Online Coloring of Bipartite Graphs With and Without Advice

Maria Paola Bianchi[1,2], Hans-Joachim Böckenhauer[2], Juraj Hromkovič[2], and Lucia Keller[2]

[1] Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy
[2] Department of Computer Science, ETH Zurich, Switzerland,
maria.bianchi@unimi.it, {hjb,juraj.hromkovic,lucia.keller}@inf.ethz.ch

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 $1.13747 \cdot \log_2 n$ colors are necessary for any deterministic online algorithm to color any bipartite graph on n vertices, thus improving on the previously known lower bound of $\log_2 n + 1$ 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 graph on n vertices with at most $\lceil \log_2 n \rceil$ colors.

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 [3]. 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 immediately after its appearance. 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 [7]. For an overview of results on the online graph coloring problem, see, e. g., [9, 10]. In particular, some bounds on the

chromatic number of the class $\Gamma(k,n)$ of k-colorable graphs on n vertices have been proven: given $G \in \Gamma(k,n)$, any online coloring algorithm for G needs at least $\Omega\left(((\log n)/(4k))^{k-1}\right)$ colors [13]. On the other hand, there exists an online algorithm for coloring G with $O\left(n\frac{\log^{(2k-3)} n}{\log^{(2k-4)} n}\right)$ colors [11], where $\log^{(k)}$ 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 $\lceil \log_2 n \rceil + 1$ colors for coloring a bipartite graph on n vertices [7]. On the other hand, an online algorithm coloring every bipartite graph with at most $2\log_2 n$ colors is known [11]. In the first part of this paper, we improve the lower bound for bipartite graphs to $1.13747 \cdot \log_2 n$.

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 [2, 4, 5, 8]. For this, we analyze online algorithms that have access to a 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 amount of advice needed by the best algorithm solving the problem.

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, a single bit of advice is sufficient to jump from being non-competitive at all to 2-competitiveness [1]. On the other hand, many problems require a linear (or even higher) amount of advice bits for computing an optimal solution [1, 2, 5].

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 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 Section 2 we formally define the online coloring problem and fix our notation, in Section 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 Section 4, while Section 5 is devoted to the advice complexity

of the online coloring of bipartite graphs. Due to space limitations, some proofs are moved to the appendix.

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

Usually, we consider the set $S = \{1, 2, \ldots, n\} \subset \mathbb{N}^+$. Let $V' \subseteq V$, then we denote by $\mathrm{col}(V')$ 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 $E' = \{\{v, w\} \in E \mid v, w \in V'\}$.
An instance I for the online coloring problem can be described as a sequence $I = (G_1, G_2, \ldots, G_n)$ of undirected graphs such that $V_i = \{v_1, v_2, \ldots, v_i\}$ and $G_i = G_n[V_i]$. Informally speaking, G_i is derived from G_{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 2 (Online Coloring Problem).

Input: *$I = (G_1, G_2, \ldots, G_n) \in \mathcal{G}$ for some $n \in \mathbb{N}^+$.*
Output: *$(c_1, c_2, \ldots, c_n) \in \mathbb{N}^+$ such that $col(v_i) = c_i$ and $col : V_i \to \mathbb{N}^+$ is a coloring for all $i \in \{1, 2, \ldots, n\}$*
Cost: *Number of colors used by the coloring.*
Goal: *Minimum.*

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 $V(G)$, $E(G)$, $S_1(G)$, $S_2(G)$. In the case of connected components, the shore partition is unique.
Given two vertices v_i and v_j, we write $v_i \leftrightarrow_t v_j$ iff there exists a path in G_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 $C_t(v_i) = [v_i]_{\leftrightarrow_t}$.

We want to analyze BipCol with 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_n. Let A be an online coloring algorithm. We denote by $F_A(G) = |\mathrm{col}_A(V_n)|$ the number of colors used by A to color graph G. Then, $F_A(n) = \max_{G \in \mathcal{G}_n} F_A(G)$ is the maximum number of colors A uses to color any online graph instance with n vertices in the final graph G_n. We say that $U : \mathbb{N} \to \mathbb{N}$ is an *upper bound* on the number of colors sufficient for online coloring, if there exists an algorithm A such that, for all $n \in \mathbb{N}$, we have $F_A(n) \leq U(n)$. Hence, it is sufficient to find a good deterministic online algorithm A to get an upper bound on the number of colors used to color an instance of $\mathcal{G}_n$. Similarly, $L : \mathbb{N} \to \mathbb{N}$ is a *lower bound* on the number of colors necessary for online coloring any graph if, for all online algorithms A and infinitely many $n \in \mathbb{N}$, we have $L(n) \leq F_A(n)$, i. e., if, for infinitely many n and for every algorithm A, there exists an online graph $G_A(n) \in \mathcal{G}_n$ for which A needs at least $L(n)$ colors.

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 [11]). *There is an online algorithm requiring at most $2\log_2 n$ colors for coloring any bipartite graph of n vertices.*

There is also a well-known lower bound on the number of needed colors which even holds for trees.

Theorem 2 (Gyárfás and Lehel [7]). *For every $k \in \mathbb{N}^+$, there exists a tree T_k on 2^{k-1} vertices such that for every online coloring algorithm A, $\mathrm{col}_A(T_k) \geq k$.*

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. This class is built recursively. In the proof, we will focus only on those G_i's in an instance $(G_1, G_2, \ldots, G_n) \in \mathcal{G}_n$ in which one new color has to be used by any deterministic online coloring algorithm to color v_i.

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. $|V(G_A(k))| =: B(k) \leq B(k-1) + B(k-2) + B(k-3) + 1$, *for* $k \geq 3$, *and* $B(0) = 0$, $B(1) = 1$, *and* $B(2) = 2$.

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

Corollary 1. $B(k) = 1.35526 \cdot 1.83929^k - .400612$

Proof (Sketch). One can show by induction that $B(k) = \sum_{i=0}^{k+1} T(i)$ where $T(i)$ is the i-th *Tribonacci number* (see [6, 12]). The number $T(n)$ can be computed as follows:

$$T(n) = 3b \cdot \frac{\left(\frac{1}{3}(a_+ + a_- + 1)\right)^n}{b^2 - 2b + 4} = .336228 \cdot 1.83929^n,$$

where $a_+ = (19 + 3\sqrt{33})^{\frac{1}{3}}$, $a_- = (19 - 3\sqrt{33})^{\frac{1}{3}}$, and $b = (586 + 102\sqrt{33})^{\frac{1}{3}}$. □

Theorem 3. *The lower bound on the number of colors used by any online coloring algorithm A to color any online graph instance with n vertices is*

$$1.13747 \cdot log_2(n).$$

Proof. The claim follows immediately from Corollary 1. □

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

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

Hence, for a fixed k, we will find in $\tilde{\mathcal{G}}(k)$, for every online coloring algorithm B, an instance 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 graphs $G_{k-1} \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2} \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3} \in \tilde{\mathcal{G}}(k-3)$, given in this order, and an additional vertex v. Let $H(k)$ be the induction hypothesis, formulated as follows:

H(k): For all $j \leq k$ and all online algorithms B, there exists a graph $G_B(j)$ with the properties 1 to 4 of Lemma 1.

Assuming $H(k-1)$ holds, it is easy to show that a graph G_{k-1} exists. To show the existence of G_{k-2} and G_{k-3} we have to take into account that the algorithm A already knows a part of the instance, and hence it behaves differently from the case where there is no part known.
In a second step, we merge the shores of G_{k-1}, G_{k-2}, and G_{k-3} in an appropriate way and, with an additional vertex v, we assure that all conditions of Lemma 1 are satisfied. We can stop the procedure also earlier, as soon as all four conditions are satisfied.
We merge two graph instances $G = (G_1, \ldots, G_l) \in \mathcal{G}$ and $\tilde{G} = (\tilde{G}_1, \ldots, \tilde{G}_m) \in \mathcal{G}$ to an instance $M = G \circ \tilde{G}$, defined as $M = (G_1, \ldots, G_l, \tilde{G}_1, \ldots, \tilde{G}_m) \in \mathcal{G}$.

(a) Case $k = 1$. (b) Case $k = 2$.

Fig. 1. Base cases: W.l.o.g., the vertices are colored as indicated. The indices of the vertices indicate their order of appearance.

4.1 Base Cases ($k \leq 3$)

For $k = 0, 1, 2$, it is easy to show that the hypothesis is satisfied (see Figure 1). In the 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 are colored w.l.o.g. with 1 and 2, and there are three possibilities to color G_1 (see Figure 2).
In the case (c) of Figure 2, we are already done, since we reached 3 colors and $S_1(G_A(3)) = \{v_1\}$ is colored with one color and $S_1(G_A(3)) = \{v_2, v_3\}$ with two colors. In cases (a) and (b), we have 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. In case (a) (equivalently, case (b)), we have $S_1(G_A(3)) = \{v_1, v_4\}$ and $S_2(G_A(3)) = \{v_2, v_3\}$. Both shores are colored with two colors, hence conditions 1, 2 and 3 are satisfied, and also condition 4, since $B(3) \leq B(2) + B(1) + 1$.

4.2 Inductive Step ($k \geq 4$)

For every online algorithm A and every $k \in \mathbb{N}^+$, we will construct $G_A(k)$ in four steps using three graphs $G_{k-1} \in \tilde{\mathcal{G}}(k-1)$, $G_{k-2} \in \tilde{\mathcal{G}}(k-2)$, $G_{k-3} \in \tilde{\mathcal{G}}(k-3)$ satisfying $H(k-1)$, $H(k-2)$, resp. $H(k-3)$, and an additional vertex v.
First, we want to show that such graphs G_{k-1}, G_{k-2}, and G_{k-3} actually exist. Then, we will show that we can merge them, using an additional vertex v, to a graph $G_A(k)$ satisfying properties 1.-4. from Lemma 1.

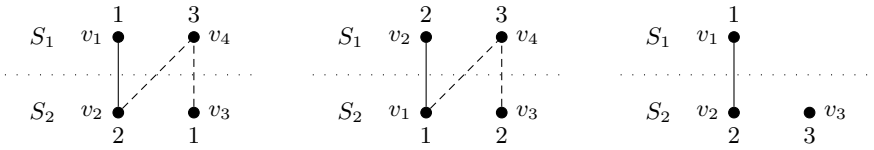

Fig. 2. The base case $k = 3$.

Existence of the graphs G_{k-1}, G_{k-2}, and G_{k-3} 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 $G_B(j)$ with the properties 1.-4. of Lemma 1.

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

Step 2 In the next phase, algorithm A receives a second graph. We denote by $A|_{G_{k-1}}$ the work of algorithm A having already processed the graph G_{k-1}. $A|_{G_{k-1}}$ can be simulated by an algorithm B which does the same work as $A|_{G_{k-1}}$ but which did not receive any other graph before. Because of $H(k-1)$, and thus $H(k-2)$, we know that, for such algorithm B, there is a graph $G_{k-2} = G_B(k-2) = G_{A|_{G_{k-1}}}(k-2)$ satisfiying the properties 1.-4. of Lemma 1.

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

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

Construction of graph $G_A(k)$ This is only a proof sketch. A detailed proof can be found in Appendix A. There are two possible cases with respect to the first graph G_{k-1}:

A) A **uses exactly** $k-1$ **colors on** G_{k-1}
Here, we will distinguish five cases with respect to the colors that appear in both shores in G_{k-1} and the placement of the remaining colors. For sure, we have at least $k-4$ common colors, since otherwise properties 2 and 3 of $H(k-1)$ would not be satisfied.
If we have $k-1$ or $k-2$ common colors, one additional vertex v forces the algorithm to use color k and immediately all properties of $H(k)$ are satisfied.
With $k-3$ common colors, we have to distinguish again two cases: Either the remaining colors a and b are both in one shore or they are in different shores. In the first case, we can proceed as before. In the second case, it depends on whether one of the colors a and b appears in G_{k-2} or not. If there is neither a nor b, there will be a new color c for sure.
If there are $k-4$ common colors, we can either satisfy the conditions of $H(k)$ using some new colors that appear only in G_{k-2} and G_{k-3}, or we find some of the colors that appear only in one shore of G_{k-1} in G_{k-2} and in G_{k-3}. Joining the shores in an appropriate way, we can force with a new vertex a new color k.

B) A **uses at least** k **colors on** G_{k-1}
The graph G_{k-1} already contains sufficiently many colors to satisfy condition 1 of $H(k)$. In general, the graph G_{k-1} has the following colors:

$\mathrm{col}(S_1(G_{k-1}))$	$1, 2, \ldots, x, \qquad\qquad b_1, b_2, \ldots, b_j$
$\mathrm{col}(S_2(G_{k-1}))$	$1, 2, \ldots, x, a_1, a_2, \ldots, a_i$

Because of $H(k-1)$, we have $x+j \geq k-3$ and $x+i \geq k-2$. Hence, we have either to add two colors to $S_1(G_{k-1})$ or one color to each shore in order to satisfy conditions 2 and 3 of $H(k)$. If l is the total number of colors in G_{k-1}, i. e., $x+i+j=l$, and m is the total number of colors in all three graphs G_{k-1}, G_{k-2}, and G_{k-3}, then we have to distinguish some cases with respect to the relation between m and l. If $m \geq l+2$, we have sufficiently many new colors in G_{k-2} or G_{k-3} to satisfy conditions 2 and 3 in $H(k)$. If $m = l+1$, we have one new color c in one of the two smaller graphs, and either we have a second new color or we can find one of the colors $a_1, \ldots, a_i, b_1, \ldots, b_j$ in either G_{k-2} or G_{k-3}. This depends on the relation of k and x. In the case $m = l$, we have to fill the gaps with some of the colors $a_1, \ldots, a_i, b_1, \ldots, b_j$ which again can be found in G_{k-2} and G_{k-3}, depending on the relation between x and k.

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 needs at most* $n-2$ *advice bits to be optimal on every instance of length* n.

Proof. 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 any other 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, the advice is a string of length one; in all other cases it is smaller than $n-2$ bits. □

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

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 (Sketch). A complete proof of this result can be found in Appendix B. Let $\hat{A}$ be an algorithm with advice for BipCol that uses 2 colors. Given, for any $0 \leq \alpha \leq n-2$, the graph G_α with n vertices described in Figure 3, 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 $\{v_j\}_{1 \leq j \leq n-2}$. We say that two instances are equivalent iff the order of the first $n-2$ vertices reflects the same shore partition, and, by a counting argument, the number of such equivalence classes is $\frac{2^{n-2}}{2} = 2^{n-3}$. It is not hard to see that, in order to avoid using a third color on $\hat{v}$ or $\hat{v}'$, the algorithm needs to assign the same color to all vertices of each shore, so instances in different equivalence classes must have different advice strings. □

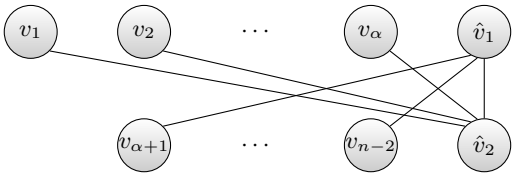

Fig. 3. Structure of the graph G_α used in the proof of Theorem 5. The set of edges is $E = \{\{v_s, \hat{v}_2\} \mid 1 \leq s \leq \alpha\} \cup \{\{v_s, \hat{v}_1\} \mid \alpha + 1 \leq s \leq n-2\} \cup \{\{\hat{v}_1, \hat{v}_2\}\}$

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

Theorem 6. *For any constant $k \in \mathbb{N}$, 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 the same algorithm A used in [10] to prove Theorem 1: the idea is to make A ask for an advice 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. Suppose $v_t \in S_1(C_t(v_t))$ and call $R_t = \mathrm{col}_{A_k}(S_2(C_t(v_t)))$ the set of colors assigned to vertices in the shore opposite to v_t. Then, A_k will choose its output as follows:

- if R_t does not contain all the colors smaller than $k-1$, then $\mathrm{col}_{A_k}(v_t) = \min\{c \geq 1 \mid c \notin R_t\}$,
- if $k-1 \in R_t$, then $\mathrm{col}_{A_k}(v_t) = k$,
- if $k \in R_t$, then $\mathrm{col}_{A_k}(v_t) = k-1$,
- if $R_t = \{1, 2, \ldots, k-2\}$, 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 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 [10] for algorithm A, so the maximum number of advice bits required is $\frac{n}{\sqrt{2^{k-1}}}$. □

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, using $2\lceil \log \log(n) \rceil$ additional bits. This leads to the following corollary.

Corollary 2. *There exists an online algorithm for* BipCol *that needs at most $O(\sqrt{n})$ advice bits to color every instance of length n with at most $\lceil \log(n) \rceil$ colors.*

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

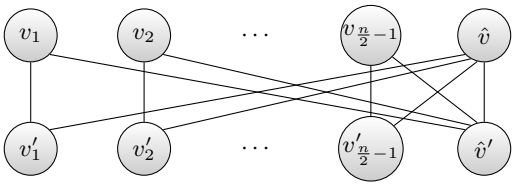

Fig. 4. Graph G' used in the proof of Theorem 7. The edges are $E' = \{\{v_s, v'_s\}, \{v_s, \hat{v}'\}, \{v'_s, \hat{v}\}, \{\hat{v}, \hat{v}'\} \mid 1 \leq s \leq \frac{n}{2} - 1\}$

Corollary 3. *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. Consider the graph $G' = (V', E')$ described in Figure 4. 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 all the c_t's from left to right. After time $\frac{n}{2} - 1$, the algorithm will receive the corresponding neighbors in each c_t of all the previously received vertices. Finally, vertices $\hat{v}$ and $\hat{v}'$ are revealed. By calling $\pi_r(i)$ the vertex revealed at step i, and $S_1(G')$ the shore containing $\hat{v}$, we associate to any input instance I_r the binary string $\sigma_r = (\sigma_r(1), \ldots, \sigma_r(\frac{n}{2} - 1))$ 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 that $\mathrm{col}_{\sigma_r}(t) = (\alpha, \beta)$ iff, in I_r, the color assigned to v_t is α and the color assigned to v'_t is β. We also write $\mathrm{col}^R_{\sigma_r}(t) = (\beta, \alpha)$ iff $\mathrm{col}_{\sigma_r}(t) = (\alpha, \beta)$.
We say that a color α is *mixed* if it appears on both shores of the bipartition, i. e., if there exist $t_1, t_2 \in \{1, \ldots, \frac{n}{2} - 1\}$ 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 mixed in the first $n - 2$ vertices. As a consequence, we can never have an advice such that

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

for some $t_1, t_2 \in \{1, \ldots, \frac{n}{2} - 1\}$, otherwise two colors would be mixed.
In general, if the advice on an instance σ_i is such that either $\sigma_i(t_1) = \sigma_i(t_2) \ \wedge \ \mathrm{col}_{\sigma_i}(t_1) = \mathrm{col}_{\sigma_i}(t_2)$ or $\sigma_i(t_1) \neq \sigma_i(t_2) \ \wedge \ \mathrm{col}_{\sigma_i}(t_1) = \mathrm{col}^R_{\sigma_i}(t_2)$, then, for any other instance σ_j such that $\sigma_i(t_1) = \sigma_j(t_1) \ \wedge \sigma_i(t_2) \neq \sigma_j(t_2)$, σ_j must have a different advice, otherwise we would have $\mathrm{col}_{\sigma_j}(t_1) = \mathrm{col}^R_{\sigma_j}(t_2)$.

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

Table 1. The possible instances using the same advice as σ_i in the second 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_i	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$
$t \in B$	σ_i	$1-\sigma_i$	σ_i	$1-\sigma_i$	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i
$t \in C$	σ_i	$1-\sigma_i$	$1-\sigma_i$	σ_i	$1-\sigma_i$	σ_i	σ_i	$1-\sigma_i$

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

Our aim now is to find out how many instances can have the same advice as σ_i. We can distinguish three situations:

1. The advice on σ_i is such that the algorithm uses only one couple of colors, i. e., for every t, $\mathrm{col}_{\sigma_i}(t) = (\alpha, \beta)$. In this case, the only other instance which can have the same advice and still avoids mixing two colors in the first $n-2$ vertices is $\bar{\sigma}_i$, such that $\bar{\sigma}_i(t) \neq \sigma_i(t)$, for all $t \in \{1, \ldots, \frac{n}{2} - 1\}$, because on any other instance we would have the situation described in Equation (1).
2. The advice is such that the algorithm uses only two couples of colors, more formally, there exists a partition $A, B \subset \{1, \ldots, \frac{n}{2} - 1\}$, for two nonempty sets A, B such that

   $$\forall t \in A\colon \mathrm{col}_{\sigma_i}(t) = (\alpha, \beta), \qquad \forall t \in B\colon \mathrm{col}_{\sigma_i}(t) = (\alpha, \gamma).$$

   In order to avoid Equation (1), the only instances σ that can have the same advice 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.
3. The advice is such that the algorithm uses all three couples of colors, i. e., there exists a partition $A, B, C \subset \{1, \ldots, \frac{n}{2} - 1\}$, with $A, B, C \neq \emptyset$, such that

   $$\forall t \in A\colon \mathrm{col}_{\sigma_i}(t) = (\alpha, \beta), \quad \forall t \in B\colon \mathrm{col}_{\sigma_i}(t) = (\alpha, \gamma), \quad \forall t \in C\colon \mathrm{col}_{\sigma_i}(t) = (\beta, \gamma).$$

   Again, in order to avoid Equation (1), an instance σ with the same advice 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 mixed, if they were given the same advice as σ_i. This implies that at most six instances of the form σ_r can have the same advice, 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. □

References

1. H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. On the advice complexity of the knapsack problem. *Proc. of the 10th Latin American Symposium on Theoretical Informatics (LATIN 2012)*. to appear.
2. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. *Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, LNCS 5878, Springer-Verlag, 2009, pp. 331–340.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
4. S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, LNCS 4910, Springer-Verlag, 2008, pp. 247–258.
5. Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, LNCS 5555, Springer-Verlag, 2009, pp. 427–438.
6. S. R. Finch. *Mathematical Constants (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, New York, NY, USA, 2003.
7. A. Gyárfás and J. Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988.
8. J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In *Proc. of the 35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*, LNCS 6281, Springer-Verlag, 2010, pp. 24–36.
9. H.A. Kierstead. Recursive and on-line graph coloring. In Yu. L. Ershov, S.S. Goncharov, A. Nerode, J.B. Remmel, and V.W. Marek, editors, *Handbook of Recursive Mathematics Volume 2: Recursive Algebra, Analysis and Combinatorics*, volume 139 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, 1998, pp. 1233–1269.
10. H.A. Kierstead and W.T. Trotter. On-line graph coloring. In Lyle A. McGeoch and Daniel D. Sleator (eds.) *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS—DIMACS—ACM, 1992, pp. 85–92.
11. L. Lovász, M. E. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1–3):319–325, 1989.
12. N.J.A. Sloane. Sequence A000073 in *The On-Line Encyclopedia of Integer Sequences*. Published electronically at http://oeis.org/A000073, 2012.
13. S. Vishwanathan. Randomized online graph coloring. *Journal of Algorithms*, 13(4):657–669, 1992.

A Proof of Lemma 1: Construction of graph $G_A(k)$

The graphs G_{k-1}, G_{k-2}, and G_{k-3} are constructed in this order. Beginning with graph G_{k-1} we distinguish two possible cases:

A) A uses exactly $k-1$ colors to color G_{k-1}
B) A uses at least k colors to color G_{k-1}

In case B), we have already sufficiently many colors for G_k, but we have to assure that the properties 2, 3 and 4 of $H(k)$ are satisfied.
In case A), we have to add, with the help of graphs G_{k-2} and G_{k-3} and with the single vertex v, a new color k, and of course, we have also assure that we satisfy conditions 2 and 3 of $H(k)$.
We will show that in both cases, A) and B), either the graphs G_{k-2} and G_{k-3} contain some of the colors that appear only in one shore of G_{k-1}, or there are enough additional colors to fill the gaps in order to satisfy $H(k)$. Only in some cases, we will need all three graphs G_{k-1}, G_{k-2}, G_{k-3} and the vertex v.

A) A uses exactly $k-1$ colors on G_{k-1}
We will distinguish several cases with respect to the number of colors that appear in both shores of G_{k-1}. Either, all colors appear in both shores. Or, we can have either only $k-1$, only $k-2$, only $k-3$, or only $k-4$ colors that appear in both shores. If only $k-5$ colors would appear in both shores then we would have only 4 colors left to color the rest of the graph. Each of those 4 colors can appear either in $S_1(G_{k-1})$ or in $S_2(G_{k-1})$ but not in both shores. But, we have five gaps to fill:

$\mathrm{col}(S_1(G_{k-1}))$	1, 2, …, $k-5$ ␣ ␣
$\mathrm{col}(S_2(G_{k-1}))$	1, 2, …, $k-5$ ␣ ␣ ␣

Hence, no matter how we distribute those 4 colors (in $S_1(G_{k-1})$ 2 colors are missing, and in $S_2(G_{k-1})$ 3 colors are missing), we cannot guarantee that $S_1(G_{k-1})$ contains $k-3$ colors and $S_2(G_{k-1})$ $k-2$ colors. Regarding this, we have 5 possibilities for a graph coloring for graph G_{k-1} that satisfy $H(k-1)$:

1. $k-1$ **colors are in both shores:**
   W.l.o.g., we assume that there are the colors $1, 2, \ldots, k-1$:

$\mathrm{col}(S_1(G_{k-1}))$	1, 2, …, $k-1$
$\mathrm{col}(S_2(G_{k-1}))$	1, 2, …, $k-1$

   Both shores already satisfy the conditions 2 and 3 of $H(k)$. But, we need an additional color k. We get this color by connecting a new vertex v to all vertices in $S_1(G_{k-1})$. This vertex has to get a new color, say w.l.o.g. k:

$\mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1}))$	1, 2, …, $k-1$
$\mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(v)$	1, 2, …, $k-1$, k

   Hence, with

   $$B(k) = B(k-1) + 1$$

   all properties of $H(k)$ are satisfied.

2. $k-2$ **colors are in both shores:**
   The remaining color, say a, appears w.l.o.g. in $S_2(G_{k-1})$ (if it would appear in $S_1(G_{k-1})$, we could simply swap the two shores):
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) & 1,\, 2,\, \ldots,\, k-2 \\ \hline \mathrm{col}(S_2(G_{k-1})) & 1,\, 2,\, \ldots,\, k-2,\, a \end{array}$$
   $S_1(G_{k-1})$ contains $k-2$ colors and $S_2(G_{k-1})$ is colored with $k-1$ colors. Hence, conditions 2 and 3 of $H(k)$ are satisfied. Only one new color is missing and we get this color by adding one vertex v to graph G_{k-1} which is connected to all vertices in $S_2(G_{k-1})$. This vertex has to get a new color, say color k:
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_k)) \cup \mathrm{col}(v) & 1,\, 2,\, \ldots,\, k-2,\quad k \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_k)) & 1,\, 2,\, \ldots,\, k-2,\, a \end{array}$$
   And we get again
   $$B(k) = B(k-1) + 1\,.$$

3. $k-3$ **colors appear in both shores plus two additional colors** $a, b \in$ **col**$(S_2(G_{k-1}))$**:**
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) & 1,\, 2,\, \ldots,\, k-3 \\ \hline \mathrm{col}(S_2(G_{k-1})) & 1,\, 2,\, \ldots,\, k-3,\, a,\, b \end{array}$$
   $S_1(G_{k-1})$ contains $k-3$ colors and $S_2(G_{k-1})$ contains $k-1$ colors. Hence, we need an additional color in $S_1(G_{k-1})$. Again, one vertex v that is connected to all vertices in $S_2(G_{k-1})$ is sufficient to generate a new color k:
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(v) & 1,\, 2,\, \ldots,\, k-3,\qquad k \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) & 1,\, 2,\, \ldots,\, k-3,\, a,\, b \end{array}$$
   We get again
   $$B(k) = B(k-1) + 1\,.$$

4. **Again, both shores contain** $k-3$ **colors, but color** a **is w.l.o.g. in** $S_2(G_{k-1})$ **and** $b \in$ **col**$(S_1(G_{k-1}))$**:**
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) & 1,\, 2,\, \ldots,\, k-3,\quad b \\ \hline \mathrm{col}(S_2(G_{k-1})) & 1,\, 2,\, \ldots,\, k-3,\, a \end{array}$$
   We have to add one color to one of the two shores since both shores in $G(k-1)$ contain only $k-2$ colors. G_{k-2} is colored either with exacly $k-2$ or with more than $k-2$ colors.

   4.1 G_{k-2} **contains either color** a **or** b
   W.l.o.g., b appears in G_{k-2} in shore $S_1(G_{k-2})$ (otherwise rename all colors such that a and b switch their names and swap $S_1(G_{k-1})$ and $S_2(G_{k-2})$). Join graphs G_{k-1} and G_{k-2} such that $S_2(G_{k-1})$ and $S_1(G_{k-2})$ form one shore[3]:
   $$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-2})) & 1,\, 2,\, \ldots,\, k-3,\quad b \\ \hline \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-2})) & 1,\, 2,\, \ldots,\, k-3,\, a,\, b \end{array}$$

[3] Note that color a might appear in $S_2(G_{k-2})$. But this does not change the following argumentation. Also, in further cases, we will not explicitely mention the subcases where more colors than used appear in the subgraphs.

This graph has already enough colors in both shores. To force color k, we add a vertex v that is connected to all vertices in $S_2(G_{k-1}) \cup S_1(G_{k-2})$:

$$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-2})) \cup \mathrm{col}(v) & 1,\, 2,\, \ldots,\, k-3,\quad b,\, k \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-2})) & 1,\, 2,\, \ldots,\, k-3,\, a,\, b \end{array}$$

And, we have

$$B(k) = B(k-1) + B(k-2) + 1\,.$$

4.2 G_{k-2} **does not contain** a **nor** b

Because G_{k-2} contains at least $k-2$ colors, we know that it contains at least one colors that did not appear in G_{k-1}. Say, w.l.o.g. $c \in S_1(G_{k-2})$ is one of those colors. Then, we can complete G_{k-1} as follows:

$$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-2})) & 1,\, 2,\, \ldots,\, k-3,\quad b \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-2})) & 1,\, 2,\, \ldots,\, k-3,\, a\,,\quad c \end{array}$$

Because we have $k-2$ colors in $S_1(G_k)$ and $k-1$ colors in $S_2(G_k)$ and the total number of colors in G_k is at least k, conditions 1 to 3 of $H(k)$ are satisfied. Additionally,

$$B(k) = B(k-1) + B(k-2)\,.$$

5. $k-4$ **colors appear in both shores:**

W.l.o.g. colors a and b are in $S_2(G_{k-1})$ and color c is in $S_1(G_{k-1})$. The situation with two colors in $S_1(G_{k-1})$ and only one additional color in $S_2(G_{k-1})$ would be equivalent.

$$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) & 1,\, 2,\, \ldots,\, k-4,\quad c \\ \hline \mathrm{col}(S_2(G_{k-1})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b \end{array}$$

We have $k-3$ colors in $S_1(G_{k-1})$ and $k-2$ colors in $S_2(G_{k-1})$. Hence, to satisfy $H(k)$, we have either to add one additional color to each shore, or, we can add two colors to $S_1(G_{k-1})$. Additionally, we have to force a new color k for graph G_k. To force a new color, we either need c in $S_2(G_{k-1})$ or a and b in $S_1(G_{k-1})$. But it might happen, that there is already an additional color in G_{k-2} or in G_{k-3} that does not appear in G_{k-1}. In this case, we do not have to force a new color.

5.1 **There are colors** $d \in \mathbf{col}(G_{k-2})$ **and** $e \in \mathbf{col}(G_{k-3})$ **with** $d, e \notin \mathbf{col}(G_{k-1})$

W.l.o.g. $d \in \mathrm{col}(S_1(G_{k-2}))$ and $e \in \mathrm{col}(S_1(G_{k-3}))$. Then the following combination of the shores satisfies $H(k)$:

$$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-2})) \cup \mathrm{col}(S_2(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\quad c,\, d \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-2})) \cup \mathrm{col}(S_1(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b,\quad e \end{array}$$

The number of colors in $S_1(G_k)$ is larger than $k-2$ and in $S_2(G_k)$ it is larger than $k-1$. The total number of colors is at least $k+1$. And with

$$B(k) = B(k-1) + B(k-2) + B(k-3)$$

all conditions of $H(k)$ are satisfied.

5.2 **$\mathbf{col}(G_{k-2}) \subseteq \mathbf{col}(G_{k-1})$ and there is a new color $e \in \mathbf{col}(G_{k-3}) \backslash \mathbf{col}(G_{k-1})$**
Since G_{k-2} contains at least $k-2$ colors and it is colored with a subset of colors of G_{k-1}, it contains at least one color of the set $\{a, b, c\}$, say w.l.o.g. $a \in S_1(G_{k-2})$ and $e \in \mathrm{col}(S_1(G_{k-3}))$. Then we get

$$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-2})) \cup \mathrm{col}(S_2(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\, a, \quad c \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-2})) \cup \mathrm{col}(S_1(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b, \quad e \end{array}$$

$H(k)$ is satisfied.

5.3 **$\mathbf{col}(G_{k-3}) \subseteq \mathbf{col}(G_{k-1})$ and there is a new color $d \in \mathbf{col}(G_{k-2}) \backslash \mathbf{col}(G_{k-1})$**
With the same argumentation as above, we can say that G_{k-3} must contain at least one of the colors $\{a, b, c\}$. Then, we have the same situation as in the case before.

In the following cases, we can assume that $\mathrm{col}(G_{k-2}) \subseteq \mathrm{col}(G_{k-1})$ and $\mathrm{col}(G_{k-3}) \subseteq \mathrm{col}(G_{k-1})$. Because G_{k-3} is colored with at least $k-3$ colors, it contains one of the three colors $\{a, b, c\}$. Analogously, G_{k-2} contains two of the three colors $\{a, b, c\}$. Then, either c appears in one of the graphs G_{k-2} or G_{k-3}, or one of the colors $\{a, b\}$ is in G_{k-3} and both a and b are in G_{k-2}:

5.4 **W.l.o.g. $S_1(G_{k-3})$ contains color c:**
Color c could also appear in G_{k-2}. Also in this case, the following argumentation would be analogous.

$$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4, \quad c \\ \hline \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b,\, c \end{array}$$

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

$$\begin{array}{l|l} \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-3})) \cup \mathrm{col}(v) & 1,\, 2,\, \ldots,\, k-4, \quad c,\, k \\ \hline \mathrm{col}(S_1(G_k) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-3})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b,\, c \end{array}$$

$H(k)$ is satisfied, because $S_1(G_k)$ contains $k-1$ colors, and $S_2(G_k)$ is colored with at least $k-2$ colors.
And, we have

$$B(k) = B(k-1) + B(k-3) + 1\,.$$

(Or, analogously: $B(k) = B(k-1) + B(k-2) + 1$.)

5.5 **W.l.o.g. $S_1(G_{k-3})$ contains color a and $S_1(G_{k-2})$ contains color b:**

$$\begin{array}{l|l} \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-3})) \cup \mathrm{col}(S_1(G_{k-2})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b,\, c \\ \hline \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-3})) \cup \mathrm{col}(S_2(G_{k-2})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b \end{array}$$

Color c might appear in $S_2(G_{k-2})$. But this case is equivalent to the following one.
We can force an algorithm to use color k by introducing a new vertex v that is connected with all vertices in $S_1(G_{k-1}) \cup S_1(G_{k-3}) \cup S_1(G_{k-2})$:

$$\begin{array}{l|l} \mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1})) \cup \mathrm{col}(S_1(G_{k-3})) \cup \mathrm{col}(S_1(G_{k-2})) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b,\, c \\ \hline \mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1})) \cup \mathrm{col}(S_2(G_{k-3})) \cup \mathrm{col}(S_2(G_{k-2})) \cup \mathrm{col}(v) & 1,\, 2,\, \ldots,\, k-4,\, a,\, b, \quad k \end{array}$$

Now, both shores contain at least $k-1$ colors and hence. It holds, that

$$B(k) = B(k-1) + B(k-2) + B(k-3) + 1\,.$$

Hence, $H(k)$ is satisfied.

B) *A* **uses at least** k **colors on** G_{k-1}

The graph G_{k-1} has already as many colors as are needed for G_k, but we have to assure that the properties 2 and 3 of $H(k)$ are also satisfied. Hence, we have to add at most two colors to G_{k-1}:

1. Either, we can add both colors to $S_1(G_{k-1})$, or
2. we can add one color to shore $S_1(G_{k-1})$ and one to shore $S_2(G_{k-1})$.

In both cases, we get a graph G_k that satisfies both properties of $H(k)$.

Graph G_{k-1} has in general the following colors:

$\mathrm{col}(S_1(G_{k-1}))$	$1, 2, \ldots, x, \qquad\qquad b_1, b_2, \ldots, b_j$
$\mathrm{col}(S_2(G_{k-1}))$	$1, 2, \ldots, x, a_1, a_2, \ldots, a_i$

Let l be the number of all colors in G_{k-1}, i. e., $x+i+j=l$. And, suppose that in all graphs G_{k-1}, G_{k-2} and G_{k-3} we have m colors in total. The colors $1, 2, \ldots, x$ will not help to fill the gaps in G_{k-1}. Hence, we have $m-x$ colors that are useful. Those colors have one of the following two properties:

1. The color is one of the colors $a_1, a_2, \ldots, a_i, b_1, b_2, \ldots, b_j$ that appear already in G_{k-1}.
2. It is a new color $c_1, c_2, \ldots, c_h$

Hence, we have $x+i+j+h=m$.

Now, we make a case distinction with respect to the relation of the total number m of colors and the number l of colors used in G_{k-1}:

Case 1: $m \geq l+2$

In total, in G_{k-2} and in G_{k-3}, we have two new colors c_1 and c_2 that are either in one shore or distributed to two shores. No matter where c_1 or c_2 are, we can either add them both to $S_1(G_{k-1})$ or one of them to $S_1(G_{k-1})$ and the other one to $S_2(G_{k-1})$ to satisfy all conditions of $H(k)$.

Case 2: $m = l+1$

We have one new color c_1 and some of the colors $a_1, a_2, \ldots, a_i, b_1, b_2, \ldots, b_j$ to fill the gaps in G_{k-1}. Now, it depends on x how many colors we need to add:

1. $x \geq k-1$
   $H(k)$ is already satisfied.
2. $x = k-2$:
   The remaining two colors in G_{k-1} fill the gap in $S_2(G_{k-1})$ and hence $H(k)$ is satisfied.
3. $x = k-3$:
   In G_{k-1}, we have at least three colors more. We can encounter one of the following two possible graphs:

$\mathrm{col}(S_1(G_k)) = \mathrm{col}(S_1(G_{k-1}))$	$1, 2, \ldots, k-3, \qquad b_1$
$\mathrm{col}(S_2(G_k)) = \mathrm{col}(S_2(G_{k-1}))$	$1, 2, \ldots, k-3, a_1, a_2$

   or

$$\begin{array}{l|l} \text{col}(S_1(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3 \\ \hline \text{col}(S_2(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3,\ a_1,\ a_2,\ a_3 \end{array}$$

In the first graph, we are already done. In the second graph, it is enough to add one color to $S_1(G_{k-1})$. We need only one new color and hence, we can take color c_1 which has to appear either in G_{k-2} or in G_{k-3}.

4. $x \leq k-4$:
   Because $x \leq k-4$, there are at least 4 colors more (additionally to the set of colors $\{1, 2, \ldots, x\}$) in G_{k-1}, say these are the colors $\{d_1, d_2, d_3, d_4\}$. At least two of the colors $\{d_1, d_2, d_3, d_4, c_1\}$ have to appear again in G_{k-2} and one of those is for sure in G_{k-3} because we have $x \leq k-4$. Hence, we can pick one of those colors out of each graph and add these colors to G_{k-1}. No matter what colors is taken out of $\{d_1, d_2, d_3, d_4\}$, we can fill the second gap with c_1 which appears for sure either in G_{k-2} or in G_{k-3}.

Case 3: $m = l$

We can fill the gaps only with the colors $a_1,\ a_2,\ \ldots,\ a_i,\ b_1,\ b_2,\ \ldots,\ b_j$. Again, in the worst case, we have to add one color to each shore in G_{k-1} or two colors to shore $S_1(G_{k-1})$ in order to satisfy the properties 2 and 3 of $H(k)$.

1. $x \geq k-1$:
   Again, $H(k)$ is satisfied and we do not have to add new colors to G_{k-1}.
2. $x = k-2$:
   The remaining two colors in G_{k-1} fill the gap in $S_2(G_{k-1})$ and hence $H(k)$ is satisfied.
3. $x \leq k-3$:
   We have again two possibilities:
   $$\begin{array}{l|l} \text{col}(S_1(G_k)) = \text{col}(S_1(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3,\qquad b_1 \\ \hline \text{col}(S_2(G_k)) = \text{col}(S_2(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3,\ a_1,\ a_2 \end{array}$$
   or
   $$\begin{array}{l|l} \text{col}(S_1(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3 \\ \hline \text{col}(S_2(G_{k-1})) & 1,\ 2,\ \ldots,\ k-3,\ a_1,\ a_2,\ a_3 \end{array}$$
   The first case is already done. In the second case, we have to add one color to $S_1(G_{k-1})$. Since we have $m = l$, one of the colors $\{a_1, a_2, a_3\}$ has to appear in G_{k-2}.
4. $x \leq k-4$:
   We will encounter the following situation:
   $$\begin{array}{l|l} \text{col}(S_1(G_{k-1})) & 1,\ 2,\ \ldots,\ x,\qquad\qquad b_1,\ b_2,\ \ldots,\ b_j \\ \hline \text{col}(S_2(G_{k-1})) & 1,\ 2,\ \ldots,\ x,\ a_1,\ a_2,\ \ldots,\ a_i \end{array}$$
   Since $x + j = k-3$ in the worst case and G_{k-2} contains at least $k-2$ colors, one of the colors $\{a_1, a_2, \ldots a_i\}$ has to appear in G_{k-2}. We take this color to fill the gap in $S_1(G_{k-1})$. Now, either we add a second color to $S_1(G_{k-1})$ or we add a color to $S_2(G_{k-2})$. Since $x \leq k-4$, we know that G_{k-2} contains at least two colors of $\{a_1, a_2, \ldots, a_i, b_1, b_2, \ldots, b_j\}$. Hence, we can use the second color of G_{k-2} to fill the last gap.

B Proof of Theorem 5

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 Figure 3, 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 $n-2$. Hence, $\hat{A}$ will be able to color $\hat{v}_1$ and $\hat{v}_2$ with values in $\{1,2\}$ only if $v_1, \ldots, v_\alpha$ all have the same color, and $v_{\alpha+1}, \ldots, v_{n-2}$ all have the opposite color.
Since there is a bijection between the instances and all the possible permutations of the first $n-2$ revealed vertices $\pi_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(n-2))$, 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, $\pi_i \sim \pi_j$ iff, for all $t_1, t_2 \leq n-2$

$$\begin{array}{c} \pi_i(t_1), \pi_i(t_2) \in S_1(G) \ \vee \ \pi_i(t_1), \pi_i(t_2) \in S_2(G) \\ \Updownarrow \\ \pi_j(t_1), \pi_j(t_2) \in S_1(G) \ \vee \ \pi_j(t_1), \pi_j(t_2) \in S_2(G) \end{array}$$

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 theorem, it is sufficient to show that $\hat{A}$ needs a different advice for each equivalence class. Suppose, for contradiction, that $\pi_i \nsim \pi_j$ and $\hat{A}$ receives the same advice for both instances. Then, since all instances look the same until time $n-2$, it must hold $\mathrm{col}_{\hat{A}}(\pi_i(t)) = \mathrm{col}_{\hat{A}}(\pi_j(t))$ for all $t < n-2$.
Because the two instances are not equivalent, w.l.o.g. we can say there are two values $t_1, t_2 < n-2$, 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:

- if $\mathrm{col}_{\hat{A}}(\pi_i(t_1)) \neq \mathrm{col}_{\hat{A}}(\pi_i(t_2))$, 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 $\pi_i(t_1)$,
- if $\mathrm{col}_{\hat{A}}(\pi_i(t_1)) = \mathrm{col}_{\hat{A}}(\pi_i(t_2))$, then $\mathrm{col}_{\hat{A}}(\pi_j(t_1)) = \mathrm{col}_{\hat{A}}(\pi_j(t_2))$. W.l.o.g., we can assume that, in the instance associated to π_j, the vertex $\hat{v}_1$ is on the shore opposite to $\pi_j(t_1)$, hence $\{\hat{v}_1, \pi_j(t_1)\}$ is an edge. As a consequence we must have $\mathrm{col}_{\hat{A}}(\hat{v}_1) \neq \mathrm{col}_{\hat{A}}(\pi_j(t_1))$ and therefore $\mathrm{col}_{\hat{A}}(\hat{v}_1) \neq \mathrm{col}_{\hat{A}}(\pi_j(t_2))$, 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 $\hat{v}_2$.

□