0$, $W(1) = 1$, and $W(2) = 2$.

Consequently, $W(k)$ is the maximum number of vertices that a graph needs in order to force an arbitrary algorithm A to use at least k colors.

We will prove Lemma 1 in the following section. The recurrence given by property 4 of Lemma 1 can be resolved as follows.

Lemma 2 *Let $W(k)$ be defined as in Lemma 1. Then,*

$$W(k) \leq 1.35527 \cdot 1.83929^k - 0.400611.$$

Proof It can be easily shown by induction that $W(k) = \sum_{n=0}^{k+1} T(n)$, where $T(n)$ is the n -th Tribonacci number (see [8, 16]). The number $T(n)$ can be computed as follows:

$$T(n) = 3b \cdot \frac{(\frac{1}{3}(a_+ + a_- + 1))^n}{b^2 - 2b + 4} \leq 0.336229 \cdot 1.83929^n,$$

where $a_+ = (19 + 3\sqrt{33})^{\frac{1}{3}}$, $a_- = (19 - 3\sqrt{33})^{\frac{1}{3}}$, and $b = (586 + 102\sqrt{33})^{\frac{1}{3}}$. Summing up the values of $T(n)$ for $n \in \{0, \dots, k + 1\}$ gives the claimed result. \square

Theorem 3 For any online coloring algorithm A , there exists an infinite sequence of online graphs $G_A^<(k) \in \mathcal{G}_{n_k}$ with $n_k < n_{k+1}$ for all $k \in \mathbb{N}$ such that A needs at least

$$\lfloor 1.13746 \cdot \log_2(n_k) - 0.49887 \rfloor$$

colors to color $G_A^<(k)$.

Proof The claim follows immediately from Lemmas 1 and 2 by resolving the following inequality for k :

$$\begin{aligned} n_k &\leq 1.35527 \cdot 1.83929^k \\ \implies \log_2(n_k) &\leq \log_2(1.35527) + k \cdot \log_2(1.83929) \\ \implies k &\geq \frac{\log_2(n_k) - \log_2(1.35527)}{\log_2(1.83929)} = 1.13746 \cdot \log_2(n_k) - 0.49887 \quad \square \end{aligned}$$

4 Proof of Lemma 1

In this section, we prove Lemma 1. We proceed by an induction over k , the number of colors. For every k , we generate a class $\tilde{\mathcal{G}}(k)$ consisting of online graphs defined as

$$\tilde{\mathcal{G}}(k) = \{G_B^<(k) \mid B \text{ is an online coloring algorithm and properties 1 to 4 of Lemma 1 are satisfied}\}.$$

Hence, for a fixed k , we will find in $\tilde{\mathcal{G}}(k)$, for every online coloring algorithm B , an instance $G_B^<(k)$ that forces B to use at least k colors to color $G_B^<(k)$. Those instances are built inductively. We will prove that we can construct, for any online coloring algorithm A , a hard instance $G_A^<(k)$, using three online graphs $G_{k-1}^< \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2}^< \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3}^< \in \tilde{\mathcal{G}}(k-3)$, that are revealed in this order, and an additional vertex v (see Fig. 1).

Let $H(k)$ be the **induction hypothesis**, formulated as follows:

$$\mathbf{H}(k): \quad \text{For all } j \leq k \text{ and all online algorithms } B, \text{ there exists a graph } G_B^<(j) \in \tilde{\mathcal{G}}(j).$$

Assuming $H(k-1)$ holds, the hypothesis states that, for every online algorithm A , a graph $G_{k-1}^< = G_A^<(k-1) \in \tilde{\mathcal{G}}(k-1)$ satisfying all the properties of Lemma 1 exists. To show the existence of the second and third constructed subgraph, $G_{k-2}^< \in \tilde{\mathcal{G}}(k-2)$ and $G_{k-3}^< \in \tilde{\mathcal{G}}(k-3)$, we have to take into account that the algorithm A already knows a part of the instance, and hence it may behave differently from the case where there is no part known.

We merge the shores of $G_{k-1}^<$, $G_{k-2}^<$, and $G_{k-3}^<$ in an appropriate way and, using an additional vertex v , we ensure that the resulting graph $G_A^<(k)$ is in $\tilde{\mathcal{G}}(k)$. In some cases, we do not need all four components to guarantee that all four properties of Lemma 1 are satisfied.

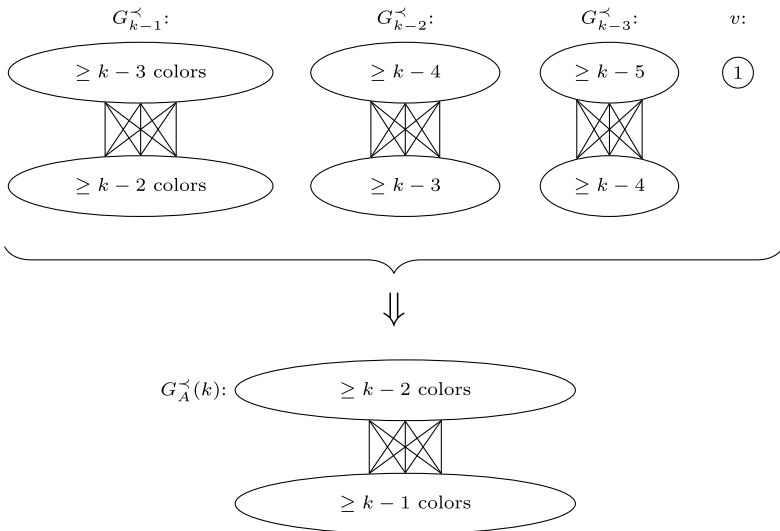
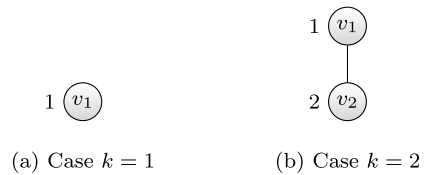


Fig. 1 Using three online graphs $G_{k-1}^< \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2}^< \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3}^< \in \tilde{\mathcal{G}}(k-3)$ and a vertex v , we can construct, for any online algorithm A , a new online graph $G_A^<(k) \in \tilde{\mathcal{G}}(k)$

Fig. 2 Base cases: W.l.o.g., the vertices are colored as indicated. The indices of the vertices indicate their order of appearance



We merge two online graph instances $G^<[V] = (G, <) \in \mathcal{G}$ with $V = \{v_1, v_2, \dots, v_n\}$ and $\overline{G}^<[V'] = (G', <) \in \mathcal{G}$ with $V' = \{v'_1, v'_2, \dots, v'_m\}$ to an instance $M^<[V''] = G^<[V] \circ \overline{G}^<[V']$, defined as

$$M^<[V''] = (G \cup G', <) \in \mathcal{G},$$

where, for two graphs $G = (V, E)$ and $G' = (V', E')$ with $V \cap V' = \emptyset$, $G \cup G'$ is defined as the graph $(V'' = \{v_1, \dots, v_n, v'_1, \dots, v'_m\}, E \cup E')$ and $v_n < v'_1$.

4.1 Base Cases ($k \leq 3$)

For $k \in \{0, 1, 2\}$, it is easy to see that the hypothesis $H(k)$ is satisfied (see Fig. 2).

In case $k = 3$, for every online coloring algorithm A , $G_A^<(3)$ can be constructed recursively using two graphs $G_2^< \in \tilde{\mathcal{G}}(2)$, $G_1^< \in \tilde{\mathcal{G}}(1)$ and possibly a new vertex. The vertices of $G_2^< = G_A^<(3)[\{v_1, v_2\}]$ are colored, w.l.o.g., with 1 and 2, and $G_1^< = G_A^<(3)[\{v_3\}]$ can be colored, w.l.o.g., with 1, 2 or 3 (see Fig. 3).

If the algorithm colors v_1, v_2 , and v_3 with different colors, as shown in case (c) of Fig. 3, obviously all properties of $H(3)$ are already satisfied. Otherwise, we have

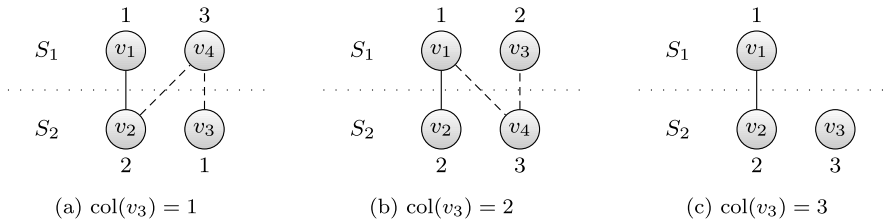


Fig. 3 Case $k = 3$: For any behaviour of an algorithm A , we can construct the graph $G_A^{\prec}(3)$ using graphs $G_2^{\prec} = G_A^{\prec}(3)[\{v_1, v_2\}] \in \tilde{\mathcal{G}}(2)$ and $G_1^{\prec} = G_A^{\prec}(3)[\{v_3\}] \in \tilde{\mathcal{G}}(1)$ that force A to use a third color

to add one new vertex v_4 , which is connected to two vertices with different colors to force every online coloring algorithm A to use a third color (see (a) and (b) of Fig. 3).

4.2 Inductive Step ($k \geq 4$)

For every online algorithm A and every $k \in \mathbb{N}^+$, we construct $G_A^{\prec}(k)$ in four steps using three graphs $G_{k-1}^{\prec} \in \tilde{\mathcal{G}}(k - 1)$, $G_{k-2}^{\prec} \in \tilde{\mathcal{G}}(k - 2)$, $G_{k-3}^{\prec} \in \tilde{\mathcal{G}}(k - 3)$, and an additional vertex v .

First, assuming that $H(k - 1)$ holds, we show that, for every algorithm A , we can construct three graphs G_{k-1}^{\prec} , G_{k-2}^{\prec} , and G_{k-3}^{\prec} in this order satisfying all the properties of Lemma 1. Then, we show that we can merge them, using an additional vertex v , to a graph $G_A^{\prec}(k) \in \tilde{\mathcal{G}}(k)$.

4.2.1 Existence of the Graphs G_{k-1}^{\prec} , G_{k-2}^{\prec} , and G_{k-3}^{\prec}

We assume that $H(k - 1)$ holds. Hence, for every online coloring algorithm B and $j \in \{k - 1, k - 2, k - 3\}$ there exists a graph $G_B^{\prec}(j) \in \tilde{\mathcal{G}}(j)$.

Step 1: Because of $H(k - 1)$, we know that a graph $G_{k-1}^{\prec} = G_A^{\prec}(k - 1)$ exists.

Step 2: In the next phase, algorithm A receives a second subgraph. We cannot simply use the graph $G_A^{\prec}(k - 2)$ here, whose existence is guaranteed by $H(k - 2)$, since $H(k - 2)$ guarantees the hardness of this input only in the case that A reads it from its initial configuration. Having already read G_{k-1}^{\prec} , A might behave differently on $G_A^{\prec}(k - 2)$. We denote by $A|_{G_{k-1}^{\prec}}$ the work of algorithm A having already processed the graph G_{k-1}^{\prec} . $A|_{G_{k-1}^{\prec}}$ can be simulated by an algorithm B , which does the same work as $A|_{G_{k-1}^{\prec}}$ but which did not receive any other graph before. In other words, if we think of the algorithms as Turing machines, B uses the same transitions as A , but its initial configuration (state and tape content) is the same as the configuration reached by A after processing G_{k-1}^{\prec} . Because of $H(k - 1)$, and thus $H(k - 2)$, we know that, for such an algorithm B , there is a graph $G_{k-2}^{\prec} = G_B^{\prec}(k - 2) = G_{A|_{G_{k-1}^{\prec}}}^{\prec}(k - 2) \in \tilde{\mathcal{G}}(k - 2)$.

Step 3: Now, algorithm A gets a third subgraph. Again, the work of $A|_{G_{k-1}^{\prec} \circ G_{k-2}^{\prec}}$ can be simulated by an algorithm C . Because of the induction hypothesis, a graph $G_{k-3}^{\prec} = G_C^{\prec}(k - 3) = G_{A|_{G_{k-1}^{\prec} \circ G_{k-2}^{\prec}}}^{\prec}(k - 3)$ exists.

Hence, we have the graphs G_{k-1}^{\prec} , G_{k-2}^{\prec} , and G_{k-3}^{\prec} at our disposal and can force a new color, possibly with the help of an additional vertex v , as follows.

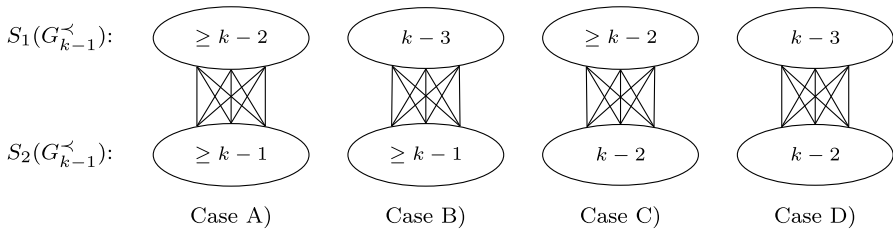


Fig. 4 There are four cases how the colors in $G_{k-1}^{<}$ can be distributed

4.2.2 Construction of Graph $G_A^{<}(k)$

The graphs $G_{k-1}^{<}$, $G_{k-2}^{<}$, and $G_{k-3}^{<}$ are presented in this order. Beginning with graph $G_{k-1}^{<}$, we distinguish four possible cases for an online algorithm A (see Fig. 4):

- (A) A uses, w.l.o.g., $\geq k - 2$ colors on $S_1(G_{k-1}^{<})$ and $\geq k - 1$ colors on $S_2(G_{k-1}^{<})$.
- (B) A uses, w.l.o.g., $k - 3$ colors on $S_1(G_{k-1}^{<})$ and $\geq k - 1$ colors on $S_2(G_{k-1}^{<})$.
- (C) A uses, w.l.o.g., $\geq k - 2$ colors on $S_1(G_{k-1}^{<})$ and $k - 2$ colors on $S_2(G_{k-1}^{<})$.
- (D) A uses $k - 3$ colors on $S_1(G_{k-1}^{<})$ and $k - 2$ colors on $S_2(G_{k-1}^{<})$.

Since $H(k - 1)$ holds, graph $G_{k-1}^{<}$ is colored by algorithm A with at least $k - 1$ colors, and $S_1(G_{k-1}^{<})$ contains at least $k - 3$ and $S_2(G_{k-1}^{<})$ at least $k - 2$ colors. Therefore, after constructing $G_{k-1}^{<}$, the algorithm A encounters one of the four cases above.

To finish the construction, we have to ensure that the final graph contains at least k colors. If this is not yet the case for $G_{k-1}^{<}$, we need to force the algorithm to use a new color k .

We will show that, in order to satisfy the properties 2 and 3 of Lemma 1, either the graphs $G_{k-2}^{<}$ and $G_{k-3}^{<}$ contain some of the colors that appear only on one shore of $G_{k-1}^{<}$, or there are enough additional colors in $G_{k-2}^{<}$ and $G_{k-3}^{<}$, which do not appear in $G_{k-1}^{<}$. Only in some cases will we need all three graphs $G_{k-1}^{<}$, $G_{k-2}^{<}$, $G_{k-3}^{<}$ and the vertex v . In many cases, a subset of those graphs is sufficient to construct a graph $G_A^{<}(k) \in \tilde{\mathcal{G}}(k)$.

4.2.3 (A) $|\text{col}(S_1(G_{k-1}^{<}))| \geq k - 2$ and $|\text{col}(S_2(G_{k-1}^{<}))| \geq k - 1$

If the graph $G_{k-1}^{<}$ contains at least k colors, properties 1, 2, and 3 of Lemma 1 are satisfied. Furthermore, we have

$$\begin{aligned}
 |V(G_A^{<}(k))| &= |V(G_{k-1}^{<})| \leq W(k - 1) \leq W(k - 1) + W(k - 2) + W(k - 3) + 1 \\
 &= W(k).
 \end{aligned}$$

Hence, all the properties of Lemma 1 are satisfied and we can finish the construction without using additional subgraphs.

Now, assume $G_{k-1}^{<}$ contains only $k - 1$ colors. To satisfy property 1, we need to force every algorithm to use one more color. Connecting an additional vertex v to all vertices in $S_2(G_{k-1}^{<})$, and thus adding it to $S_1(G_{k-1}^{<})$, forces the algorithm to use

color k :

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\leftarrow}) \cup \{v\}) &: \boxed{1 \ 2 \ \cdots \ k-2} \quad \text{---} \quad \textcircled{k} \\ \text{col}(S_2(G_{k-1}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-2 \ k-1} \end{aligned}$$

We have

$$\begin{aligned} |V(G_A^{\leftarrow}(k))| &= |V(G_{k-1}^{\leftarrow})| + 1 \leq W(k-1) + 1 \\ &\leq W(k-1) + W(k-2) + W(k-3) + 1 = W(k) \end{aligned}$$

and therefore all properties of Lemma 1 are satisfied.

4.2.4 (B) $|\text{col}(S_1(G_{k-1}^{\leftarrow}))| = k - 3$ and $|\text{col}(S_2(G_{k-1}^{\leftarrow}))| \geq k - 1$

Because we assume that the induction hypothesis holds, G_{k-2}^{\leftarrow} is colored by algorithm A with at least $k - 2$ colors. Therefore, there exists, w.l.o.g., a color $a \in \text{col}(S_1(G_{k-2}^{\leftarrow}))$ such that $a \notin \text{col}(S_1(G_{k-1}^{\leftarrow}))$. Hence, merging G_{k-1}^{\leftarrow} and G_{k-2}^{\leftarrow} such that color a is added to shore S_1 , we obtain an analogous situation as in case (A):

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-3 \ a} \\ \text{col}(S_2(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-2 \ k-1} \end{aligned}$$

4.2.5 (C) $|\text{col}(S_1(G_{k-1}^{\leftarrow}))| \geq k - 2$ and $|\text{col}(S_2(G_{k-1}^{\leftarrow}))| = k - 2$

If $|\text{col}(S_1(G_{k-1}^{\leftarrow}))| \geq k - 1$ holds, we can swap the shores and obtain case (A). Therefore, we can assume that $|\text{col}(S_1(G_{k-1}^{\leftarrow}))| = k - 2$.

Because G_{k-1}^{\leftarrow} is colored with at least $k - 1$ colors, its shores can have at most $k - 3$ common colors:

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-3 \ b} \\ \text{col}(S_2(G_{k-1}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-3 \ a} \end{aligned}$$

Hence, G_{k-2}^{\leftarrow} has to contain either color a, b or a new color $c \notin \text{col}(G_{k-1}^{\leftarrow})$, w.l.o.g. $b \in \text{col}(S_1(G_{k-2}^{\leftarrow}))$ or $c \in \text{col}(S_1(G_{k-2}^{\leftarrow}))$. Merging G_{k-1}^{\leftarrow} and G_{k-2}^{\leftarrow} in an appropriate way leads again to a situation as in case (A):

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-3 \ b} && \boxed{1 \ 2 \ \cdots \ k-3 \ b} \\ \text{col}(S_2(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-3 \ a} \boxed{b} && \text{or} && \boxed{1 \ 2 \ \cdots \ k-3 \ a} \boxed{c} \end{aligned}$$

4.2.6 (D) $|\text{col}(S_1(G_{k-1}^{\prec})| = k - 3$ and $|\text{col}(S_2(G_{k-1}^{\prec})| = k - 2$

The shores of G_{k-1}^{\prec} contain at most $k - 4$ common colors since we have to have at least $k - 1$ colors in total:

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ c} \\ \text{col}(S_2(G_{k-1}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \end{aligned}$$

To satisfy all properties of Lemma 1, we need either one additional color on each shore or two additional colors in $S_1(G_{k-1}^{\prec})$. And we have to force a new color k for the construction of graph $G_A^{\prec}(k)$. We distinguish five cases according to the sets of colors present in G_{k-2}^{\prec} and G_{k-3}^{\prec} :

1. **There are $d \in \text{col}(G_{k-2}^{\prec})$ and $e \in \text{col}(G_{k-3}^{\prec})$ with $d, e \notin \text{col}(G_{k-1}^{\prec})$:**

W.l.o.g., $d \in \text{col}(S_1(G_{k-2}^{\prec}))$ and $e \in \text{col}(S_1(G_{k-3}^{\prec}))$. Then the following combination of the shores leads to $G_A^{\prec}(k) \in \tilde{\mathcal{G}}(k)$:

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\prec}) \cup S_1(G_{k-2}^{\prec}) \cup S_2(G_{k-3}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ c} \boxed{d} \\ \text{col}(S_2(G_{k-1}^{\prec}) \cup S_2(G_{k-2}^{\prec}) \cup S_1(G_{k-3}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \boxed{e} \end{aligned}$$

The number of colors in $S_1(G_A^{\prec}(k))$ is larger than $k - 2$ and in $S_2(G_A^{\prec}(k))$ it is larger than $k - 1$. The total number of colors is at least k (in the case of $d \neq e$ we have at least $k + 1$ colors). And with

$$\begin{aligned} |V(G_A^{\prec}(k))| &= |V(G_{k-1}^{\prec})| + |V(G_{k-2}^{\prec})| + |V(G_{k-3}^{\prec})| \\ &\leq W(k - 1) + W(k - 2) + W(k - 3) \leq W(k), \end{aligned}$$

we have $G_A^{\prec}(k) \in \tilde{\mathcal{G}}(k)$.

2. **$\text{col}(G_{k-2}^{\prec}) \subseteq \text{col}(G_{k-1}^{\prec})$ and one new color $e \in \text{col}(G_{k-3}^{\prec}) \setminus \text{col}(G_{k-1}^{\prec})$:**

Since G_{k-2}^{\prec} contains at least $k - 2$ colors and it is colored with a subset of colors of G_{k-1}^{\prec} , it has to contain at least one color of the set $\{a, b\}$, say $a \in S_1(G_{k-2}^{\prec})$ and $e \in \text{col}(S_1(G_{k-3}^{\prec}))$. Then we get

$$\begin{aligned} \text{col}(S_1(G_{k-1}^{\prec}) \cup S_1(G_{k-2}^{\prec}) \cup S_2(G_{k-3}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ c} \boxed{a} \\ \text{col}(S_2(G_{k-1}^{\prec}) \cup S_2(G_{k-2}^{\prec}) \cup S_1(G_{k-3}^{\prec})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \boxed{e} \end{aligned}$$

Then $G_A^{\prec}(k) \in \tilde{\mathcal{G}}(k)$ since

$$\begin{aligned} |V(G_A^{\prec}(k))| &= |V(G_{k-1}^{\prec})| + |V(G_{k-2}^{\prec})| + |V(G_{k-3}^{\prec})| \\ &\leq W(k - 1) + W(k - 2) + W(k - 3) \leq W(k). \end{aligned}$$

3. $\text{col}(G_{k-3}^{\leftarrow}) \subseteq \text{col}(G_{k-1}^{\leftarrow})$ and one new color $d \in \text{col}(G_{k-2}^{\leftarrow}) \setminus \text{col}(G_{k-1}^{\leftarrow})$:

With the same argumentation as above, we conclude that G_{k-3}^{\leftarrow} must contain at least one of the colors $\{a, b, c\}$. Then, we have the same situation as in the case before (in the case of $c \in \text{col}(G_{k-3}^{\leftarrow})$, the merging of the shores is reversed).

In the following cases, we can assume that

$$\text{col}(G_{k-2}^{\leftarrow}) \subseteq \text{col}(G_{k-1}^{\leftarrow}) \text{ and } \text{col}(G_{k-3}^{\leftarrow}) \subseteq \text{col}(G_{k-1}^{\leftarrow}).$$

Because G_{k-3}^{\leftarrow} is colored with at least $k - 3$ colors, it contains one of the three colors $\{a, b, c\}$. Analogously, G_{k-2}^{\leftarrow} contains two of the three colors $\{a, b, c\}$. Then, either c appears in one of the graphs G_{k-2}^{\leftarrow} or G_{k-3}^{\leftarrow} , or one of the colors $\{a, b\}$ is in G_{k-3}^{\leftarrow} and both a and b are in G_{k-2}^{\leftarrow} :

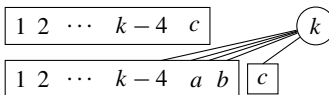
4. $c \in \text{col}(G_{k-2}^{\leftarrow}) \cup \text{col}(G_{k-3}^{\leftarrow})$:

W.l.o.g., assume $c \in \text{col}(S_1(G_{k-2}^{\leftarrow}))$. The other cases are analogous.

$$\text{col}(S_1(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow})) : \boxed{1 \ 2 \ \cdots \ k-4 \ c}$$

$$\text{col}(S_2(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})) : \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \ \boxed{c}$$

$S_1(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow})$ contains at least $k - 3$ colors and $S_2(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})$ consists of at least $k - 1$ colors. The k -th color, say color k , can be forced by adding a new vertex v , which is connected to all vertices in $S_2(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})$. Then, the resulting graph $G_A^{\leftarrow}(k)$ will look as follows:

$$\begin{array}{l} \text{col}(S_1(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow}) \cup \{v\}) : \boxed{1 \ 2 \ \cdots \ k-4 \ c} \\ \text{col}(S_2(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow})) : \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \ \boxed{c} \end{array}$$


Hence, we have $G_A^{\leftarrow}(k) \in \tilde{\mathcal{G}}(k)$ because $S_1(G_A^{\leftarrow}(k))$ contains $k - 2$ colors, $S_2(G_A^{\leftarrow}(k))$ is colored with at least $k - 1$ colors, and

$$\begin{aligned} |V(G_A^{\leftarrow}(k))| &= |V(G_{k-1}^{\leftarrow})| + |V(G_{k-2}^{\leftarrow})| + 1 \\ &\leq W(k - 1) + W(k - 2) + 1 \leq W(k). \end{aligned}$$

5. $a, b \in \text{col}(G_{k-2}^{\leftarrow})$ and one of those colors is in G_{k-3}^{\leftarrow} :

W.l.o.g., let $a \in \text{col}(S_1(G_{k-2}^{\leftarrow}))$ and $b \in \text{col}(S_1(G_{k-3}^{\leftarrow}))$. The shores can be matched as follows:

$$\text{col}(S_1(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow}) \cup S_1(G_{k-3}^{\leftarrow})) : \boxed{1 \ 2 \ \cdots \ k-4 \ c} \ \boxed{a} \ \boxed{b}$$

$$\text{col}(S_2(G_{k-1}^{\leftarrow}) \cup S_2(G_{k-2}^{\leftarrow}) \cup S_2(G_{k-3}^{\leftarrow})) : \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b}$$

We can force an algorithm to use color k by introducing a new vertex v that is connected to all vertices in $S_1(G_{k-1}^{\leftarrow}) \cup S_1(G_{k-2}^{\leftarrow}) \cup S_1(G_{k-3}^{\leftarrow})$:

$$\begin{aligned} \text{col}(\mathcal{S}_1(G_{k-1}^{\leftarrow}) \cup \mathcal{S}_1(G_{k-2}^{\leftarrow}) \cup \mathcal{S}_1(G_{k-3}^{\leftarrow})) &: \boxed{1 \ 2 \ \cdots \ k-4 \ c} \ \boxed{a} \ \boxed{b} \\ \text{col}(\mathcal{S}_2(G_{k-1}^{\leftarrow}) \cup \mathcal{S}_2(G_{k-2}^{\leftarrow}) \cup \mathcal{S}_2(G_{k-3}^{\leftarrow}) \cup \{v\}) &: \boxed{1 \ 2 \ \cdots \ k-4 \ a \ b} \ \textcircled{k} \end{aligned}$$

Now, both shores contain at least $k - 1$ colors and hence $G_A^{\leftarrow}(k) \in \tilde{\mathcal{G}}(k)$ because

$$\begin{aligned} |V(G_A^{\leftarrow}(k))| &= |V(G_{k-1}^{\leftarrow})| + |V(G_{k-2}^{\leftarrow})| + |V(G_{k-3}^{\leftarrow})| + 1 \\ &\leq W(k - 1) + W(k - 2) + W(k - 3) + 1 = W(k). \end{aligned}$$

Note that this is the hardest case, leading to exactly the recurrence from property 4 in Lemma 1.

5 Advice Complexity

In this section, we investigate the advice complexity of the online coloring problem on bipartite graphs. We start with giving an upper bound on the amount of advice needed for achieving an optimal coloring.

Theorem 4 *There exists an online algorithm for BIPCOL, which uses at most $n - 2$ advice bits to be optimal on every instance of length n .*

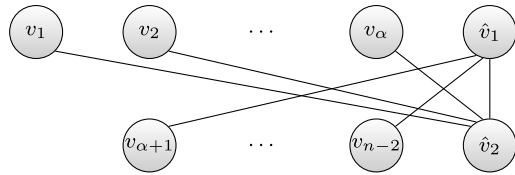
Proof Intuitively, we want to give advice only for those vertices that appear without connections to the vertices appearing before them. In the following, we call those vertices *isolated* (although they might get connected to some other vertices appearing later). We cannot reach the upper bound of $n - 2$ by simply asking for one bit of advice for every vertex that is isolated, except the first and the last (if isolated) vertices in the input sequence, because this strategy would require knowing the input length in advance, since the advice tape is infinite and it is up to the algorithm to decide how many bits to read. Therefore, in order to achieve the desired bound, we present the algorithm A_2 that works as follows.

- The first vertex receives color 1.
- Then the algorithm asks for one bit of advice: if it is 1, then A_2 will assign color 1 to every isolated vertex, otherwise it will ask for a bit of advice for every further isolated vertex, to decide whether to assign color 1 or 2.
- Any vertex that has an edge to some previously received vertex v receives the opposite color with respect to v .

It is easy to see that, on an input of length n , whenever there are at least $n - 1$ isolated vertices, assigning color 1 to every isolated vertex is an optimal strategy, therefore the appropriate advice is a string of length one. This implies that the first advice bit is 0 only when at most $n - 2$ vertices are isolated in the input sequence. Since the first vertex is among them and does not need any advice, the upper bound of $n - 2$ holds. \square

We can complement this result by an almost matching lower bound.

Fig. 5 Structure of the graph G_α used in the proof of Theorem 5. The set of edges is $E = \{\{v_s, \hat{v}_2\} \mid 1 \leq s \leq \alpha\} \cup \{\{v_s, \hat{v}_1\} \mid \alpha + 1 \leq s \leq n - 2\} \cup \{\{\hat{v}_1, \hat{v}_2\}\}$



Theorem 5 Any deterministic online algorithm for BIPCOL needs at least $n - 3$ advice bits to be optimal on every instance of length n .

Proof For a contradiction, assume there exists an algorithm \hat{A} for BIPCOL that uses 2 colors and less than $n - 3$ bits of advice. Given, for any $0 \leq \alpha \leq n - 2$, the graph G_α with n vertices described in Fig. 5, we consider as the set of possible instances of \hat{A} any online presentation of G_α , for all $0 \leq \alpha \leq n - 2$, such that the first $n - 2$ vertices are presented as a permutation of the vertices $\{v_j\}_{1 \leq j \leq n-2}$. This means, the algorithm will always receive isolated vertices until time step $n - 2$. Hence, \hat{A} will be able to color \hat{v}_1 and \hat{v}_2 with values in $\{1, 2\}$ only if v_1, \dots, v_α all have the same color, and $v_{\alpha+1}, \dots, v_{n-2}$ all have the opposite color.

Since there is a bijection between the instances and all possible permutations of the first $n - 2$ revealed vertices $\pi_i = (\pi_i(1), \pi_i(2), \dots, \pi_i(n - 2))$, we define an equivalence relation among the input instances in the following way: we say that two instances are equivalent iff the order of the first $n - 2$ vertices reflects the same shore partition. More formally, for all $t \leq n - 2$ and all permutations π_i , let

$$S(\pi_i(t)) := \begin{cases} S_1(G) & \text{if } \pi_i(t) \in S_1(G) \\ S_2(G) & \text{else} \end{cases}$$

be the shore containing $\pi_i(t)$. Then, $\pi_i \sim \pi_j$ iff, for all $t_1, t_2 \leq n - 2$,

$$S(\pi_i(t_1)) = S(\pi_i(t_2)) \iff S(\pi_j(t_1)) = S(\pi_j(t_2)).$$

It is not hard to see that \sim is an equivalence relation and, by a counting argument, the number of equivalence classes of \sim is $\frac{2^{n-2}}{2} = 2^{n-3}$.

To prove the claimed lower bound, it is sufficient to show that \hat{A} needs a different advice string for each equivalence class. Suppose, for contradiction, that $\pi_i \approx \pi_j$ and \hat{A} receives the same advice string for both instances. Then, since all instances look the same until time step $n - 2$, this implies $\text{col}_{\hat{A}}(\pi_i(t)) = \text{col}_{\hat{A}}(\pi_j(t))$, for all $t \leq n - 2$.

Because the two instances are not equivalent, there are two values $t_1, t_2 \leq n - 2$, with $t_1 \neq t_2$, such that $\pi_i(t_1)$ and $\pi_i(t_2)$ are on the same shore, while $\pi_j(t_1)$ and $\pi_j(t_2)$ are on opposite shores. We then have two cases (see Fig. 6):

Case (a): If $\text{col}_{\hat{A}}(\pi_i(t_1)) \neq \text{col}_{\hat{A}}(\pi_i(t_2))$, then in the instance associated to π_i , either \hat{v}_1 or \hat{v}_2 is forced to have a third color assigned, since one of them will be on the opposite shore with respect to both $\pi_i(t_1)$ and $\pi_i(t_2)$.

Case (b): If $\text{col}_{\hat{A}}(\pi_i(t_1)) = \text{col}_{\hat{A}}(\pi_i(t_2))$, then $\text{col}_{\hat{A}}(\pi_j(t_1)) = \text{col}_{\hat{A}}(\pi_j(t_2))$. W.l.o.g., we can assume that, in the instance associated to π_j , the vertex \hat{v}_1 is on the shore

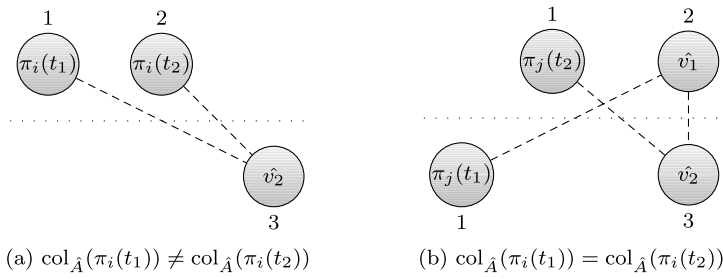
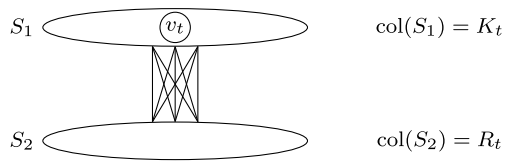


Fig. 6 Two non-equivalent instances with the same advice string in the proof of Theorem 5

Fig. 7 The sets K_t and R_t in the proof of Theorem 6



opposite to $\pi_j(t_1)$, hence there is an edge $\{\hat{v}_1, \pi_j(t_1)\}$ and as a consequence we must have $\text{col}_{\hat{A}}(\hat{v}_1) \neq \text{col}_{\hat{A}}(\pi_j(t_1))$ and therefore $\text{col}_{\hat{A}}(\hat{v}_1) \neq \text{col}_{\hat{A}}(\pi_j(t_2))$, but since \hat{v}_1 and $\pi_j(t_2)$ are on the same shore, which is opposite to \hat{v}_2 , the algorithm \hat{A} is forced to assign a third color to \hat{v}_2 . \square

We now analyze how much advice is sufficient to guarantee a given constant competitive ratio.

Theorem 6 For any integer constant $k > 2$, there exists an online algorithm for BIPCOL that needs less than $\frac{n}{\sqrt{2^{k-1}}}$ advice bits to color every instance of length n with at most k colors.

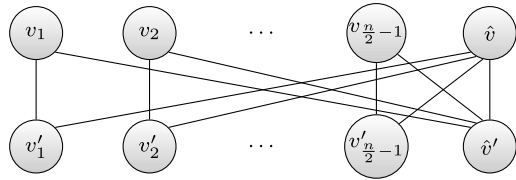
Proof We will consider an algorithm A_k that is an adaptation of the algorithm A used in the proof of Theorem 1: the idea is to make A_k ask for an advice bit only when it is about to assign color $k - 1$, in order to avoid assigning that color to vertices on both shores of the final graph. This implies that the algorithm will always have vertices of color $k - 1$ (if any) only on one shore and vertices of color k (if any) only on the other shore, so that color $k + 1$ will never be needed.

We now describe the algorithm A_k at step t , when the vertex v_t is revealed. Consider the connected component $C_t(v_t)$ to which v_t belongs at time step t .

By calling K_t (R_t , respectively) the set of colors assigned to vertices of $C_t(v_t)$ on the same (opposite, respectively) shore as v_t (see Fig. 7), A_k will choose its output as follows:

- if R_t does not contain all the colors smaller than $k - 1$, then $\text{col}_{A_k}(v_t) = \min\{c \geq 1 \mid c \notin R_t\}$,
- if either $k - 1 \in R_t$ or $k \in K_t$, then $\text{col}_{A_k}(v_t) = k$,
- if either $k \in R_t$ or $k - 1 \in K_t$, then $\text{col}_{A_k}(v_t) = k - 1$,

Fig. 8 The graph G' used in the proof of Theorem 7. The edges are $E' = \{\{v_s, v'_s\}, \{v_s, \hat{v}'\}, \{v'_s, \hat{v}\}, \{\hat{v}, \hat{v}'\} \mid 1 \leq s \leq \frac{n}{2} - 1\}$



- if $R_t = \{1, 2, \dots, k - 2\}$, then A_k asks for one bit of advice to decide whether to assign color $k - 1$ or k to v_t .

Algorithm A_k asks for an advice bit only when it is about to assign color $k - 1$, which may happen at most every $2^{\frac{k-1}{2}}$ vertices, as shown in the proof of Theorem 1 for algorithm A , so the maximum number of advice bits required is $\frac{n}{\sqrt{2^{k-1}}}$. \square

The proof of Theorem 6 can be easily extended to the case of using a non-constant number of colors, only the size n of the input has to be encoded into the advice string. Since the advice tape is infinite and it is up to the algorithm to decide how many bits to read, we need to encode the value n using a prefix code, such as Elias’ delta-code [6], otherwise the algorithm could not determine where the encoding of n stops and where the actual advice string starts. Therefore, the new advice string will have $\lceil \log_2(n) \rceil + 2\lceil \log_2 \lceil \log_2(n) \rceil + 1 \rceil + 1$ additional bits. This leads to the following corollary.

Corollary 1 *There is an online algorithm for BIPCOL that needs at most $O(\sqrt{n})$ advice bits to color every instance of length n with at most $\lceil \log_2(n) \rceil$ colors.*

In the remainder of this section, we analyze the case of near-optimal coloring using 3 colors. For this case, Theorem 6 gives the following upper bound on the advice complexity.

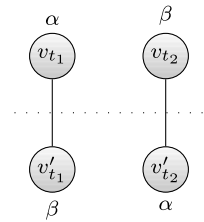
Corollary 2 *There exists an online algorithm for BIPCOL that needs at most $\frac{n}{2}$ advice bits to color every instance of length n with at most 3 colors.*

We conclude with an almost matching lower bound for coloring with 3 colors.

Theorem 7 *Any deterministic online algorithm for BIPCOL needs at least $\frac{n}{2} - 4$ advice bits to color every instance of length n with at most 3 colors.*

Proof Let $n \geq 4$ be even. Consider the graph $G' = (V', E')$ described in Fig. 8. By calling c_t the subgraph of G' induced by the vertices $\{v_t, v'_t\}$, we will consider a set of instances where the first $\frac{n}{2} - 1$ vertices are revealed isolated: each of those can be arbitrarily chosen between the two shores of c_t , for each c_t from left to right. After time step $\frac{n}{2} - 1$, the algorithm will receive the corresponding neighbors in each c_t of all the previously received vertices. Finally, the vertices \hat{v} and \hat{v}' are revealed. By calling $\pi_r(i)$ the vertex revealed at step i , we identify any input instance I_r with the

Fig. 9 If an advice string led to $\text{col}_{\sigma_r}(t_1) = \text{col}_{\sigma_r}^R(t_2)$, two colors would be common



binary string $\sigma_r = (\sigma_r(1), \dots, \sigma_r(\frac{n}{2} - 1))$ such that, for all $1 \leq t \leq \frac{n}{2} - 1$,

$$\sigma_r(t) = \begin{cases} 0 & \text{if } \pi_r(t) = v_t \\ 1 & \text{if } \pi_r(t) = v'_t. \end{cases}$$

In other words, σ_r tells us in which order the two vertices in each c_t are revealed.

With a slight abuse of notation, we say, for $1 \leq t \leq \frac{n}{2} - 1$, that $\text{col}_{\sigma_r}(t) = (\alpha, \beta)$ iff, in I_r , the color assigned to v_t is α and the color assigned to v'_t is β . We also write $\text{col}_{\sigma_r}^R(t) = (\beta, \alpha)$ iff $\text{col}_{\sigma_r}(t) = (\alpha, \beta)$. A color α is common if it appears on both shores of the bipartition, i.e., if there exist $t_1, t_2 \in \{1, \dots, \frac{n}{2} - 1\}$ such that v_{t_1} and v'_{t_2} receive both color α . It is easy to see that an instance of the form considered above can be colored with at most 3 colors only if at most 1 color is common in the first $n - 2$ vertices. As a consequence, we can never have an advice string such that

$$\text{col}_{\sigma_r}(t_1) = \text{col}_{\sigma_r}^R(t_2), \tag{1}$$

for some $t_1, t_2 \in \{1, \dots, \frac{n}{2} - 1\}$, otherwise two colors would be common (see Fig. 9).

In general, if the advice string on an instance σ_i is such that either $(\sigma_i(t_1) = \sigma_i(t_2) \wedge \text{col}_{\sigma_i}(t_1) = \text{col}_{\sigma_i}(t_2))$ or $(\sigma_i(t_1) \neq \sigma_i(t_2) \wedge \text{col}_{\sigma_i}(t_1) = \text{col}_{\sigma_i}^R(t_2))$, then, any other instance σ_j such that $\sigma_i(t_1) = \sigma_j(t_1) \wedge \sigma_i(t_2) \neq \sigma_j(t_2)$ must have a different advice string, otherwise we would have $\text{col}_{\sigma_j}(t_1) = \text{col}_{\sigma_j}^R(t_2)$.

Our aim now is to find out, for a fixed instance σ_i , how many other instances can have the same advice string as σ_i . We can distinguish three situations:

1. The advice string on σ_i is such that the algorithm uses only one pair of colors, i.e.,

$$\forall t : \text{col}_{\sigma_i}(t) = (\alpha, \beta).$$

In this case, the only other instance, which can have the same advice string and still avoids two common colors in the first $n - 2$ vertices is $\bar{\sigma}_i$, such that $\bar{\sigma}_i(t) \neq \sigma_i(t)$, for all $t \in \{1, \dots, \frac{n}{2} - 1\}$, because on any other instance we would have the situation described in (1).

2. The advice string is such that the algorithm uses only two pairs of colors, more formally, there exists a partition $A, B \subset \{1, \dots, \frac{n}{2} - 1\}$, for two nonempty sets A, B such that

$$\forall t \in A : \text{col}_{\sigma_i}(t) = (\alpha, \beta),$$

$$\forall t \in B : \text{col}_{\sigma_i}(t) = (\alpha, \gamma).$$

Table 1 The possible instances using the same advice string as σ_i in the second situation in the proof of Theorem 7

| | σ_i | $\bar{\sigma}_i$ | σ_j | $\bar{\sigma}_j$ |
|-----------|------------|------------------|----------------|------------------|
| $t \in A$ | σ_i | $1 - \sigma_i$ | σ_j | $1 - \sigma_j$ |
| $t \in B$ | σ_i | $1 - \sigma_i$ | $1 - \sigma_j$ | σ_j |

Table 2 The possible instances using the same advice string as σ_i in the third situation in the proof of Theorem 7

| | σ_i | $\bar{\sigma}_i$ | σ_j | $\bar{\sigma}_j$ | σ_k | $\bar{\sigma}_k$ | σ_h | $\bar{\sigma}_h$ |
|-----------|------------|------------------|----------------|------------------|----------------|------------------|----------------|------------------|
| $t \in A$ | σ_i | $1 - \sigma_i$ | σ_j | $1 - \sigma_j$ | σ_k | $1 - \sigma_k$ | σ_h | $1 - \sigma_h$ |
| $t \in B$ | σ_i | $1 - \sigma_i$ | σ_j | $1 - \sigma_j$ | $1 - \sigma_k$ | σ_k | $1 - \sigma_h$ | σ_h |
| $t \in C$ | σ_i | $1 - \sigma_i$ | $1 - \sigma_j$ | σ_j | $1 - \sigma_k$ | σ_k | σ_h | $1 - \sigma_h$ |

In order to avoid (1), the only instances σ that can have the same advice string as σ_i are the ones such that $\sigma(t_1) = \sigma(t_2)$ for any t_1, t_2 in the same set of the partition $\{A, B\}$, which are the four described in Table 1.

- The advice string is such that the algorithm uses all three pairs of colors, i.e., there exists a partition $A, B, C \subset \{1, \dots, \frac{n}{2} - 1\}$, with $A, B, C \neq \emptyset$, such that

$$\begin{aligned} \forall t \in A : \text{col}_{\sigma_i}(t) &= (\alpha, \beta), \\ \forall t \in B : \text{col}_{\sigma_i}(t) &= (\alpha, \gamma), \\ \forall t \in C : \text{col}_{\sigma_i}(t) &= (\beta, \gamma). \end{aligned}$$

Again, in order to avoid (1), an instance σ with the same advice string as σ_i must be such that $\sigma(t_1) = \sigma(t_2)$ for any t_1, t_2 in the same set of the partition $\{A, B, C\}$. This property is satisfied only by the eight instances described in Table 2.

However, the two instances σ_h and $\bar{\sigma}_h$ would have all colors common, if they were given the same advice string as σ_i . This implies that at most six instances of the form σ_r can have the same advice string, and since the number of instances of the form σ_r is $2^{\frac{n}{2}-1}$, there must be at least $\frac{2^{\frac{n}{2}-1}}{6} > 2^{\frac{n}{2}-4}$ different advice strings. □

Acknowledgements The authors would like to thank the anonymous referees for helpful suggestions. The research was partially funded by the SNF grant 200021–141089.

References

- Bean, D.R.: Effective coloration. *J. Symb. Log.* **41**(2), 469–480 (1976)
- Böckenhauer, H.-J., Komm, D., Kráľovič, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: Proc. of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012). LNCS, vol. 7256, pp. 61–72. Springer, Berlin (2012)
- Böckenhauer, H.-J., Komm, D., Kráľovič, R., Kráľovič, R., Mömke, T.: On the advice complexity of online problems. In: Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009). LNCS, vol. 5878, pp. 331–340. Springer, Berlin (2009)

4. Borodin, A., El-Yaniv, R.: *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge (1998)
5. Dobrev, S., Kráľovič, R., Pardubská, D.: How much information about the future is needed? In: *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*. LNCS, vol. 4910, pp. 247–258. Springer, Berlin (2008)
6. Elias, P.: Universal codeword sets and representations of the integers. *IEEE Trans. Inf. Theory* **21**(2), 194–203 (1975)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: *Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*. LNCS, vol. 5555, pp. 427–438. Springer, Berlin (2009)
8. Finch, S.R.: *Mathematical Constants (Encyclopedia of Mathematics and Its Applications)*. Cambridge University Press, New York (2003)
9. Forišek, M., Keller, L., Steinová, M.: Advice complexity of online coloring for paths. In: *Proc. of the 6th International Conference on Language and Automata Theory and Applications (LATA 2012)*. LNCS, vol. 7183, pp. 228–239. Springer, Berlin (2012)
10. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. *J. Graph Theory* **12**(2), 217–227 (1988)
11. Hromkovič, J., Kráľovič, R., Kráľovič, R.: Information complexity of online problems. In: *Proc. of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*. LNCS, vol. 6281, pp. 24–36. Springer, Berlin (2010)
12. Kierstead, H.A.: Recursive and on-line graph coloring. In: Ershov, Y.L., Goncharov, S.S., Nerode, A., Remmel, J.B., Marek, V.W. (eds.) *Handbook of Recursive Mathematics, Vol. 2: Recursive Algebra, Analysis and Combinatorics*. Studies in Logic and the Foundations of Mathematics, vol. 139, pp. 1233–1269. Elsevier, Amsterdam (1998)
13. Kierstead, H.A., Trotter, W.T.: On-line graph coloring. In: McGeoch, L.A., Sleator, D.D. (eds.) *On-Line Algorithms*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 7, pp. 85–92 (1992). AMS/DIMACS/ACM
14. Komm, D., Kráľovič, R.: Advice complexity and barely random algorithms. *RAIRO ITA* **45**(2), 249–267 (2011)
15. Lovász, L., Saks, M.E., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Math.* **75**(1–3), 319–325 (1989)
16. Sloane, N.J.A.: Sequence A000073 in the on-line encyclopedia of integer sequences. Published electronically at <http://oeis.org/A000073> (2012)
17. Vishwanathan, S.: Randomized online graph coloring. *J. Algorithms* **13**(4), 657–669 (1992)