On the advice complexity of the online L(2,1)-coloring problem on paths and cycles

On the Advice Complexity of the Online $L(2,1)$-Coloring Problem on Paths and Cycles[☆]

Maria Paola Bianchi[a], Hans-Joachim Böckenhauer[b], Juraj Hromkovič[b], Sacha Krug[b], Björn Steffen[b]

[a] *Dipartimento di Informatica, Università degli Studi di Milano, Italy*
[b] *Department of Computer Science, ETH Zurich, Switzerland*

Abstract

In an $L(2,1)$-coloring of a graph, the vertices are colored with colors from an ordered set such that neighboring vertices get colors that have distance at least 2 and vertices at distance 2 in the graph get different colors. We consider the problem of finding an $L(2,1)$-coloring using a minimum range of colors in an online setting where the vertices arrive in consecutive time steps together with information about their neighbors and vertices at distance 2 among the previously revealed vertices. For this, we restrict our attention to paths and cycles.

Offline, paths can easily be colored within the range $\{0,\ldots,4\}$ of colors. We prove that, considering deterministic algorithms in an online setting, the range $\{0,\ldots,6\}$ is necessary and sufficient while a simple greedy strategy needs range $\{0,\ldots,7\}$.

Advice complexity is a recently developed framework to measure the complexity of online problems. The idea is to measure how many bits of advice about the yet unknown parts of the input an online algorithm needs to compute a solution of a certain quality. We show a sharp threshold on the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. While achieving color range $\{0,\ldots,6\}$ does not need any advice, improving over this requires a number of advice bits that is linear in the size of the input. Thus, the $L(2,1)$-coloring problem is the first known example of an online problem for which sublinear advice does not help.

We further use our advice complexity results to prove that no randomized online algorithm can achieve a better expected competitive ratio than $\frac{5}{4}(1-\delta)$, for any $\delta > 0$.

Keywords: Online coloring, frequency assignment, advice complexity, randomized algorithms

1. Introduction

Graph coloring is a well-known problem with many applications, and many variants of it have been considered in the literature. One of these variations

[☆]This work was partially supported by SNF grant 200021-141089.

Email addresses: maria.bianchi@unimi.it (Maria Paola Bianchi), hjb@inf.ethz.ch (Hans-Joachim Böckenhauer), juraj.hromkovic@inf.ethz.ch (Juraj Hromkovič), sacha.krug@inf.ethz.ch (Sacha Krug), bjoern.steffen@inf.ethz.ch (Björn Steffen)

is motivated by a problem arising in the context of assigning frequencies to transmitters in a multihop radio network. To avoid interference, the difference between the frequencies used by transmitters should be anti-proportional to their proximity, see the survey by Murphey et al. [17] for an overview of frequency assignment problems. The simplest graph-theoretic model of the frequency assignment problem is the $L(2,1)$-coloring problem, which was introduced by Griggs and Yeh [12]. It is also commonly known as the radio coloring problem [8]. Here, the transmitters are the vertices of a graph and the frequencies are modeled by colors from an ordered set, usually $\{0,1,\ldots,\lambda\}$, for some $\lambda \in \mathbb{N}$. For the coloring, two classes of proximity are distinguished. Neighboring vertices have to be assigned colors that are at least 2 apart in their given order, and vertices at distance 2 (called *square neighbors* in the following) have to get different colors. The goal is to find such a coloring of the graph with a minimum λ. This problem has been intensively studied for various graph classes; for a survey, see the papers by Yeh [21] or Bodlaender et al. [2]. In this paper, we introduce and investigate an online version of the problem. Here, the vertices of the graph appear in consecutive time steps together with information about those neighbors and square neighbors that have already been revealed. In each time step, the online algorithm has to irrevocably determine the color of the newly revealed vertex respecting the above restrictions.

The quality of an online solution is traditionally measured using the competitive analysis introduced by Sleator and Tarjan [20]. The *competitive ratio*, defined as the quotient of the cost of the computed online solution and the cost of an optimal (offline) solution, is the standard measure for the quality of an online algorithm. A detailed introduction to the theory of online algorithms can, e. g., be found in the textbook by Borodin and El-Yaniv [7]. The recently introduced framework of *advice complexity* aims at a more fine-grained measurement of the complexity of online problems. The idea is to measure how much information about the future parts of the input is needed to compute an optimal solution or a solution with a specific competitive ratio [13]. This is modeled by an oracle that knows the whole input in advance and prepares some infinite tape of *advice bits* which the online algorithm can use as an additional resource. The *advice complexity* of an online algorithm is then defined as the maximum length of the prefix of the advice tape that the algorithm accesses over all inputs of a fixed length. The advice complexity of an online problem is the minimal advice complexity over all admissible algorithms. The first model of advice complexity was introduced by Dobrev et al. [9] and later refined by Emek et al. [10]. The first model was not exact enough, discrepancies up to a multiplicative factor were possible. The latter model was suitable only if the advice complexity was at least linear. We are using the general model as proposed by Hromkovič et al. [13]. The concept was successfully applied to various online problems [1, 3, 4, 5, 10, 11, 15, 18].

There are many connections between advice complexity and randomized online computations [4, 6, 14]. Obviously, every online algorithm using b advice bits is as least as powerful as a randomized algorithm using the same number of random binary decisions. But under certain conditions, one can use lower bounds on the advice complexity also to prove lower bounds on randomized online computation with an unbounded number of random bits [4].

In this paper, we focus on the online $L(2,1)$-coloring problem on paths, cycles and graphs of maximum degree 2. We first present our results for paths and

later generalize these results to cycles and graphs of maximum degree 2, see Section 5.

While the $L(2,1)$-coloring problem on paths is almost trivial in the offline case (simply coloring the vertices along the path using the pattern 0-2-4 is optimal), it turns out to be surprisingly hard in the online case. We first analyze a simple greedy strategy and show that it uses $\lambda = 7$. Then, we present an improved deterministic online algorithm using $\lambda = 6$ and prove a matching lower bound for deterministic online algorithms without advice. Considering online algorithms with advice, we prove that, in order to achieve an optimal solution on a path on n vertices, $0.6955n + d$ advice bits are sufficient, for some positive constant d, and $0.0519n$ advice bits are necessary. Surprisingly, also to compute a 1.25-competitive solution (i. e., for $\lambda = 5$), a linear number of advice bits is necessary. Thus, the $L(2,1)$-coloring is the first known problem for which sublinear advice does not help at all, not even on the very simple graph classes of graphs with maximum degree 2. Finally, we employ this lower bound to show that no randomized online algorithm for the online $L(2,1)$-coloring problem has an expected competitive ratio of $\frac{5}{4}(1-\delta)$, for any $\delta > 0$. Table 1 summarizes our advice complexity results.

Quality	Lower bound	Upper bound
5/4-competitive	$3.9402 \cdot 10^{-10} \cdot n$	$0.4n + d_1$
optimal	$0.0518n + d_2$	$0.6955n + d_3$ (for paths and cycles) $1.5314n + O(\log n)$ (for graphs with maximum degree 2)

Table 1: Advice complexity of the online $L(2,1)$-coloring problem on paths, cycles, and graphs with maximum degree 2, where d_1, d_2, and d_3 are positive constants. (1.5-competitiveness can be achieved without advice.)

2. Preliminaries

Let us first formally define the framework we are using.

Definition 1. Consider an input sequence $I = (x_1, \ldots, x_n)$ for some minimization problem U. An *online algorithm* A computes the output sequence $A(I) = (y_1, \ldots, y_n)$, where $y_i = f(x_1, \ldots, x_i)$, for some function f. The *cost* of the solution is denoted by $\mathrm{cost}(A(I))$. An algorithm A is *strictly c-competitive*, for some $c \geq 1$, if, for every input sequence I, $\mathrm{cost}(A(I)) \leq c \cdot \mathrm{cost}(\mathrm{Opt}(I))$, where Opt is an optimal offline algorithm for the problem. A is *optimal* if it is strictly 1-competitive.

Because we are dealing with a problem whose solution costs are bounded by a constant, we only consider strict competitiveness in this paper and hence omit the term "strictly".

Definition 2. Consider an input sequence $I = (x_1, \ldots, x_n)$. An *online algorithm A with advice* computes the output sequence $A^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the advice tape, i. e., an infinite bit string. The algorithm A is *c-competitive with advice complexity $s(n)$* if, for every n and every input sequence I of length at most n,

there is some ϕ such that $\text{cost}(A^\phi(I)) \leq c \cdot \text{cost}(\text{Opt}(I))$ and at most the first $s(n)$ bits of ϕ have been accessed during the computation of A on I.

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The number of vertices in G is denoted by n. A *path* is a graph $P = (V, E)$ such that $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}\}$. For short, we write $P = (v_1, v_2, \ldots, v_n)$ and $v_i \in P$. A cycle is a path with an edge between the first and the last vertex, i. e., $\{v_1, v_n\} \in P$. The *length* of a path is the number of edges in it. A *subpath* $(v_i, v_{i+1}, \ldots, v_j)$, for $1 \leq i \leq j \leq n$, is the induced subgraph $P[\{v_i, v_{i+1}, \ldots, v_j\}]$. If two vertices have distance 2 from each other, we call them *square neighbors*, since they are neighbors in P^2, the graph resulting from adding an edge between any two endpoints of a subpath of length 2. A vertex is *isolated* in some induced subgraph of P if it has neither direct nor square neighbors.

Definition 3. An $L(2,1)$-*coloring* of a graph G is a function assigning to every vertex of G a *color* from $\{0, 1, \ldots, \lambda\}$ such that adjacent vertices receive colors at least 2 apart and square neighbors receive different colors, i. e., at least 1 apart.

The aim of the $L(2,1)$-*coloring problem* is to find an $L(2,1)$-coloring minimizing λ. An $L(2,1)$-coloring of a graph with minimal λ is called *optimal*.

Clearly, $\lambda = 4$ is enough to color any path optimally offline. The algorithm simply follows the pattern 0-2-4 periodically from left to right.

Lemma 1 (Griggs and Yeh [12]). *To color a path on at least five vertices or any cycle, $\lambda = 4$ is necessary and sufficient.* □

In the following, we always mean $L(2,1)$-coloring when we speak of coloring.

Definition 4. The *online* $L(2,1)$-*coloring problem* is the following minimization problem. For some graph $G = (V, E)$ and an order on V, the vertices in V are revealed one by one with respect to this order. Let V_t denote the set of vertices revealed up to time step t. Together with each revealed vertex v, all edges between v and previously revealed vertices are revealed as well, i. e., up to time step t, the graph $G_t = G[V_t]$ is revealed. Additionally, in every time step t, information about the square neighbors of v among the vertices in V_t is revealed.[1]

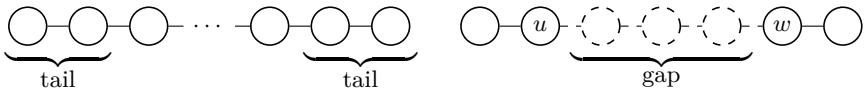
The goal is to find an $L(2,1)$-coloring $c\colon V \to \{0, \ldots, \lambda\}$ minimizing λ. For each revealed vertex v, the online algorithm immediately has to decide what color $c(v)$ this vertex gets.

In this paper, we consider paths, cycles, and graphs of maximum degree 2 as inputs for the online $L(2,1)$-coloring problem. Note that the online algorithm neither gets information about the number of vertices in G nor about the index of the currently revealed vertex, i. e., its position in the graph. In other words, the vertices are anonymous. Otherwise, the problem is trivial. In a path, the

[1] Intuitively speaking, the algorithm is told which of the already revealed vertices are square neighbors of v, but it gets no information about intermediate vertices that are not yet revealed. This additional constraint is usually not part of an online graph coloring setting, but necessary for our problem.

algorithm just assigns color $2 \cdot (i \bmod 3)$ to vertex v_i and is always optimal. In a cycle, this is also possible, but the algorithm needs to color the last few vertices differently with respect to the value $n \bmod 3$.

We denote by $c(P)$ the coloring of the path P computed by an algorithm and by $c(v)$ the color assigned to the vertex $v \in P$. We call the two directions in which a path/subpath can be extended from a vertex v the two *sides* of v. We call the two outmost vertices at the ends of a path/subpath the *tails* (see Figure 1a). If two or three vertices have not yet been revealed between two vertices u and w, we call these missing vertices a gap (see Figure 1b).



(a) The tails of a path. (b) A gap in a path.

Figure 1: The tails and gaps of a path.

Furthermore, we construct algorithms that sometimes need to choose one color out of three. Since they are reading from a binary advice tape, we need the following lemma.

Lemma 2 (Seibert et al. [19]). *Encoding a number x, for $1 \leq x \leq 3^n$, i. e., n one-out-of-three decisions, in a bit string needs at most $\frac{46}{29}n + d$ bits, for some positive constant d.* □

Throughout this paper, $\log x$ denotes the binary logarithm of x.

3. Online Algorithms without Advice

First, we consider the greedy algorithm, i. e., the algorithm that always picks the lowest possible color for a newly revealed vertex. The range $\{0, 1, \ldots, 8\}$, i. e., $\lambda \leq 8$, is obviously sufficient: A revealed vertex v has at most two direct and two square neighbors, which together forbid at most eight colors. Thus, at least one of the colors in $\{0, 1, \ldots, 8\}$ is still available for v. We show, however, that the greedy algorithm never uses the ninth color, i. e., that the range $\{0, 1, \ldots, 7\}$ is sufficient.

Theorem 1. *The greedy algorithm for the online $L(2, 1)$-coloring problem on paths achieves $\lambda = 7$, and this bound is tight.*

Proof. Let us first consider the upper bound. We need to show that the greedy algorithm uses $\lambda \leq 7$. In other words, we need to show that the greedy algorithm is never forced to color a vertex with a color greater than 7.

Assume for a contradiction that the greedy algorithm assigns a color greater than 7 to a vertex v. This means all the colors 0 through 7 were not available for v. This can only happen if the two direct neighbors of v forbid three colors each, the two square neighbors of v forbid one color each, these four color sets are disjoint, and all four neighbors are revealed before v. There are twelve possible arrangements satisfying these conditions (see Table 2).

1	3-1-v-6-4	4	0-5-v-2-7	7	3-1-v-5-7	10	0-6-v-3-1
2	3-6-v-1-4	5	0-2-v-6-4	8	3-5-v-1-7	11	6-1-v-4-7
3	0-2-v-5-7	6	0-6-v-2-4	9	0-3-v-6-1	12	6-4-v-1-7

Table 2: Arrangements forcing the greedy algorithm to assign a color greater than 7 to a vertex.

The arrangements can be interpreted as the possibilities to cover the interval from 0 to 7 with two blocks of size 3 and two blocks of size 1 (corresponding to the two direct and square neighbors, respectively). There are six possibilities to do this, each of which corresponds to two entries in Table 2 (resulting from exchanging the direct neighbors). For example, entries 1 and 2 correspond to choosing the two blocks 0-1-2 and 5-6-7 of size 3 (see Figure 2). We now show that none of these arrangements is possible.

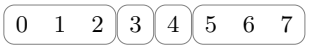


Figure 2: There are six possibilities to cover the interval from 0 to 7 with two blocks of size 3 and two blocks of size 1.

First, observe that the greedy algorithm only assigns color 6 to a vertex v if both direct neighbors of v are already revealed, as otherwise at most five colors are forbidden and v can always be assigned a color from $\{0, 1, \ldots, 5\}$. Hence, arrangements 1, 2, 5, 6, 9, and 10 cannot occur.

Consider now the arrangements 7, 11, and 12. The direct neighbor u of v not equal to 1 has, when it is revealed, at most one direct neighbor as well as at most two square neighbors. Since the direct neighbor of u is 6 or 7 and one square neighbor is 1, u can always be colored 0 or 2 by the greedy algorithm. Hence, these sequences cannot occur.

In arrangements 3 and 4, the vertex colored 5 can always be assigned color 3 or 4, and so the greedy algorithm never produces them.

Consider now arrangement 8. Let u be the vertex colored 1 and w the vertex colored 7. Either u is revealed before w, then the other direct neighbor of w is colored 0 (otherwise, u would get color 0), which in turn implies that w is assigned a color less than 7 by the greedy algorithm. Or w is revealed before u, then w has at most one direct neighbor and one square neighbor and is thus always assigned a color of at most 4 by the greedy algorithm.

We have shown that none of the twelve arrangements can occur and the greedy algorithm thus never uses a color greater than 7.

For the lower bound, consider the path $(v_1, v_2, \ldots, v_9)$. The vertices are revealed in the following order: v_1, v_2, v_3, v_9, v_8, v_7, v_5, v_6, v_4. The greedy algorithm hence assigns the colors $c(v_1) \coloneqq 0$, $c(v_2) \coloneqq 2$, $c(v_3) \coloneqq 4$, $c(v_9) \coloneqq 0$, $c(v_8) \coloneqq 2$, $c(v_7) \coloneqq 4$, $c(v_5) \coloneqq 0$, $c(v_6) \coloneqq 6$, $c(v_4) \coloneqq 7$, thus indeed needing $\lambda = 7$. Any problem instance that starts with this sequence of requests requires color 7, so infinitely many problem instances need $\lambda = 7$. □

Now we show that we cannot ensure optimality without advice. In fact, not even $\lambda = 5$ is enough to color every path online.

Theorem 2. *Without advice, $\lambda \geq 6$ is necessary to solve the online $L(2,1)$-coloring problem on paths.*

PROOF. We show that $\lambda = 5$ is not sufficient.

Let A be an online algorithm for the online $L(2,1)$-coloring problem on paths using only the colors 0 to 5, and consider the following instance. Seven vertices are revealed isolated, thus defining seven disjoint components $C_1, C_2, \ldots, C_7$.[2] For $1 \leq i \leq 7$, the next revealed vertex is then a direct neighbor of the last revealed vertex in C_i, until one tail of C_i is colored either 0-α or 5-β, with $\alpha \in \{2,3,4\}$, $\beta \in \{1,2,3\}$. The fact that such a tail always appears after a constant number of steps is guaranteed by the lower bound of $\lambda = 4$ stated in Lemma 1, because at some point, without loss of generality, a vertex v is colored 0. The next vertex is either colored 2, 3, or 4, and we are done, or it is colored 5, and we add another vertex, which can only be colored 1, 2, or 3, since its neighbor is colored 5 and its square neighbor is colored 0. (The case where v is colored 5 is analogous.) Furthermore, the adversary knows when this happens, because A is deterministic.

There are only six different tails with that property and we have seven components, so one tail must occur twice. We consider an instance that fills the gap between those two tails as follows.

Two tails that have both the form 0-2 or both the form 0-4 are connected by adding two vertices in between. As the two vertices in the gap need to be assigned colors with distance 2, at least one of them receives a color greater than 5. Two tails of the form 0-3 are connected by adding three vertices in between. As the leftmost vertex in the gap receives, without loss of generality, color 1 and the rightmost vertex color 5, the middle vertex can only receive a color greater than 5. (Due to symmetry, the argument is analogous for tails of the form 5-β, $\beta \in \{1,2,3\}$.)

Thus, none of the gaps can be filled using only the colors 0 to 5. □

The following theorem shows that this lower bound is tight.

Theorem 3. *Algorithm 1 solves the online $L(2,1)$-coloring problem on paths using $\lambda \leq 6$ without advice.*

PROOF. The algorithm assigns a pattern to the sides of a path. This means that the vertices are assigned colors according to the pattern even before they are revealed.

Observe that a vertex v is revealed either isolated, connected to one component or connected to two components. In the first case, line 3, v is marked such that the algorithm later knows that v was revealed as an isolated vertex. In the second case, v is revealed on a side of some marked vertex w. Either v is revealed on a side of w that was already assigned a pattern, and line 7 is used (see Figure 3a). Or v is the first vertex on a side of w, and line 5 is used (see Figures 3b and 3c). In the third case, lines 9–14 are used (see Figures 3d–3f). One can easily see that whenever a color is assigned, it is at most 6.

Lastly, we show that the algorithm produces a valid coloring. An isolated vertex is colored 0, and a vertex connected to one component is colored according to the pattern 0-3-5 or 5-3-0, i. e., these vertices indeed receive a valid coloring. The only remaining case is when two components are merged. Line 10 results in

[2]Note that we did not fix the order of these components nor the distance between them, and so the adversary is free to concatenate the components in any order or direction afterwards.

Algorithm 1

1: **for** every revealed vertex v at time t **do**
2: **if** v is isolated in G_t **then**
3: Set $c(v) \coloneqq 0$ and *mark* v.
4: **else if** v is the first revealed vertex on a side of some *marked* vertex w **and** v is only connected to the component containing w in G_{t-1} **then**
5: Assign the pattern 0-3-5 to the side containing v if possible, and 5-3-0 otherwise, and color v accordingly.
6: **else if** v is connected to a single component of G_{t-1} **and** v is on a side that was already assigned a pattern **then**
7: Color v according to that pattern.
8: **else** {v is connected to two vertices v_1, v_2 in different components of G_{t-1}.}
9: **if** v_1 and v_2 are isolated in G_{t-1} **then**
10: Color the vertices in the gap, depending on the gap size, with the pattern 3-5 or 3-1-5, and color v accordingly.
11: **else if** without loss of generality v_1 is isolated and v_2 belongs to a tail in G_{t-1} **then**
12: Assign the colors from Table 3a to the vertices in the gap, depending on the gap size, and color v accordingly.
13: **else** {v_1 and v_2 belong to two different tails in G_{t-1}.}
14: Assign the colors shown in Table 3b to the vertices in the gap, depending on the gap size, and color v accordingly.

Output: Color $c(v)$ of each revealed vertex v.

a valid coloring, and Tables 3a and 3b also contain only valid colorings, hence the algorithm indeed returns a valid coloring. □

4. Online Algorithms with Advice

Now, we want to investigate how much advice is necessary and sufficient to achieve optimality and 5/4-competitiveness, i. e., for $\lambda = 4$ and $\lambda = 5$.

4.1. Lower and Upper Bounds for Optimality

We first show that there is an optimal algorithm that reads advice bits from the tape such that it always knows what color from $\{0, 2, 4\}$ to assign to the currently revealed vertex. Then, we complement this result by a linear lower bound.

Theorem 4. *Algorithm 2 solves the online $L(2,1)$-coloring problem on paths optimally using at most $0.6955n + d$ advice bits, for some positive constant d.*

Proof. The oracle writes advice bits on the tape in such a way that the final coloring is just a repetition of the pattern 0-2-4.

Let us first explain what happens in line 9. There are two cases. In the first case, v is connected to at least one component C in G_{t-1} that consists of more than one vertex. Then, $c(v)$ is determined by the colors of the vertices in C by following the pattern 0-2-4 (see Figures 4a–4d). In the second case, v is connected to two vertices u and w that were isolated in G_{t-1} and have distance 4 from each other in G_t as in Figure 4e.

Now we calculate how many advice bits are needed. There are three situations where the algorithm reads advice bits (see Figure 5):

0-*3*-*5*-0-3-$\cdots$
0-*5*-*3*-0-5-$\cdots$
0-*5*-*1*-3-$\cdots$
0-*3*-*1*-5-$\cdots$
0-*3*-*1*-*5*-0-3-$\cdots$
0-*5*-*1*-*3*-0-5-$\cdots$
0-*3*-*5*-*1*-3-$\cdots$
0-*5*-*3*-*1*-5-$\cdots$

(a)

	0-3-	0-5-	3-0-	3-5-	5-0-	5-3-
-3-0	2-4 5-1-4					
-5-0	2-4 3-1-4	2-4 3-1-4				
-0-3	1-4 1-4-2	1-4 1-4-2	1-5 1-4-6			
-5-3	1-4 1-4-2	1-4 1-4-2	1-5 0-4-1	1-6 0-4-1		
-0-5	1-4 1-4-2	1-4 1-4-2	1-6 2-4-1	2-0 2-4-1	1-3 2-4-1	
-3-5	1-4 1-4-2	1-4 1-4-2	1-6 2-4-1	2-0 2-4-1	1-3 2-4-1	2-0 2-4-1

(b)

Table 3: (a) How to color gaps in line 12 of Algorithm 1, where the leftmost vertex is v_1 and the italic numbers correspond to the vertices in the gap. (b) How to color gaps in line 14 of Algorithm 1.

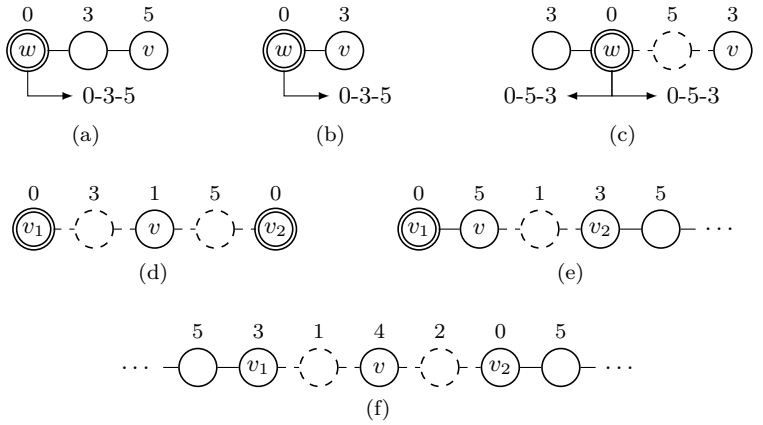


Figure 3: Examples of the different cases that Algorithm 1 checks when a new vertex v is revealed. Marked vertices are indicated with a double circle.

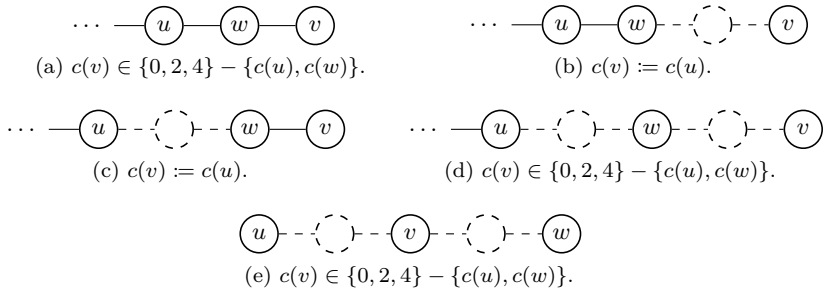


Figure 4: How to color v in line 9 of Algorithm 2.

Algorithm 2

1: **for** each revealed vertex v at time t **do**
2: **if** v is isolated in G_t **then**
3: Read a one-out-of-three decision from the advice tape and color v accordingly with a color from $\{0, 2, 4\}$.
4: **else if** v is connected to one vertex w that was isolated in G_{t-1} **or** v is connected to two vertices w and x that were both isolated in G_{t-1} and have distance 3 from each other in G_t **then**
5: Let $S \coloneqq \{0, 2, 4\} - c(w)$.
6: Read one advice bit b.
7: Color v with the lower of the two colors in S if $b = 0$, and with the higher one otherwise.
8: **else**
9: Inspect G_t to determine the color of v and color it accordingly.

Output: Color $c(v)$ of each revealed vertex v.

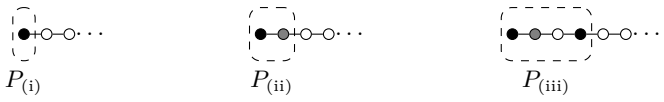


Figure 5: The three types of subpaths forcing Algorithm 2 to read advice bits. The black vertices are revealed isolated to force a one-out-of-three decision and the gray ones require one advice bit to color them.

(i) an isolated vertex is revealed (one-out-of-three decision);

(ii) a vertex connected to one previously isolated vertex is revealed (one advice bit needed);

(iii) a vertex is revealed that is connected to two previously isolated vertices that are at distance 3 to each other (one advice bit needed).

We show that, even for the worst possible input for the algorithm, the claimed amount of advice bits is sufficient.

Since vertices corresponding to line 9 do not force the algorithm to read advice, we can assume, without loss of generality, that they only occur to connect subpaths $P_{(\mathrm{i})}$, $P_{(\mathrm{ii})}$ and $P_{(\mathrm{iii})}$ that enforce situations (i), (ii) or (iii) (see Figure 5). Let $P'_{(\mathrm{i})}$, $P'_{(\mathrm{ii})}$ and $P'_{(\mathrm{iii})}$ denote the respective subpaths together with two such connecting vertices.

The subpath $P_{(\mathrm{i})}$ is just one isolated vertex, i. e., $P'_{(\mathrm{i})}$ contains three vertices. For situation (ii), the subpath $P_{(\mathrm{ii})}$ consists of two vertices, a vertex v_1 revealed in isolation and a directly adjacent vertex v_2 revealed after v_1, thus $P'_{(\mathrm{ii})}$ contains four vertices. Situation (iii) can be enforced by the subpath $P_{(\mathrm{iii})} = (w_1, w_2, w_3, w_4)$ where first the two isolated vertices w_1 and w_4 are revealed and then the vertices w_2 and w_3. Hence, $P'_{(\mathrm{iii})}$ contains six vertices.

For each of these subpaths, we calculate how many advice bits the algorithm reads on average asymptotically per vertex. Let m_1, m_2, and m_3 be the number of vertices in subpaths $P'_{(\mathrm{i})}$, $P'_{(\mathrm{ii})}$, and $P'_{(\mathrm{iii})}$, respectively.

The part of the instance built by subpaths $P'_{(\mathrm{i})}$ needs one-out-of-three decision for every three vertices, requiring asymptotically $\frac{1}{3} \cdot \frac{46}{29} m_1 = \frac{46}{87} m_1$ advice bits.

The part of the instance consisting of subpaths $P'_{(\text{ii})}$ needs one one-out-of-three decision and one binary decision for every four vertices, hence requiring the algorithm to read asymptotically $\frac{1}{4}(\frac{46}{29}+1)m_2 = \frac{75}{116}m_2$ advice bits. The part of the instances containing all subpaths $P'_{(\text{iii})}$ needs two one-out-of-three decisions and one binary decision for every six vertices. Thus, the algorithm reads $\frac{1}{6}(2 \cdot \frac{46}{29}+1)m_3 = \frac{121}{174}m_3$ advice bits.

As we have seen, we can assume, without loss of generality, that a hard instance for the algorithm is built by concatenating multiple copies of these subpaths.

As the instances built from the subpaths $P'_{(\text{iii})}$ require the biggest amount of advice per vertex, we consider instances that are built solely from these subpaths. Observe that there is one gap less than there are paths $P_{(\text{iii})}$, i. e., one of these paths need not be extended to a path $P'_{(\text{iii})}$. On the other hand, the number of vertices is not necessarily 4 mod 6, so up to five more vertices might require some constant amount of advice.

A simple case distinction on the value n mod 6, i. e., on the length of the last path, yields that we can bound the number of advice bits read by Algorithm 2 by $\frac{121}{174}n + 1.3909$. □

In order to show a lower bound, we need the following technical lemma.

Lemma 3. *Consider two paths $P = (u, v)$ and $P' = (w, x)$ of two vertices each, and let c be any valid coloring for P and P'. Let now (u, v, y, z, w, x), (u, v, y, z, x, w), (v, u, y, z, w, x), and (v, u, y, z, x, w) be the four possible paths that result from concatenating P and P' at arbitrary ends by adding two vertices. For at least one of the four paths, any valid coloring that extends c needs to use a color greater than 4.*

Proof. Table A.5 in the appendix shows, for every possible combination of paths P and P', how to connect them such that every coloring of the gap needs to use a color greater than 4. Note that there are colorings for which only one combination enforces a color greater than 4. □

When proving lower bounds on the advice complexity, we usually construct a set of special instances having the property that the algorithm cannot distinguish between their prefixes of a certain length. Using at most b advice bits, the algorithm can only use 2^b different strategies and thus can only use one of 2^b different colorings. We describe an adversary that constructs at least $2^b + 1$ different continuations that require pairwise different colorings of the prefix to be colored optimally. Thus, at least one of the instances cannot be colored optimally by the algorithm, i. e., b advice bits are not sufficient. Since the prefix of the instance in general consists of some set of isolated vertices and subpaths and the adversary is free to choose the order and orientation of these subgraphs, we cannot in general assume that the prefix consists of exactly the same vertices in the same order for all instances. But for the proof method to work it suffices if the prefixes are isomorphic subgraphs with respect to both direct and square neighbors. We call two subgraphs satisfying this property *indistinguishable*.

Theorem 5. *Any algorithm that solves the online $L(2, 1)$-coloring problem on paths optimally needs to read at least $0.0518n$ advice bits.*

PROOF. We consider a path P with $n = 8m$ vertices, for some $m \in \mathbb{N}$, and we partition P into m consecutive vertex-disjoint subpaths $B_1, B_2, \ldots, B_m$ of eight vertices each. In every $B_j = (v_i, v_{i+1}, \ldots, v_{i+7})$, where $i = 8(j-1)+1$, we define the two subpaths $T_{2j-1} = (v_{i+1}, v_{i+2})$ and $T_{2j} = (v_{i+5}, v_{i+6})$. We call B_j the *j-th block* and T_h the *h-th tuple* of P. This block-tuple division of P is shown in Figure 6.

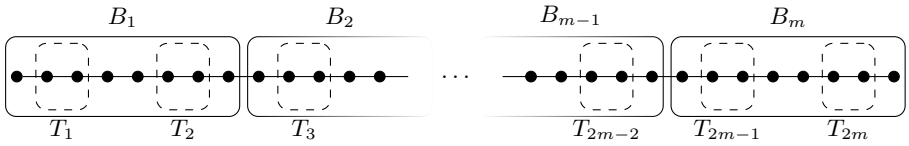


Figure 6: The block-tuple division of the path P in the proof of Theorem 5.

We consider as instances all possible online presentations of P satisfying the following conditions.

- The algorithm first receives the vertices from every tuple T_h. In G_2, i. e., after time step 2, only T_1 is revealed, in G_4, T_1 and T_2 are revealed, and so on, until all subpaths $T_1, T_2, \ldots, T_{2m}$ have been revealed at time $4m$.
- After time step $4m$, all remaining vertices are revealed sequentially from left to right.

Under these conditions, there are two possible orders of revealing the vertices in each tuple T_h. Hence, there are 4^m different instances of this type. Moreover, the prefixes of all instances until time step $4m$ are indistinguishable, so if two instances get the same advice, they have the same coloring at time step $4m$.

Consider an instance I whose associated advice induces a coloring c on the $2m$ tuples in G_{4m}. We want to determine how many other instances can receive the same advice string, i. e., for how many of them the algorithm can use the same coloring of the tuples in G_{4m} and still be able to use only colors 0 to 4 in the remaining part. Consider the block B_j containing the subpaths T_{2j-1} and T_{2j}. Lemma 3 shows that there is always an appearance order of the vertices in T_{2j-1} and T_{2j} such that the gap in between cannot be colored with values from 0 to 4. This means that c is suitable for at most three choices of B_j out of four, and since the same reasoning holds for all other blocks, c is suitable for at most 3^m different instances.

Hence, there must be at least $4^m/3^m$ different advice strings, implying that at least $\log((4/3)^m) = (2 - \log 3) \cdot n/8 \geq 0.0518n$ advice bits are necessary. □

4.2. Lower and Upper Bounds for $\frac{5}{4}$-Competitiveness

The main idea is to define an algorithm that works like Algorithm 1 most of the time and avoids situations that lead to using color 6. The algorithm reads an advice bit for the first vertex revealed on each side, unless it merges two components. Advice bit 0 means follow the pattern 0-3-5 as in Algorithm 1, while advice bit 1 means switch to pattern 0-2-4.

Theorem 6. *Algorithm 3 solves the online $L(2,1)$-coloring problem on paths with $\lambda \leq 5$ and uses at most $0.4n + d$ advice bits, for some positive constant d.*

Algorithm 3

for every revealed vertex v at time t **do**
    if v is isolated in G_t **then**
        Set $c(v) \coloneqq 0$ and *mark* v.
    else if v is the first revealed vertex on a side of some *marked* vertex w in G_t **and** v is only connected to the component containing w in G_{t-1} **then**
        Read an advice bit b.
        if $b = 0$ **then**
            if there is a direct neighbor on the other side of w that has or will get color 2 or 3, **then** assign pattern 0-5-3; **else** assign pattern 0-3-5.
        else
            if there is a direct neighbor on the other side of w that has or will get color 2 or 3, **then** assign pattern 0-4-2; **else** assign pattern 0-2-4.
        Color v according to the pattern.
    else if v is connected to a single component of G_{t-1} **and** v is on a side that was already assigned a pattern **then**
        Color v according to that pattern.
    else {v is connected to two vertices v_1, v_2 of different components in G_{t-1}.}
        if v_1 and v_2 are isolated in G_{t-1} **then**
            Color the vertices in the gap, depending on the gap size, with the pattern 3-5 or 3-1-5, and color v accordingly.
        else if without loss of generality v_1 is isolated and v_2 belongs to a tail in G_{t-1} **then**
            if the tail is colored with the pattern 0-3-5 **then**
                Assign the colors from Table 3a to the vertices in the gap, depending on the gap size, and color v accordingly.
            else {The tail is colored with the pattern 0-2-4.}
                Color the gap between v_1 and v_2 with the pattern 3-1 or 3-5-1, depending on the gap size, if v_2 has color 4, otherwise use the pattern 3-5 or 3-1-5.
        else {v_1 and v_2 belong to a tail in G_{t-1}.}
            Color the gap according to Table A.6 in the appendix.

Output: Color $c(v)$ of each revealed vertex v.

	Original	Alternative		Original	Alternative
1	-0-3-␣-␣-␣-3-0-	-0-2-*4*-*1*-*5*-3-0-	7	-5-3-␣-␣-3-5-	-4-2-*5*-*0*-3-5-
2	-0-3-␣-␣-5-0-	-0-2-*4*-*1*-5-0-	8	-5-3-␣-␣-␣-2-0-	-4-2-*5*-*0*-*4*-2-0-
3	-0-3-␣-␣-5-3-	-0-2-*4*-*1*-5-3-	9	-5-3-␣-␣-4-2-	-4-2-*5*-*1*-4-2-
4	-0-2-␣-␣-0-3-	-0-3-*1*-*5*-0-3-	10	-0-2-␣-␣-2-0-	-0-3-*1*-*5*-2-0-
5	-4-2-␣-␣-0-3-	-5-3-*1*-*4*-0-3-	11	-0-4-␣-␣-4-0-	-0-5-*3*-*1*-4-0-
6	-3-5-␣-␣-2-4-	-2-4-*1*-*5*-2-4-	12	-2-4-␣-␣-4-2-	-3-5-*2*-*0*-4-2-

Table 4: The twelve problematic situations in the proof of Theorem 6.

Proof. Let us first explain what "will get color x or y" in lines 7 and 9 means. Without loss of generality, v is revealed to the left of a marked vertex w. It may happen that, in this time step, only the right square neighbor of w has already been revealed and colored according to some pattern. But then, the right direct neighbor of w will be assigned a color according to that pattern when it is revealed. The algorithm considers this color for its decision.

Observe that a vertex v revealed at time t either (i) occurs as an isolated vertex, (ii) is connected to one or (iii) two components of G_{t-1}. In case (i), line 3 tells us what to do. In case (ii), there are two possibilities. Either v extends a tail that was already assigned a pattern, then line 12 is used. Or the tail was not yet assigned a pattern, then lines 5 to 10 are used. In case (iii), v is revealed between two vertices v_1 and v_2. If both of them are isolated, line 15 is used. If only one of them is isolated, lines 17 to 20 are used. And if neither of them is isolated, then line 22 is used.

Now we show that the resulting coloring is proper and does not use a color greater than 5. As long as a vertex v is revealed isolated or extends a tail, v is colored according to a valid pattern. More interesting situations occur when two components are merged. Lines 15 and 20 as well as Table 3a show that no color greater than 5 is used when one of the components consists of an isolated vertex. The only problematic case is when two tails are merged, i. e., when Table A.6 in the appendix is used in line 22 of the algorithm. Every tail that occurs at some point during the computation is colored 0-3, 0-5, 3-5, 0-2, 0-4, or 2-4 (or the reverse), other color combinations are only used to close a gap. Hence, Table A.6 covers all possible cases that can occur in line 22. Thus, it is always possible to color a gap using at most color 5 except in the twelve cases marked by a bolt. These twelve cases are also shown in Table 4.

However, these twelve cases do not occur, because the oracle can always give advice in such a way as to avoid these cases, as we shall see.

Note that in Table 4, the colors of a left tail, i. e., a tail left of the gap, are determined by some advice bit. Let w be the rightmost marked vertex in such a left tail. There are two cases. The first case is that the direct neighbor u left of w has a color in $\{2, 3\}$. Then, the pattern right of w is 0-5-3 or 0-4-2. The second case is that u was not yet revealed, there is no u, or u has a color in $\{4, 5\}$, then the pattern right of w is 0-3-5 or 0-2-4. In both cases, the corresponding advice bit can be flipped to change from one pattern to the other, which allows the gap to be colored as desired (see Table 4). Figure 7 shows the pattern change in situations 5 and 12.

Finally, we need to show that Algorithm 3 needs at most $0.4n + d$ advice bits. Note that no advice is read when an isolated vertex is revealed. But one advice

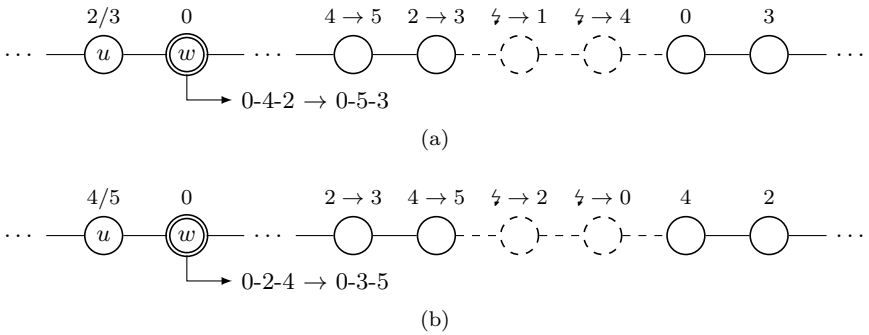


Figure 7: Situations 5 (a) and 12 (b) of Table 4. In situation 5, the pattern 0-4-2 can be changed to 0-5-3, and in situation 12, 0-2-4 can be changed to 0-3-5.

bit is read for the first revealed vertex on each side of an isolated vertex. This pattern can only be repeated with at least two additional vertices in between, so asymptotically at most $\frac{2}{5}n + d = 0.4n + d$ advice bits are read in total, for some positive constant d. □

Theorem 7. *Every algorithm for the online $L(2,1)$-coloring problem on paths with $\lambda \leq 5$ needs to read at least $3.9402 \cdot 10^{-10}n$ advice bits.*

Proof. Let us fix an arbitrary algorithm A. In order to force A to use a certain amount of advice, the adversary constructs an instance as follows.

First, 25 isolated vertices are revealed. Then, there are two possibilities.

1. The adversary selects seven arbitrary vertices $u_1, u_2, \ldots, u_7$ out of the 25. It reveals a neighbor v_i of each u_i such that the paths (u_i, v_i) are still separate components. Then, there are again two possibilities.
   (a) The adversary connects two arbitrary vertices v_j and v_k by adding two or three vertices between them.
   (b) It reveals another seven vertices w_i that are neighbors of the v_i such that the paths (u_i, v_i, w_i) are still separate components. Then, it connects two arbitrary vertices w_j and w_k by adding two or three vertices in between.
2. The adversary selects four arbitrary vertices u_1, u_2, u_3, u_4 out of the 25. It reveals four vertices v_1, v_2, v_3, v_4, two of which form a path between u_1 and u_2 and between u_3 and u_4, respectively, i. e., there are now two paths (u_1, v_1, v_2, u_2) and (u_3, v_3, v_4, u_4). Then, it connects these two paths at arbitrary ends by adding two vertices in between.

At the end, the adversary connects all components according to some fixed order such that the resulting graph is a path. This is possible because all components are paths.

We now show that, for every coloring of the 25 initial vertices, there is an instance that forces A to use color 6. To this end, we make a case distinction. Note that the pigeonhole principle implies that at least one of the following six cases is always true.

1. At least seven vertices $u_1, \ldots, u_7$ receive color 0. In some instance, they are extended to paths (u_i, v_i). Now there are two subcases, at least one of which is true due to the pigeonhole principle.

   (a) There are two paths where the second vertex has color 2, 3 or 4, i. e., without loss of generality, $c(v_1) = c(v_2) = j$, for $j \in \{2, 3, 4\}$. There is an instance that connects these two paths by inserting three vertices between them if $j = 3$ or by inserting two vertices if $j \in \{2, 4\}$. The proof of Theorem 2 tells us that any valid coloring of this path uses color 6.

   (b) There are four paths where the second vertex has color 5, i. e., without loss of generality, $c(v_i) = 5$, for $1 \leq i \leq 4$. There is an instance that extends all the paths with an additional vertex, i. e., we then have paths (u_i, v_i, w_i). Once more, the pigeonhole principle tells us that at least two paths are colored identically, i. e., without loss of generality, we have $c(u_1) = c(u_2) = 0$, $c(v_1) = c(v_2) = 5$ and $c(w_1) = c(w_2) = j$, for $j \in \{1, 2, 3\}$. There is an instance that connects these two paths by inserting three vertices between them if $j = 2$ or by inserting two vertices if $j \in \{1, 3\}$. Again, the proof of Theorem 2 tells us that every valid coloring of this path uses color 6.

2. At least four vertices u_1, u_2, u_3, u_4 receive color 1. There is an instance in which the two vertices u_1 and u_2, and also u_3 and u_4, respectively, are each connected by two vertices to (u_1, v_1, v_2, u_2) and to (u_3, v_3, v_4, u_4), respectively. We can without loss of generality assume that $c(v_1) = c(v_3) = 3$ and $c(v_2) = c(v_4) = 5$. There is an instance that connects u_2 and u_4 by inserting two vertices between them. Once more, every valid coloring of this path uses color 6, as can be seen in the proof of Theorem 2.

3. At least four vertices receive color 2. Then, we can connect the vertices as in the previous case and then w.l.o.g. assume that $c(v_2) = c(v_4) = 0$. Then, we can use exactly the same argument as in the previous case.

4. At least four vertices receive color 3. This case is symmetric to case 3.

5. At least four vertices receive color 4. This case is symmetric to case 2.

6. At least seven vertices receive color 5. This case is symmetric to case 1.

Let us now count the overall number of vertices. Case 1(b) needs the most vertices, the adversary selects seven of the 25 isolated vertices and attaches two vertices to each of them as described above. Hence, we have $7 \cdot 3 + 18$ vertices. But we still need to connect the 25 components with each other, using additional $23 \cdot 2 + 3$ vertices, because w_j and w_k may be connected by a path on three vertices. Thus, one particular instance consists of at most 88 vertices.

Let us count how many instances are in our special class of problem instances for a fixed coloring of 25 given initial vertices. There are $\binom{25}{7}$ possible ways to select seven vertices, two ways to expand them (either to paths of length 1 or 2), $\binom{7}{2}$ ways to select two vertices among those seven, and finally another two possible ways to connect them, by inserting two or three vertices in between. Moreover, there are $\binom{25}{2} \cdot \binom{23}{2}$ possible ways to select two couples of vertices and insert two additional vertices in between, and four possible ways to connect

the resulting two paths with each other at arbitrary ends. The total number of instances is at most

$$N \coloneqq \binom{25}{7} \cdot 2 \cdot \binom{7}{2} \cdot 2 + \binom{25}{2} \cdot \binom{23}{2} \cdot 4.$$

We have shown that, for every coloring of the 25 initial vertices, at least one instance forces a color greater than 5 later on. In other words, one fixed coloring of the initial 25 vertices can be used for at most $N - 1$ of the N instances. Because the first 25 vertices are all isolated, reading advice is the only way in which a deterministic algorithm can achieve different colorings of these vertices.

Consider the following scenario. The adversary presents m times one of the instances described above. There are N^m possible ways to do this, and every initial coloring of the isolated vertices can result in a valid coloring of the entire graph for at most $(N-1)^m$ of them. Thus, any algorithm needs to choose between at least $(N/(N-1))^m$ many colorings for the initial $25m$ vertices. To distinguish them, it needs at least

$$\log\left(\left(\frac{N}{N-1}\right)^m\right) = m \log\left(\frac{N}{N-1}\right)$$

advice bits.

The overall construction consists of $n \leq 88m + 2(m-1) \leq 90m$ vertices, because we need to connect the instances with each other by adding at least two additional vertices in between. Thus, we need at least

$$m \log\left(\frac{N}{N-1}\right) \geq \frac{n}{90} \log\left(\frac{N}{N-1}\right) \geq 3.9402 \cdot 10^{-10} n$$

advice bits in total. □

5. Cycles and Graphs with Maximum Degree 2

Many of our results can be extended to cycles and graphs of maximum degree 2, i. e., collections of paths and cycles. The results for graphs of maximum degree 2 are exactly the same, except for the upper bound for optimality, which is $1.5314n + O(\log n)$.

5.1. Cycles

The following lower bound for cycles is known.

Lemma 4 (Griggs and Yeh [12]). *Every algorithm that solves the online $L(2,1)$-coloring problem on cycles without advice needs $\lambda \geq 4$.* □

We improve the previous result by establishing a tight bound.

Theorem 8. *To color a cycle without advice, $\lambda = 6$ is necessary and sufficient.*

Proof. For the lower bound, we reduce this to the lower bound for paths (Theorem 2). We enforce color 6 in a path and add two additional vertices at the end to connect the two tails of the path with each other.

For the upper bound, we can reuse Algorithm 1; however, the algorithm needs to treat different ends of the same component as two different components. This is only relevant when the cycle is closed. This is also the only situation that can occur in cycles and not in paths. Then, the algorithm colors the gap according to Table 3b. However, cycles of length 3 need special treatment. The first revealed vertex is always colored 0, the second one 3, and the third one 5.

Clearly, the algorithm only uses colors 0 to 6. The argumentation that the resulting coloring is a valid $L(2,1)$-coloring is—except for cycles of length 3—completely analogous to the one in the proof of Theorem 3. □

Theorem 9. *There is an optimal algorithm with advice for the online $L(2,1)$-coloring problem on cycles that uses at most $0.6955n + d$ advice bits, for some positive constant d.*

Proof. The first bit on the advice tape is 1 if $n' \coloneqq n \bmod 3 = 0$, and 0 otherwise.

Case 1: $n' = 0$. The algorithm behaves as Algorithm 2. This works because in this case, we do not care that we are dealing with a cycle, because it does not pose any additional restrictions on our coloring. Every cycle with a number of vertices divisible by 3 can be colored with the pattern 0-2-4. Hence, the algorithm asymptotically needs $0.6955n + 1.3909$ advice bits.

Case 2: $n' \neq 0$. The algorithm assigns color 1 to the first revealed vertex v. Let u and w be the direct neighbors of v. As soon as either of them is revealed, the algorithm reads one advice bit. If it is 0, $c(u) = 3$ and $c(w) = 4$, otherwise $c(u) = 4$ and $c(w) = 3$. Observe that the remaining graph can always be colored with the pattern 0-2-4, and we can thus use Algorithm 2. Thus, in this case, the algorithm asymptotically needs $0.6955n + 0.3047$ advice bits. □

Theorem 10. *Any algorithm that solves the online $L(2,1)$-coloring problem on cycles optimally needs to read at least $\frac{n-2}{8}(2 - \log 3) \geq 0.0518n + d$ advice bits, for some negative constant d.*

Proof. Again, we can reduce this to the respective lower bound for paths (Theorem 5). At the end, we insert two additional vertices to close the cycle. Therefore, we have $n = 8m + 2$, which implies that

$$\log\left(\left(\frac{4}{3}\right)^m\right) = \frac{(2 - \log 3)(n-2)}{8} \geq 0.0518n - 0.1038$$

advice bits are necessary. □

Theorem 11. *There is an algorithm that solves the online $L(2,1)$-coloring problem on cycles with $\lambda \leq 5$ using at most $0.4n + d$ advice bits, for some positive constant d.*

Proof. As above, we can reuse the respective algorithm for paths, i. e., Algorithm 3. However, we need to adapt it for cycles. Lines 5 to 10 and line 12 are only executed when the cycle is not closed, whereas lines 14 to 22 are executed when the cycle is closed.

Again, a vertex v is revealed isolated, connected to one component C or to two components.

In the first case, v is marked such that the algorithm knows later that it was revealed isolated. In the second case, there are two possibilities. The first one is that v is connected to C at only one end. Then, either v is on a side that was already assigned a pattern, and line 12 is used, or v is the first revealed vertex on the side of a marked vertex, and lines 5–10 are used. The other possibility is that v is connected to C at two different ends, and Table A.6 is used. The last case is that v is connected to two different components, and lines 14–22 are used.

Clearly, the algorithm only uses colors 0 to 5. The argumentation that the resulting coloring is a valid $L(2,1)$-coloring is completely analogous to the one in the proof of Theorem 6. □

Theorem 12. *Any algorithm that solves the online $L(2,1)$-coloring problem on cycles with $\lambda \leq 5$ needs to read at least $3.9402 \cdot 10^{-10} n$ advice bits.*

Proof. This proof is again almost completely analogous to the one for paths, i. e., to the one of Theorem 7. At the end, we insert two additional vertices to connect the two ends of the path. Thus, $n \leq 88m + 2(m-1) + 2 = 90m$, and the result directly carries over to cycles. □

5.2. Graphs with Maximum Degree 2

We have considered paths and cycles until now. Graphs with a maximum degree of 2 consist only of such components. We can therefore generalize our results as shown in Table 1. The lower bounds follow from Theorems 10 and 12, respectively. To be 5/4-competitive, we can use Algorithm 3 with the modifications described in the proof of Theorem 11. This is possible because Algorithm 3 does not treat the very first vertex specially, i. e., we do not need to distinguish between the first revealed vertex of every component and all other vertices that are revealed isolated.

Now we prove the upper bound for optimality, for which we need the following result.

Lemma 5. *Every component with at most five vertices and maximum degree 2 can be colored optimally without advice.*

Proof. A possible algorithm assigns to the first revealed vertex v the color 0 and assigns the pattern 0-3-1-4- to the side of v from which a vertex is first revealed, and the pattern 0-2-4- to the other side. In a cycle of size at most 5, exactly one vertex is revealed isolated. Therefore, we know that a component is a path as soon as a second isolated vertex u is revealed. In this case, we assign color 0 to u and when another vertex w is revealed, we make a case distinction. If w connects u and v, then we color the gap with -2-4- or -3-1-4- (assuming that v is on the left of the gap). Otherwise, assign color 3 to w. Then, there is always a valid coloring for the remaining vertices.

Observe that, for a cycle of size 3, the algorithm knows the structure of the component after the second vertex is revealed, and can color the vertices 0-2-4. For a cycle of size 4, however, the above strategy always works, independent of the presentation order of the vertices. For a cycle of size 5, the algorithm knows the structure of the component after the third or fourth vertex is revealed and can always extend the coloring to 0-3-1-4-2. □

Theorem 13. *There is an optimal algorithm for the online $L(2,1)$-coloring problem on graphs with maximum degree 2 that reads at most $1.5314n + O(\log n)$ advice bits.*

Proof. Let us call a component *large* if it consists of at least six vertices, and let l denote the number of vertices in large components. Some of the presented algorithms treat the first revealed vertex in a component specially. We call these vertices *pioneers* from now on.

First, the algorithm reads from the advice tape the values n and l. The number n can be encoded in a self-delimiting way using at most $2\lceil \log n \rceil \leq 2\log n + 2$ advice bits [16], and since $l \leq n$, we need at most another $2\log n + 2$ advice bits for l.

Depending on the next advice bit, the algorithm chooses one of the following two strategies. Both strategies consist of two phases each.

Strategy 1. In the first phase, the algorithm reads from the tape m advice bits that encode the indices of all pioneers in the ordered list of all revealed vertices. Clearly, there can be at most $\lfloor l/6 \rfloor$ many pioneers. We now give an upper bound on m. There are $\sum_{i=0}^{\lfloor l/6 \rfloor} \binom{n}{i}$ many possible ways to distribute the pioneers among all vertices, and we need equally many advice strings to distinguish them. Hence,

$$\begin{aligned} m &\leq \left\lceil \log \left(\sum_{i=0}^{\lfloor l/6 \rfloor} \binom{n}{i} \right) \right\rceil \leq \log \left(\sum_{i=1}^{\lfloor l/6 \rfloor} \binom{n}{i} \right) + 2 \\ &\leq \log \left(\frac{l}{6} \cdot \binom{n}{\lfloor l/6 \rfloor} \right) + 2 \leq \log l - \log 6 + \frac{l}{6} \log \left(\frac{\mathrm{e}n}{\lfloor l/6 \rfloor} \right) + 2 \\ &\leq \log l - \log 6 + \frac{l}{6} \log \left(\frac{6\mathrm{e}n}{l} \right) + 4, \end{aligned}$$

where the fourth inequality follows from $\binom{n}{k} \leq \frac{n^k}{k!} \leq \frac{(\mathrm{e}n)^k}{k^k}$ due to the Taylor series of e.

In the second phase, every pioneer v is colored 4. Let u and w be the direct and square neighbor of v on the side where a neighbor is first revealed. When this happens, the algorithm reads one advice bit b. If $b = 0$, the component is a path or a cycle of size 0 mod 3, and the algorithm assigns the colors 0 and 2 to u and w. If $b = 1$, the component is a cycle of size $\neq 0$ mod 3, and the algorithm assigns the colors 1 and 3 to u and w.

For all other vertices in large components, the algorithm behaves as in the proof of Theorem 9. Let us now analyze how much advice is necessary for the large components. In every large component, the pioneer is colored 4, and one advice bit is sufficient to color at least three more vertices optimally.[3] Let k be the number of large components, and let l_i, for $1 \leq i \leq k$, be the number of vertices in the i-th large component.

[3]The worst case is if the end vertex of a path is a pioneer, and its direct neighbor is then colored based on an advice bit.

We know that $d \leq 1.3909$ in Theorem 9. This implies that we need at most

$$\left\lceil \sum_{i=1}^{k} (0.6955(l_i - 4) + 1 + 1.3909) \right\rceil \leq 0.6955 \sum_{i=1}^{k} l_i = 0.6955l$$

advice bits.

But we still need to consider small components. The worst-case here is that there are only isolated vertices, because the algorithm does not know whether such a vertex remains isolated until the very end, i. e., forms a small component of size 1, or whether it belongs to some large component whose pioneer was already revealed. Thus, it needs asymptotically 1.5863 advice bits for each such vertex, i. e., $\lceil 1.5863(n-l) \rceil \leq 1.5863(n-l) + 1$ advice bits in total.

In total, strategy 1 needs at most

$$\begin{aligned}
&\log l - \log 6 + \frac{l}{6} \log \left(\frac{6en}{l} \right) + 4 + 0.6955l + 1.5863(n-l) + 1 \\
&\leq \frac{l}{6} \log \left(\frac{6en}{l} \right) + 1.5863n - 0.8908l + \log l + 2.4151 \\
&\leq \frac{l}{6} \log \frac{n}{l} + 1.5863n - 0.2195l + \log l + 2.4154 =: \alpha(l)
\end{aligned}$$

advice bits.

Strategy 2. In the first phase, the algorithm reads from the tape m' advice bits that encode the indices of all pioneers as well as of all other vertices in large components that are revealed isolated. One can easily see that there can be at most $\lfloor 3l/7 \rfloor$ many such vertices.[4] Moreover, there are exactly $\sum_{i=0}^{\lfloor 3l/7 \rfloor} \binom{n}{i}$ many vertex subsets of size at most $\lfloor 3l/7 \rfloor$. For each vertex in such a subset, the algorithm must know whether it is a pioneer or not. We get

$$\begin{aligned}
m' &\leq \left\lceil \log \left(\sum_{i=0}^{\lfloor 3l/7 \rfloor} \binom{n}{i} 2^i \right) \right\rceil \leq \log \left(\sum_{i=1}^{\lfloor 3l/7 \rfloor} \binom{n}{i} 2^i \right) + 2 \\
&\leq \log \left(\frac{3l}{7} \binom{n}{\lfloor 3l/7 \rfloor} 2^{\lfloor 3l/7 \rfloor} \right) + 2 \\
&\leq \log \frac{3l}{7} + \log \left(\binom{n}{\lfloor 3l/7 \rfloor} \right) + \frac{3l}{7} + 2 \leq \log \frac{3l}{7} + \frac{3l}{7} \log \left(\frac{7ne}{3l} \right) + \frac{3l}{7} + 2 \\
&\leq \log \frac{3}{7} + \log l + \frac{3l}{7} \log \frac{7e}{3} + \frac{3l}{7} \log \frac{n}{l} + \frac{3l}{7} + 2 \\
&\leq -1.2223 + \log l + 1.1422l + \frac{3l}{7} \log \frac{n}{l} + \frac{3l}{7} + 2 \\
&\leq \frac{3l}{7} \log \frac{n}{l} + 1.5708l + \log l + 0.7777.
\end{aligned}$$

In the second phase, the algorithm colors all vertices in large components according to the proof of Theorem 9, and all vertices in small components as

[4] The worst case is a path of length 6 where the first, the last, and the middle vertex are revealed isolated.

shown in the proof of Lemma 5. Clearly, for the latter, no advice is necessary at all. For the former, we know from the analysis above that the algorithm needs $0.6955l$ advice bits.

Thus, in total, strategy 2 needs at most

$$\begin{aligned}&\frac{3l}{7}\log\frac{n}{l} + 1.5708l + \log l + 0.7777 + 0.6955l\\ &\leq \frac{3l}{7}\log\frac{n}{l} + 2.2663l + \log l + 2.4154 =: \beta(l)\end{aligned}$$

advice bits.

We have $\alpha(l) = \beta(l)$ for $l \approx 0.5909n$. Moreover, we now show that we can indeed bound $\min\{\alpha, \beta\}$ from above by $\alpha(0.5909n)$.

Using the (for our purposes) reasonable conventions that $0 \log \frac{n}{0} = \log 0 = 0$, we get $\alpha(0) > \beta(0)$ and $\alpha(n) < \beta(n)$. Moreover,

$$\beta'(l) = \frac{3\log n}{7} - \left(\frac{3l}{7l} + \frac{3\log l}{7}\right) + 2.2663 + \frac{1}{l} \geq \frac{1}{l} + 1.8377,$$

i. e., β is monotonously increasing. Also, for $0.5909n \leq l \leq n$ and every $n \geq 6$,

$$\begin{aligned}\alpha'(l) &= \frac{\log n}{6} - \left(\frac{l}{6}\frac{1}{l} + \frac{\log l}{6}\right) - 0.2195 + \frac{1}{l}\\ &\leq \frac{\log(n/l)}{6} - 0.3861 + \frac{1}{l} \leq -0.2985 + \frac{1}{0.5909n} < 0,\end{aligned}$$

i. e., α is monotonously decreasing. Thus, indeed

$$\min\{\alpha(l), \beta(l)\} \leq \alpha(0.5909n) \leq 1.5314n + O(\log n),$$

for $0 \leq l \leq n$. Hence, in total, we need

$$2\log n + 2 + 1 + \min\{\alpha(l), \beta(l)\} = 1.5314n + O(\log n)$$

advice bits. □

6. Randomized Online Algorithms

In this section, we give a lower bound on the competitive ratio achievable by any randomized online algorithm. Our proof is based on the following result.

Lemma 6 (Böckenhauer et al. [4]). *Consider an online minimization problem U, and let $\mathcal{I}(n)$ be the set of all possible inputs of length n and $I(n) \coloneqq |\mathcal{I}(n)|$. Furthermore, suppose that there is a randomized online algorithm for U with worst-case expected competitive ratio at most E. Then, for any fixed $\varepsilon > 0$, it is possible to construct a deterministic online algorithm that uses at most*

$$\log n + 2\log\log n + \log(\log I(n)/\log(1+\varepsilon)) + c$$

advice bits, for a constant c,[5] *and achieves a competitive ratio of $(1+\varepsilon)E$.* □

[5]The (small) constant c stems from rounding up the logarithms to natural numbers.

Together with this result, Theorem 7 implies that the worst-case expected color range of any randomized online algorithm is bounded from below by a value of almost 5.

Theorem 14. *For arbitrarily small $\delta > 0$, every randomized algorithm for the online $L(2,1)$-coloring problem on graphs with maximum degree 2 has a worst-case expected competitive ratio of at least $\frac{5}{4}(1-\delta)$ on sufficiently large instances.*

Proof. There are $2^{\binom{n}{2}}$ graphs of size n, and the vertices can be presented in an arbitrary order. Therefore, the number of instances $I(n)$ is bounded from above by $2^{\binom{n}{2}}n!$. Then,

$$\begin{aligned}
&\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log(1+\varepsilon)}\right) + c' \\
&\qquad \leq \log n + 2\log\log n + \log\left(\frac{\log\left(2^{\binom{n}{2}}n!\right)}{\log\frac{1}{1-\delta}}\right) + c' \\
&\qquad \leq 3\log n + 2\log\log n - \log\log\frac{1}{1-\delta} + c.
\end{aligned}$$

For any δ between 0 and 1, there is an $n_\delta \in \mathbb{N}$ such that

$$3\log n + 2\log\log n - \log\log\frac{1}{1-\delta} + c < 3.9401 \cdot 10^{-10} n,$$

for $n \geq n_\delta$. We know from Table 1 that $3.9402 \cdot 10^{-10} n$ advice bits are not sufficient to achieve a competitive ratio of $\frac{5}{4}$. Lemma 6 thus implies that no randomized online algorithm has worst-case expected competitive ratio $E = \frac{5}{4}/(1+\varepsilon) = \frac{5}{4}(1-\delta)$. □

7. Conclusion

We showed that the online $L(2,1)$-coloring problem on graphs consisting only of paths and cycles has the following interesting property. No advice at all is necessary to color a graph with seven colors, but already linear advice is necessary to improve by only one color. In other words, sublinear advice does not help at all. This is something not previously observed—many other problems allow for a smooth tradeoff between advice complexity and competitive ratio.

All our lower bounds directly carry over to more general graph classes that contain paths or cycles as special cases, e. g. trees or Hamiltonian graphs. An open problem is to improve the upper bounds or even match the lower bounds—for paths and cycles as well as for other graph classes.

References

[1] M. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, and L. Keller. Online coloring of bipartite graphs with and without advice. In *Proc. of COCOON 2012*, LNCS 7434, Springer, pp. 519–530, 2012.

[2] H. L. Bodlaender, T. Kloks, R. B. Tan, and J. van Leeuwen. Approximations for λ-colorings of graphs. *Comput. J.*, 47(2):193–204, 2004.

[3] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. On the advice complexity of the knapsack problem. In *Proc. of LATIN 2012*, LNCS 7256, Springer, pp. 61–72, 2012.

[4] H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič. On the advice complexity of the k-server problem. In *Proc. of ICALP 2011, Part I,* LNCS 6755, Springer, pp. 207–218, 2011.

[5] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In *Proc. of ISAAC 2009*, LNCS 5878, Springer, pp. 331–340, 2009.

[6] H.-J. Böckenhauer, J. Hromkovič, D. Komm, R. Královič, and P. Rossmanith. On the power of randomness versus advice in online computation. In *Language Alive: Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday*, LNCS 7300, Springer, pp. 30–43, 2012.

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[8] H. Broersma. A General Framework for Coloring Problems: Old Results, New Results, and Open Problems. In *Proc. of IJCCGGT 2003,* LNCS 3330, Springer, pp. 65–79, 2005.

[9] S. Dobrev, R. Královic, and D. Pardubská. Measuring the problem-relevant information in input. *RAIRO ITA* 43(3):585–613, 2009.

[10] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. In *Proc. of ICALP 2009, Part I*, LNCS 5555, Springer, pp. 427–438, 2009.

[11] M. Forišek, L. Keller, and M. Steinová. Advice complexity of online coloring for paths. In *Proc. of LATA 2012*, LNCS 7183, Springer, pp. 228–239, 2012.

[12] J. R. Griggs and R. K. Yeh. Labelling graphs with a condition at distance 2. *SIAM J. Discrete Math.*, 5(4):586–595, 1992.

[13] J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In *Proc. of MFCS 2010*, LNCS 6281, Springer, pp. 24–36, 2010.

[14] D. Komm and R. Královič. Advice complexity and barely random algorithms. In *Proc. of SOFSEM 2011*, LNCS 6543, Springer, pp. 332–343, 2011.

[15] D. Komm, R. Královič, and T. Mömke. On the advice complexity of the set cover problem. In *Proc. of CSR 2012*, LNCS 7353, Springer, pp. 241–252, 2012.

[16] D. Komm. Advice and Randomization in Online Computation. PhD Thesis, ETH Zurich, 2012.

[17] R. A. Murphey, P. M. Pardalos, and M. G. C. Resende. Frequency assignment problems. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization, Supplement 1*, Kluwer Academic Publishers, pp. 295–377, 1999.

[18] M. Renault and A. Rosén. On online algorithms with advice for the k-server problem. In R. Solis-Oba and G. Persiano, editors, *Approximation and Online Algorithms*, LNCS 7164, Springer, pp. 198–210, 2012.

[19] S. Seibert, A. Sprock, and W. Unger. Advice complexity of the online vertex coloring problem. In *Proc. of CIAC 2013,* LNCS 7878, pp. 345–357. Springer, 2013.

[20] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[21] R. K. Yeh. A survey on labeling graphs with a condition at distance two. *Discrete Mathematics*, 306(12):1217–1231, 2006.

A. Appendix

A.1. Case Distinction in the Proof of Lemma 3

P	P'	Concatenation	Possible colors for y	Possible color pairs for (y, z)
0-2	0-2	0-2-y-z-0-2	4	↯
0-2	2-0	0-2-y-z-2-0	4	↯
2-0	0-2	2-0-y-z-0-2	3,4	↯
2-0	2-0	2-0-y-z-2-0	3,4	↯
0-2	0-3	0-2-y-z-0-3	4	↯
0-2	3-0	0-2-y-z-3-0	4	(4,1)
2-0	0-3	2-0-y-z-0-3	3,4	(4,2)
2-0	3-0	2-0-y-z-3-0	4	(4,1)
0-2	0-4	0-2-y-z-0-4	4	↯
0-2	4-0	0-2-y-z-4-0	↯	↯
2-0	0-4	2-0-y-z-0-4	3,4	(4,2)
2-0	4-0	2-0-y-z-4-0	3	(3,1)
0-2	1-3	0-2-y-z-1-3	4	↯
0-2	3-1	0-2-y-z-3-1	4	(4,0)
2-0	1-3	2-0-y-z-1-3	3,4	↯
2-0	3-1	2-0-y-z-3-1	4	↯
0-2	1-4	0-2-y-z-1-4	4	↯
0-2	4-1	0-2-y-z-4-1	↯	↯
2-0	1-4	2-0-y-z-1-4	3,4	↯
2-0	4-1	2-0-y-z-4-1	3	↯
0-2	2-4	0-2-y-z-2-4	4	(4,0)
0-2	4-2	0-2-y-z-4-2	↯	↯
2-0	2-4	2-0-y-z-2-4	3,4	↯
2-0	4-2	2-0-y-z-4-2	3	(3,1)
0-3	0-3	0-3-y-z-0-3	1	(1,4)
0-3	3-0	0-3-y-z-3-0	1	↯
3-0	0-3	3-0-y-z-0-3	2,4	(2,4), (4,2)
3-0	3-0	3-0-y-z-3-0	2,4	(4,1)
0-3	0-4	0-3-y-z-0-4	1	↯
0-3	4-0	0-3-y-z-4-0	1	↯
3-0	0-4	3-0-y-z-0-4	2,4	(4,2)
3-0	4-0	3-0-y-z-4-0	2	↯
0-3	1-3	0-3-y-z-1-3	↯	↯
0-3	3-1	0-3-y-z-3-1	1	↯
3-0	1-3	3-0-y-z-1-3	2,4	(2,4)
3-0	3-1	3-0-y-z-3-1	2,4	↯
0-3	1-4	0-3-y-z-1-4	↯	↯
0-3	4-1	0-3-y-z-4-1	1	↯
3-0	1-4	3-0-y-z-1-4	2,4	↯
3-0	4-1	3-0-y-z-4-1	2	↯
0-3	2-4	0-3-y-z-2-4	1	↯
0-3	4-2	0-3-y-z-4-2	1	↯
3-0	2-4	3-0-y-z-2-4	4	↯
3-0	4-2	3-0-y-z-4-2	2	↯
0-4	0-4	0-4-y-z-0-4	1,2	(1,3)

P	P'	Concatenation	Possible colors for y	Possible color pairs for (y, z)
0-4	4-0	0-4-y-z-4-0	1,2	↯
4-0	0-4	4-0-y-z-0-4	2,3	↯
4-0	4-0	4-0-y-z-4-0	2,3	(3,1)
0-4	1-3	0-4-y-z-1-3	2	↯
0-4	3-1	0-4-y-z-3-1	1,2	(2,0)
4-0	1-3	4-0-y-z-1-3	2,3	(2,4)
4-0	3-1	4-0-y-z-3-1	2	↯
0-4	1-4	0-4-y-z-1-4	2	↯
0-4	4-1	0-4-y-z-4-1	1,2	(2,0)
4-0	1-4	4-0-y-z-1-4	2,3	↯
4-0	4-1	4-0-y-z-4-1	2,3	↯
0-4	2-4	0-4-y-z-2-4	1	↯
0-4	4-2	0-4-y-z-4-2	1,2	(2,0)
4-0	2-4	4-0-y-z-2-4	3	↯
4-0	4-2	4-0-y-z-4-2	2,3	(3,1)
1-3	1-3	1-3-y-z-1-3	0	(0,4)
1-3	3-1	1-3-y-z-3-1	0	↯
3-1	1-3	3-1-y-z-1-3	4	↯
3-1	3-1	3-1-y-z-3-1	4	(4,0)
1-3	1-4	1-3-y-z-1-4	0	↯
1-3	4-1	1-3-y-z-4-1	0	(0,2)
3-1	1-4	3-1-y-z-1-4	4	↯
3-1	4-1	3-1-y-z-4-1	↯	↯
1-3	2-4	1-3-y-z-2-4	0	↯
1-3	4-2	1-3-y-z-4-2	0	↯
3-1	2-4	3-1-y-z-2-4	4	(4,0)
3-1	4-2	3-1-y-z-4-2	↯	↯
1-4	1-4	1-4-y-z-1-4	0,2	(0,3)
1-4	4-1	1-4-y-z-4-1	0,2	(0,2), (2,0)
4-1	1-4	4-1-y-z-1-4	3	↯
4-1	4-1	4-1-y-z-4-1	3	(3,0)
1-4	2-4	1-4-y-z-2-4	0	↯
1-4	4-2	1-4-y-z-4-2	0,2	(2,0)
4-1	2-4	4-1-y-z-2-4	3	(3,0)
4-1	4-2	4-1-y-z-4-2	3	(3,0)
2-4	2-4	2-4-y-z-2-4	0,1	↯
2-4	4-2	2-4-y-z-4-2	0,1	↯
4-2	2-4	4-2-y-z-2-4	0	↯
4-2	4-2	4-2-y-z-4-2	0	↯

Table A.5: All possible combinations of tails.

A.2. How to Color the Gaps in Algorithm 3

	3-0-	5-0-	0-3-	5-3-	0-5-	3-5-	2-0-	4-0-	0-2-	4-2-	0-4-	2-4-
-0-3	1-5 ↯ ①	↯ ② 5-0- 3	1-5 5-1- 4	↯ ③ 5-0- 2	1-4 5-1- 4	5-0 1-5- 0	1-5 5-1- 4	5-1 1-5- 2	1-4 5-1- 4	5-1 1-5- 0	1-5 5-1- 3	1-5 1-5- 0
-0-5		3-1 3-0- 2	1-4 3-1- 4	3-0 3-0- 2	1-4 3-1- 4	2-0 1-4- 0	1-4 1-3- 5	3-1 3-0- 2	2-4 3-1- 4	3-1 1-3- 0	1-3 1-4- 2	3-0 3-1- 5
-3-0			2-5 4-1- 5	4-2 5-3- 0	4-2 4-1- 3	4-1 2-5- 1	↯ ④ 5-1- 4	5-1 5-3- 1	5-3 5-1- 4	5-1 5-3- 1	5-3 5-1- 3	↯ ⑤ 2-5- 0
-3-5				2-0 2-4- 0	1-3 0-2- 4	2-0 2-4- 0	1-4 2-0- 4	0-2 0-3- 1	2-4 0-2- 4	2-0 1-3- 0	1-3 1-4- 2	↯ ⑥ 1-3- 0
-5-0					2-4 3-1- 4	4-1 4-2- 0	3-5 3-1- 4	3-1 2-5- 1	2-4 3-1- 4	3-1 3-5- 1	4-2 3-5- 2	3-5 3-1- 5
-5-3						↯ ⑦ 1-4- 0	0-4 ↯ ⑧	0-2 0-5- 2	1-4 1-5- 3	↯ ⑨ 0-5- 1	1-5 1-4- 2	1-5 1-4- 0
-0-2							↯ ⑩ 4-0- 5	5-1 5-3- 1	5-3 5-1- 3	5-1 5-3- 1	5-3 5-1- 3	4-0 4-1- 5
-0-4								↯ ⑪ 1-5- 2	1-3 1-5- 3	2-0 2-5- 0	1-5 1-3- 5	1-5 1-3- 5
-2-0									3-5 3-1- 5	5-1 5-3- 1	5-3 5-1- 3	3-5 3-1- 5
-2-4										↯ ⑫ 1-5- 0	1-3 1-5- 3	1-5 1-3- 5
-4-0											3-5 3-1- 5	3-5 3-1- 5
-4-2												5-0 5-3- 0

Table A.6: All possible tails that can occur in line 22 of Algorithm 3. The circled numbers correspond to the numbering in Table 4.