Advice Complexity of Online Graph Coloring[☆]

Michal Forišek[a,*], Lucia Keller[b], Monika Steinová[b]

[a]*Department of Computer Science; Faculty of Mathematics, Physics, and Informatics; Comenius University; Mlynska dolina; 842 48 Bratislava; Slovakia*
[b]*ETH Zurich; Switzerland*

Abstract

In online graph coloring, a graph is revealed to an online algorithm one vertex at a time, and the algorithm must color the vertices as they appear. This paper investigates the advice complexity of this problem – the amount of oracle information an online algorithm needs in order to make optimal choices. We formally define a classification of online graph coloring variants and use advice complexity to analyze and compare the difficulty of solving them. In the paper we obtain tight bounds for general graphs (under various vertex presentation orders) and also for some of the simplest graph classes: paths, cycles, and spider graphs.

Keywords: advice complexity, online graph coloring

[☆]A preliminary version of this paper, containing just the results for paths, was presented at LATA 2012 [1].

[*]Corresponding author.

Email addresses: `forisek@dcs.fmph.uniba.sk` (Michal Forišek), `lucia.keller@inf.ethz.ch` (Lucia Keller), `monika.steinova@inf.ethz.ch` (Monika Steinová)

1. Overview of advice complexity

A great challenge in classical algorithmics are problems that work in an online fashion: The instance is not shown to the algorithm all at once. Instead, the algorithm receives it piecewise in consecutive turns. In each turn, the algorithm must produce a piece of the output which must not be changed afterwards. Such algorithms are called *online algorithms*, more on them can be found for example in [2].

Obviously, it is harder (and sometimes even impossible) to compute the best partial solutions without knowing the future. The output quality of online algorithms is measured by the *competitive ratio* of a particular algorithm. For a given instance, the ratio is the quotient of the cost of the solution produced by the given online algorithm and the cost of an optimal solution (i.e., the cost of the output of an optimal offline algorithm for that particular instance). We are usually interested in the worst competitive ratio over all instances. For randomized online algorithms we consider the *expected* competitive ratio. See [2, 3] for more details.

The exact advantage of the offline algorithm can be measured as the amount of additional information, "advice", the online algorithm needs in order to produce an optimal solution. This notion can be formalized by providing the online algorithm with an additional source of information: an advice tape. The online algorithm may access this tape at any time and use its contents as additional information for computing the partial solutions. The content of the advice tape is assumed to be prepared in advance by an oracle which has unlimited computational power and has access to the whole input instance.

This model of advice complexity has first been introduced in [4]. The original model had a technical disadvantage: the advice tape was finite, hence its length could have been used to encode additional information. There were two attempts to fix this. In [5], the authors force the algorithm to use a fixed number of advice bits in every turn. A drawback of this approach is that it is not possible to determine sublinear advice complexity, or investigate situations in which the need for advice is not uniform. A better version of the model was proposed in [6]. In that version, the advice tape is considered to be infinite, and the amount of advice used is defined as the length of its prefix that is actually examined by the online algorithm. We are using this model in our paper. In Section 3 we give a brief summary of the formal definitions.

There are indeed problems where already a small amount of information is enough for an online algorithm to be optimal. For example, in the simplest online problem SkiRental we need just a single bit of advice in order to solve it optimally [4]. In recent years, advice complexity was also investigated for other classical online problems, such as the k-server problem [7] and the paging problem [8].

Advice complexity has its theoretical importance in measuring an exact quantity of information that directly characterizes the hardness of an online problem compared to its offline version. This can bring new insights even into well-researched topics. For instance, in the paging problem it is well known that online algorithms perform well in practice, but on the other hand, the worst-case competive ratio for any deterministic online algorithm for Paging is k, where k is the cache size. (I.e., for any deterministic online algorithm

there is an instance such that the optimal offline algorithm produces k times less page faults.) This dichotomy can be explained by an insight gained from advice complexity. For an instance with n page requests, one needs $n \lg k$ bits to encode a particular optimal solution. However, in [6] it is shown that:

- With 2 advice bits we can guarantee a competitive ratio of $k/2 + O(1)$.
- $O(\log k)$ advice bits are enough to guarantee a competitive ratio $O(\log k)$, which is asymptotically equal to optimal randomized algorithms.
- As few as $n+k$ advice bits are sufficient to produce an optimal solution.

These results show that the amount of information lost in transition from the offline to the online version of this problem is significantly lower than expected – in other words, even before the first request, the online solver already has to have a lot of information about the set of all optimal solutions.

It is also worth noting that advice complexity is closely related to both non-determinism (in terms of the oracle) and randomization. In [9], relations between advice complexity and randomized algorithms are shown, and a new randomized algorithm is designed based on a careful computation of the advice complexity of a given problem. This makes advice complexity a new and important point of view on online algorithms.

Advice complexity of graph coloring

A preliminary version of this paper [1] (with just the results for paths) was presented at LATA 2012 as the first published paper dealing with the advice complexity of graph coloring.

Bianchi et al. [10] address online coloring of bipartite graphs (using an arbitrary presentation order, see Section 2 of this paper). As one of their results, the authors analyze the advice complexity of this problem, showing that the advice complexity of constructing an optimal solution is either $n-2$ or $n-3$ bits. Furthermore, there is a trade-off between the amount of advice and the quality of the solution: for any given integer $k > 2$, having $n/\sqrt{2^{k-1}}$ advice bits is sufficient to guarantee that the online algorithm with advice will produce a valid coloring that uses at most k colors (and therefore has a competitive ratio $k/2$).

2. Online graph coloring

The term "online graph coloring" actually denotes a huge class of computational problems. The general offline version of this problem is a well-known NP-complete problem, as are many of its simpler variants. All online algorithms without advice are known to be extremely poor in the worst case [11]. This makes this problem very interesting from the advice complexity point of view. The primary goal of this paper is to classify some of the most important variants of online graph coloring and to analyze the advice complexity of some of those variants.

A huge number of practical problems can be modeled as some type of online graph coloring [12]. The computational complexity of a particular problem depends on multiple mutually orthogonal aspects. We will now present the most important ones.

First of all, we need to identify the **class of graphs** that may appear as instances. Instead of coloring general, arbitrary graphs it may be the case

that we are guaranteed that the input graph is somehow special – e.g., planar, sparse, bipartite, a tree, or even a path. The complexity of graph coloring (both online and offline) is clearly influenced by this additional information.

The second aspect is the **presentation order**. This aspect only matters in online coloring. In the general case we know nothing about the order in which the vertices of the graph are presented to the online algorithm. This does not always have to be the case in practice. And if we know something about the presentation order, this, once again, can be seen as additional information that may make the problem easier to solve. (And advice complexity offers us a natural way of measuring the *amount* of this information.) In particular, an interesting presentation order is the connected presentation order in which each vertex after the first one is adjacent to at least one of the previous vertices. And as special cases of this presentation order, we have DFS and BFS order (i.e., the order in which a depth-first or a breadth-first search would visit the vertices). Another common presentation order is one commonly used as a heuristic in the offline setting: vertices are presented ordered according to their degrees, starting with the largest one.[1]

These two main aspects define a particular computational problem. In the rest of this paper, we always specify both of them. For instance, OnlineColoring(planar,bfs) denotes the problem of coloring planar graphs with vertices presented in BFS order, and OnlineColoring(any,any) is the most general version of online graph coloring.

The third aspect is the **quality of the solution** we need. This does

[1]Several other presentation orders are interesting from a theoretical or practical point of view. Some additional examples are considered in [11].

not influence the definition of the problem, but it does influence the solution. Exact graph coloring is hard, but sometimes in practice a good approximation is all we need. This is captured in theory by the approximation ratio (in the offline setting) and the competitive ratio (in the offline setting). In terms of advice complexity, we will be looking at trade-offs between the amount of advice available and the competitive ratio that can be guaranteed.

There are also other minor aspects worth taking into consideration. For instance, we may care about the time complexity of the online algorithm. In some cases there may be a trade-off between the amount of available advice and the time complexity we are able to guarantee. In this paper we will not pursue this line of thought any further; but we note that all online algorithms presented in upper bound proofs in this paper have a polynomial time complexity.

3. Notation and definitions

Whenever we consider a fixed problem, the cost of a particular solution S will be denoted by $C(S)$. The output of an algorithm A for an instance I will be denoted by $A(I)$. Hence $C(A(I))$ denotes the cost of the solution that algorithm A produced for the instance I. The optimal solution for I will be denoted by $Opt(I)$.

The expected value of a random variable X will be denoted by $E[X]$.

Definition 1. *Consider an optimization problem in which the goal is to minimize the cost of a solution. An algorithm is* c-competitive *if there is a constant* α *such that for each instance* I *we have* $C(A(I)) \leq c \cdot C(Opt(I)) + \alpha$.

If $\alpha = 0$, we say that A is strictly c-competitive. *The* competitive ratio *of A is the smallest c such that A is c-competitive.*

The above definition can be extended to randomized algorithms: We require that for each instance I we must have $E[C(A(I))] \leq c \cdot C(Opt(I)) + \alpha$. Afterwards we claim that the *expected* competitive ratio of A (against an oblivious adversary) is the smallest corresponding c.

The following definitions are rephrased from [6].

Definition 2. *An* online algorithm A with advice *is defined as follows: The input for the algorithm is a sequence $X = (x_1, \ldots, x_n)$ and an infinite advice string $\varphi \in \{0,1\}^\omega$. The algorithm produces an output sequence $Y = (y_1, \ldots, y_n)$ with the restriction that, for all i, y_i is computed only from $x_1, \ldots, x_i$ and φ. This is denoted by $A^\varphi(X) = Y$.*

The computation of A can be seen as a sequence of turns, where in the i-th turn A reads x_i and then produces y_i using all the information read so far and possibly some new bits of the advice string. Note that the definition does not restrict the computational power of the online algorithms. Still, all of the algorithms in our paper will be deterministic and they will all have a polynomial time complexity.

Definition 3. *The* advice complexity of A *is a function s such that $s(n)$ is the smallest value such that for no input sequence of size n the algorithm A examines more than the first $s(n)$ bits of the advice string φ. The* advice complexity of an online problem *is the smallest advice complexity an online algorithm with advice needs to produce an optimal solution (i.e., a solution as good as an optimal offline algorithm would produce).*

Definition 4. *An online algorithm with advice A is* c-competitive *if there is a constant α such that for every $n \in \mathbb{N}$ and for every instance I of size at most n there is an advice string φ such that $C(A^{\varphi}(I)) \leq c \cdot C(Opt(I)) + \alpha$.*

Again, the above definition extends to randomized algorithms with advice in a natural way.

When defining a particular variant of online graph coloring, we have to specify the class of graphs and presentation order. For the class of graphs we will use its common name (e.g., "TREE", "PLANAR"), with the addition that "ANY" denotes that all possible graphs are valid instances. For the presentation orders defined above we will use the names "ANY", "CONNECTED", "BFS", "DFS", and "MAX-DEGREE".

Definition 5. *In* ONLINECOLORING *the instance is an undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$. This graph is presented to an online algorithm in turns: In the k-th turn the online algorithm receives the graph $G_k = G[\,\{1, 2, \ldots, k\}\,]$, i.e., a subgraph of G induced by the vertex set $\{1, 2, \ldots, k\}$. As its reply, the online algorithm must return a positive integer: the color it wants to assign to vertex k. The goal is to produce an optimal coloring of G – the online algorithm must assign distinct integers to adjacent vertices, and the largest integer used must be as small as possible.*

Vertex labels in G correspond to the order in which the online algorithm is asked to color the vertices. As an equivalent definition, we may assume that in the k-th turn the online algorithm gets a list of edges between k and vertices in $\{1, 2, \ldots, k-1\}$. In ONLINECOLORING(X,Y) we have the additional information that G belongs to the class X and the presentation order belongs to the set Y.

Note that the value n is not known a priori by the online algorithm. This is intentional and necessary. Announcing the value n to the online algorithm would give it additional information about the instance it will be processing, and this amount of information would then not be reflected in the advice complexity.

Selected bounds for combinatorial sequences

Below we define the notation for several combinatorial sequences and present the bounds we will be using in our estimates.

In the rest of the paper "lg" is the base-2 (binary) logarithm, and "ln" is the base-e (natural) logarithm.

Bell numbers count the number of ways a set of n distinct elements can be partitioned into arbitrarily many nonempty subsets. They will be denoted by $B(n)$. De Bruijn [13] gives an asymptotic estimate for Bell numbers that can be simplified to $\lg B(n) = n \lg n - n \lg \lg n + O(n)$.

Stirling numbers of the second kind count the number of ways a set of n distinct elements can be partitioned into exactly k nonempty subsets. They will be denoted by $S(n, k)$.

Lemma 1. *The value* $\max_k(\lg S(n,k))$ *is asymptotically equal to* $n \lg n - n \lg \lg n + O(n)$. *The* k *for which* $\lg S(n,k)$ *is maximized is* $k = n/\ln n + O(1)$.

Proof. Clearly for all $n > 0$, $B(n) = \sum_{k=1}^{n} S(n,k)$. Hence there is some k such that $S(n,k) \geq B(n)/n$, and the first claim follows trivially. In [14] it is shown that for all sufficiently large n the k where the maximum occurs is within 1 of $e^r - 1$, where $re^r = n$. In [13] it is shown that $r = \ln n - \ln \ln n + O(\ln \ln n / \ln n)$. From this we can easily derive that $k = n/\ln n + O(1)$. □

4. General graphs

In this section we consider the problems OnlineColoring(any,·) from the point of view of their advice complexity. We show that for each of our presentation modes the amount of advice needed to produce an optimal solution is proportional to the size of the encoding of a particular optimal solution. This result can be seen as an explanation why online graph coloring is so hard – the online algorithm knows almost nothing useful about the problem it aims to solve.

Theorem 2. *The advice complexity of* OnlineColoring(any,any) *is at most* $n \lg n - n \lg \lg n + O(n)$.

Proof. Suppose that the instance has n vertices. The oracle producing the advice tape will pick one optimal coloring. This coloring determines one of the $B(n)$ ways to partition the vertices into nonempty subsets. The advice tape will contain n and the number of the correct partitioning in lexicographic order. This can be done using $2 \lg n + \lg B(n)$ bits, which can be estimated as $n \lg n - n \lg \lg n + O(n)$. □

Note that the oracle could also encode the number k. Such an encoding would have been slightly better, as instead of $\lg B(n)$ we would have the binary logarithm of a Stirling number of the second kind. Still, the asymptotic estimate would remain the same.

Corollary 3. *The advice complexity of* OnlineColoring(any,x) *is at most* $n \lg n - n \lg \lg n + O(n)$ *for any presentation order* x.

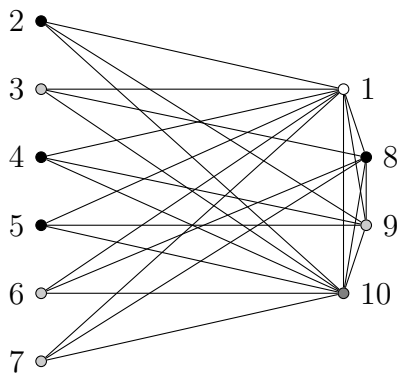

Figure 1: One of the instances for $n = 10$, $m = 6$, and $k = 3$.

Theorem 4. *The advice complexity of* OnlineColoring(any,bfs) *is at least* $n \lg n - n \lg \lg n + O(n)$.

Proof. For a fixed n we will construct a set of instances $\mathcal{B}_n$ as follows: Let $m(n) = n - n/\ln n$, and let $k(n)$ be the number of subsets (i.e., colors) for which $S(m(n), k(n))$ is maximized. From Lemma 1 we have $k(n) = m(n)/\lg(m(n)) + O(1)$, hence $m(n) + k(n) + 1 \leq n$ for sufficiently large n. Below we consider a fixed n and write m and k instead of $m(n)$ and $k(n)$.

The set $\mathcal{B}_n$ will contain precisely $S(m, k)$ instances – one for each way in which we can partition m elements into exactly k nonempty subsets.

Each instance has vertices 1 through $m+k+1$. Vertex 1 is connected to all vertices. The subgraph induced by vertices $\{2, \ldots, m+1\}$ is an independent set. These vertices represent the elements. The subgraph induced by vertices $\{m+2, \ldots, m+k+1\}$ is a clique. These vertices represent the subsets, in any particular order. Each vertex in $\{2, \ldots, m+1\}$ is connected to all vertices in $\{m+2, \ldots, m+k+1\}$ except for one – the one representing the subset containing the particular element. A sample instance is shown in Figure 1.

The order in which the vertices of an instance are labeled corresponds to one possible BFS traversal order. For any instance in $\mathcal{B}_n$, each optimal coloring uses exactly $k+1$ colors and corresponds to the chosen partitioning of the m elements into nonempty subsets. Hence no two instances in $\mathcal{B}_n$ share the same optimal coloring.

For all instances in $\mathcal{B}_n$ the subgraphs induced by vertices $\{1, 2, \ldots, m+1\}$ are identical. Hence the only source of information when coloring the vertices 2 through $m+1$ is the advice tape. In order to produce the optimal coloring, the online algorithm must know the correct partitioning of those vertices. Hence $\lg S(m,k)$ advice bits are necessary.

Using Lemma 1 to estimate $S(m,k)$ and the fact that $\lg(n - n/\ln n) = \lg n + O(1)$, we get:

$$\begin{aligned}\lg S(m,k) &= \left(n - \frac{n}{\ln n}\right)\lg n - \left(n - \frac{n}{\ln n}\right)\lg\lg n + O(n) \\ &= n\lg n - n\lg\lg n + O(n)\end{aligned}$$

□

Corollary 5. *The advice complexity of* OnlineColoring(any,any) *and* OnlineColoring(any,connected) *is at least* $n\lg n - n\lg\lg n + O(n)$.

The results can be summarized as follows: For the presentation orders we considered, the knowledge that a particular presentation order is used does not contain a significant amount of information. The problem still remains approximately as hard as in the most general setting.

5. Paths

In this section, we present our results on advice complexity for online coloring on paths.

Lemma 6. *Let X be a class of graphs such that each graph in X is 2-colorable. An online algorithm with advice for* OnlineColoring(X,·) *never needs to access advice bits whenever the degree of the currently processed vertex k in the current graph G_k is positive.*

Proof. If the degree of k in G_k is positive, in G the vertex a_k must be adjacent to some a_i such that $i < k$. The online algorithm already assigned a color to a_i, and now it must use the other color for a_k. This can be done without advice. □

Lemma 7. *There is a deterministic online algorithm solving* OnlineColoring(path,any) *with advice complexity $\lceil n/2 \rceil - 1$.*

Proof. As suggested by Lemma 6, our algorithm only asks for advice (i.e., reads the next bit of the advice string) whenever the current vertex k is isolated in G_k. One bit of advice is sufficient – the advice can be interpreted as the correct color to use. The above only applies for $k > 1$, as we may pick an arbitrary color for the first isolated vertex.

Let S be the set of vertices that were isolated at the moment when we processed them. Clearly, it follows that no two of them are adjacent in G, hence S is an independent set in G and therefore $|S| \leq \lceil n/2 \rceil$. □

Lemma 8. *Any deterministic online algorithm solving* OnlineColoring(path,any) *needs at least $\lceil n/2 \rceil - 1$ bits of advice in the worst case.*

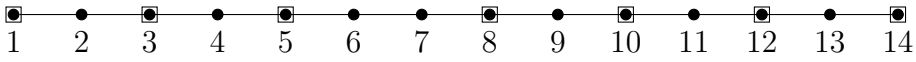

Figure 2: Example for $n = 14$ and $x = 3$: $P_x = \{1, 3, 5\}$ and $Q_x = \{8, 10, 12, 14\}$.

PROOF. Note that the proof of the lower bound of $\lfloor n/2 \rfloor - 1$ bits is reasonably simple; for odd n we have to use a more careful analysis to force the extra bit.

We will denote the vertices $v_1, \ldots, v_n$ in the order in which they appear on the path: for all i the vertices v_i and v_{i+1} are adjacent. Note that we do not know the exact numbers of these vertices. (Each path produces two such sequences. But in our proof we only consider sequences where $v_1 = 1$, so different sequences $(v_1, \ldots, v_n)$ indeed correspond to different graphs G.)

Let $k = \lfloor n/2 \rfloor$. The graph G_k will be called the prefix of an instance. We only consider instances where the prefix consists of k isolated vertices. Out of these instances, we pick a set of instances S with the following property: for no two instances in S can their prefixes be colored in the same way. (Note that each instance has exactly two valid colorings, hence the prefix of each instance also has exactly two valid colorings – one the complement of the other.) As for a deterministic algorithm the prefixes of instances in S are indistinguishable, all information about their correct coloring must be given as advice.

For any x ($1 \leq x \leq \lfloor n/2 \rfloor$) consider the two sets of positions on the path $P_x = \{2i - 1 \mid 1 \leq i \leq x\}$ and $Q_x = \{2i \mid x + 1 \leq i \leq \lfloor n/2 \rfloor\}$ (see Fig. 2). Note that all vertices v_i for $i \in P_x$ must share one color, and all vertices v_j for $j \in Q_x$ must share the other color. (Also note, that P_x and Q_x are sets of indices of vertices v_i and not their labels.)

Let I_x be the set of all instances where the vertices on positions in $P_x \cup Q_x$ form the prefix. Formally, in these instances $\forall i \in P_x \cup Q_x : v_i \leq k$. Note that for any such instance the prefix indeed consists of k isolated vertices.

We will now define the set S: Consider all strings $w \in \{\mathtt{p}\} \cdot \{\mathtt{p}, \mathtt{q}\}^{k-1}$. For each such string, we include into S a single instance: Let x be the number of '$\mathtt{p}$'s in w. We will pick the lexicographically smallest[2] instance from I_x such that for all vertices $v_i \leq k$ we have ($i \in P_x$ iff the i-th letter of w is $\mathtt{p}$). In other words, the string w determines the value of x and gives the order in which vertices from P_x and Q_x are picked for coloring. There is always at least one such instance; if there are multiple such instances, any will do, so we pick the lexicographically smallest one.

In this way we constructed a set S of 2^{k-1} instances (S contains one instance per each string $\{\mathtt{p}\} \cdot \{\mathtt{p}, \mathtt{q}\}^{\mathtt{k-1}}$) such that for no two instances in S the prefix can be colored in the same way. By the pigeon-hole principle, if there were a deterministic online algorithm that always uses less than $k-1$ bits of advice, two of the instances would receive the same advice, hence the algorithm would produce the same coloring of their prefixes, which is a contradiction. Therefore any deterministic online algorithm needs at least $k - 1 = \lfloor n/2 \rfloor - 1$ bits of advice. That concludes the proof for even n.

For odd n, we want to prove by contradiction that any deterministic online algorithm must use at least k bits of advice. Assume that the algorithm always uses less than k bits of advice for paths of length n. This means that on instances from S the algorithm always reads all $k-1$ bits of advice, and

[2]I.e., one for which the vector $(v_1, v_2, \dots)$ is the lexicographically smallest.

different instances in S must correspond to different $k-1$ bits of advice.

In S we have an instance J_1 that corresponds to $\mathtt{p}^k$. For this instance all vertices in the prefix must receive the same color. Let φ_1 be the first $k-1$ bits of advice for this instance. Consider any instance (possibly with more than n vertices) such that G_k consists of isolated vertices that should receive the same color as J_1. Clearly, for any such instance the first $k-1$ bits of advice must be φ_1 – otherwise the deterministic algorithm would color the first k vertices in a different way.

Now consider one additional instance J_2: the lexicographically smallest one where $v_i \leq k+1$ iff i is odd. (This instance is similar to J_1, but in J_1 we have $v_2 = k+1$ and in J_2 we have $v_n = k+1$. For J_2 the graph G_{k+1} has $k+1$ isolated vertices.) As our algorithm never uses k bits of advice, it must process v_n without any additional advice. Thus the instances J_1 and J_2 are undistinguishable for the algorithm and the algorithm cannot use extra bits of advice to color J_2. Hence whenever our algorithm is presented with an instance (of any size) such that the first k vertices are isolated and must share the same color, it will color the next isolated vertex using the same color. And this is a contradiction: we can easily create an instance of size $n+1$ where the $(k+1)$-st isolated vertex should have the opposite color. □

Theorem 9. *The advice complexity of* OnlineColoring(path,any) *is exactly* $\lceil n/2 \rceil - 1$ *bits.*

Proof. Follows immediately from Lemma 7 and Lemma 8. □

Theorem 10. *The advice complexity of* OnlineColoring(path,connected) *is zero.*

PROOF. Follows trivially from Lemma 6. □

We conclude this section with a consideration of a special presentation order that lies somewhere between the two orders considered above. The problem considered below can also be seen from a different angle – not as a special case of a presentation order, but as a special case of a partially online problem.[3]

In this presentation order, denoted by 2PASS, the path is traversed twice from one end to the other, and each vertex is processed online during one of the passes. Of course, by Lemma 6 no advice is necessary in the second pass. Hence the alternate interpretation mentioned above – an equivalent statement would be that some of the vertices are processed online in the first pass, and the rest is processed offline, all at once. (These two points of view are only equivalent for paths, for more complicated graph classes their generalizations will differ.)

Theorem 11. *The advice complexity of* ONLINECOLORING(PATH,2PASS) *is* $\beta n + O(\log n)$*, where* $\beta \approx 0.4057$ *is the binary logarithm of the plastic constant (i.e., of the only real root of the polynomial* $x^3 - x - 1$*).*

PROOF. We gave a detailed proof in [1]. As this result lies slightly beyond

[3]In partially online problems only a part of an instance is processed online, the rest is then processed offline (as if all remaining requests appeared at the same time). One practical motivation are situations with a fixed deadline. As an example, note that most authors submit camera ready copies of conference papers only on the day of the deadline. Processing them to produce the conference proceedings can be seen as an example of a partially online version of a lot sizing problem.

the scope of this paper, we opted to just present an outline of the proof here, and refer an interested reader to [1].

For the lower bound we identify a large set of instances that cannot be distinguished by the online algorithm and require pairwise distinct colorings. This set of instances is defined by two parameters k and l and contains all instances in which:

- the first vertex of the path is processed in the first pass,
- every other vertex processed in the first pass lies in distance 2 or 3 from the previous one, and
- there are k steps of length 2 and l steps of length 3.

For an n-vertex path, taking $k = l = \lfloor (n-1)/5 \rfloor$ gives us the lower bound of $0.4n$ advice bits. However, we computed that the optimal choice of k and l is slightly asymmetric, yielding the lower bound of $\beta n - \lg n + O(1)$ advice bits.

For the upper bound we show how to map any instance to one of the instances of the type described in the lower bound, and then use its lexicographic index as advice. In this way we obtain the upper bound of $\beta n + 2 \lg n + \lg \lg n + O(1)$. □

6. Cycles

In this section, we consider the advice complexity of online coloring for cycles. Cycles with an even number of vertices can be optimally colored using 2 colors and cycles with an odd number of vertices using 3 colors. In the following, we will call these two cycle types "even cycles" and "odd cycles".

Lemma 12. *There is a deterministic online algorithm that solves the problem* OnlineColoring(odd-cycle,any) *without using advice.*

Proof. A greedy strategy works. As we are allowed to use three colors and each vertex has at most 2 neighbors, we always have at least one color we may use. □

Lemma 13. *There is a deterministic online algorithm solving* OnlineColoring(cycle,any) *with advice complexity* $\lfloor n/2 \rfloor - 1$.

Proof. We will use the following greedy algorithm: Color the first vertex arbitrarily. In the rest of the instance, whenever you are shown an isolated vertex, use an advice bit to color it 1 or 2. Whenever you are shown a vertex that is not isolated, use the smallest available color (1, 2, or 3).

In an instance that is a cycle with n vertices there can be at most $\lfloor n/2 \rfloor$ isolated vertices, hence this algorithm always uses at most $\lfloor n/2 \rfloor - 1$ advice bits. It is easily verified that for any instance there is an advice string such that the algorithm produces an optimal coloring – using two colors for even n, three for odd n. (For even n there is one such advice string – the correct coloring. For odd n all possible advice strings work.) □

Corollary 14. *There is a deterministic online algorithm solving* OnlineColoring(even-cycle,any) *with advice complexity* $n/2 - 1$.

Lemma 15. *Any deterministic online algorithm solving the problem* OnlineColoring(even-cycle,any) *needs at least* $n/2 - 1$ *bits of advice in the worst case.*

PROOF. Observe that optimal colorings for paths and even cycles look almost the same – they use two alternating colors. We will use this fact to show that any algorithm solving ONLINECOLORING(EVEN-CYCLE,ANY) can be used to solve ONLINECOLORING(PATH,ANY).

Assume that there exists a deterministic online algorithm A_C with advice, solving ONLINECOLORING(EVEN-CYCLE,ANY) with advice complexity $f(n)$. We will use A_C as a subroutine to define a deterministic online algorithm A_P with advice, solving ONLINECOLORING(PATH,ANY) with the same advice complexity $f(n)$.

Suppose that A_P is given a path on m vertices. Let $n = m$ if m is even, and $n = m + 1$ if m is odd. The algorithm A_P will interpret all incoming isolated vertices as vertices on a cycle with n vertices, and process them by calling A_C. The remaining vertices can be handled directly by A_P (using Lemma 6).

Clearly any valid coloring of the path corresponds to a valid coloring of the cycle and vice versa. So the coloring A_C constructs for the cycle directly maps to a valid coloring of the path. Note that A_P does not know m and A_C does not know n. The algorithm A_P never actually computes n, it just checks each vertex for adjacency to the previously processed vertices.

As A_P does not use any additional advice bits, the advice complexity of A_P for a path with m vertices is equal to the advice complexity of A_C for a cycle with n vertices. By Lemma 8 we know that $\lceil m/2 \rceil - 1$ advice bits are necessary for a path with m vertices. Hence $f(n) \geq \lceil m/2 \rceil - 1 = n/2 - 1$. $\square$

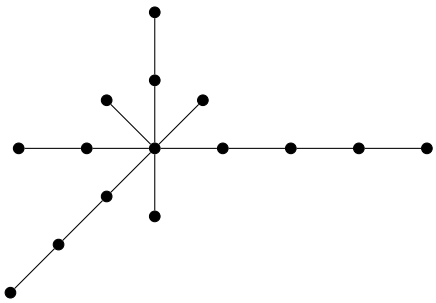

Figure 3: Example of a spider graph.

7. Spider graphs

In this section, we will consider special trees where all vertices except for one have degree at most 2. We call those graphs spider graphs (see [15]). Figure 3 shows an example. The vertex with the degree larger than 2 is referred to as the *center* of the spider and the paths from the center are called the *legs* of the spider graph.

Lemma 16. *Spider graphs are 2-colorable.*

Proof. Spider graphs are trees, and trees are bipartite. □

Lemma 17. *There is an online algorithm without advice that can color any spider graph using at most 3 colors.*

Proof. The naive greedy strategy "always use the smallest available color" has this property. Any vertex other than the center has at most two neighbours, so it will surely receive a valid color. Now consider the turn t when the online algorithm processes the center c. Some neighbours of the center may have been processed before the turn t. Consider any such neighbour v. At the moment when v was processed, it had at most one processed neighbour –

because it has at most two, and c was not processed yet. Therefore v surely received one of the colors 1 and 2. But this means that it is certainly possible to color the center using the color 3. □

Note that from Lemma 17 it follows that for spider graphs it still does not make much sense to consider the tradeoff between advice size and the competitive ratio of the online algorithm. For spider graphs, a ratio 3/2 can be achieved without advice, and we are only able to guarantee a better competive ratio by solving the problem optimally.

Definition 8. *A spider graph is called a* 1-2-spider graph *if each leg consists of at most two vertices. A 1-2-spider graph is* partially labeled *if the vertices at the ends of its legs are labeled 1 through ℓ (where ℓ is the number of legs), the center is labeled n (where n is the total number of vertices) and the other vertices are unlabeled.*

Lemma 18. *For each $n \geq 1$, there are exactly F_n partially labeled 1-2-spider graphs on n vertices, where F_n is the n-th Fibonacci number.*

Proof. Let $v(n)$ be the number of partially labeled 1-2-spider graphs with n vertices. If $n = 1$, the spider graph consists of just the center, and there is exactly one such spider graph. If $n = 2$, the spider graph consists of just the center and one vertex, and there is exactly one such spider graph. For $n > 2$, vertex 1 is either the end of a length-2 leg, or of a length-1 leg. By removing the leg containing the vertex 1 and decreasing all labels, we get some partially labeled 1-2-spider graph on either $n - 2$ or $n - 1$ vertices.

This is clearly a bijection between all our spider graphs on n vertices and all our spider graphs on ($n - 1$ and $n - 2$) vertices. Hence the numbers

$v(n)$ satisfy $v(1) = v(2) = 1$ and for all $n > 2$, $v(n) = v(n-1) + v(n-2)$. Therefore obviously $v(n) = F_n$. □

In the following text, let $\phi = \frac{1+\sqrt{5}}{2}$ be the golden ratio. It is well-known that $F_n = (\phi^n - (-\phi)^{-n})/\sqrt{5}$. And as $|\phi^{-1}| < 1$, F_n equals $\phi^n/\sqrt{5} + O(1)$. (Actually, it is $\phi^n/\sqrt{5}$ rounded to the nearest integer.) Note that $\lg \phi \approx 0.69424$.

Lemma 19. *There is a deterministic online algorithm solving* OnlineColoring(spider,any) *with advice complexity* $\lg \phi \cdot n + 3 \lg n + O(1)$.

Proof. The general idea of this proof is that if we only care about coloring isolated vertices, each possible spider graph is equivalent to a partially labeled 1-2-spider graph in which the isolated vertices are precisely the endpoints of all legs. (Multiple spider graphs will correspond to the same 1-2-spider graph, which decreases the amount of advice we need.)

The advice string for our algorithm will consist of three parts. The first part will be the number of vertices n. For this we use $2 \lg n$ bits of advice ($\lg n$ for the unary representation of n's length and another $\lg n$ bits for the actual value of n).

The second part will be the label of the center vertex, i.e., the turn in which we color the center vertex. This value is encoded in $\lg n$ bits (as the online algorithm already knows n). We will pick a coloring in which the center vertex has color 2.

According to Lemma 6, the only interesting vertices where the online algorithm needs advice are the vertices that are isolated at the moment when we are requested to color them. We will call them "special vertices" for short.

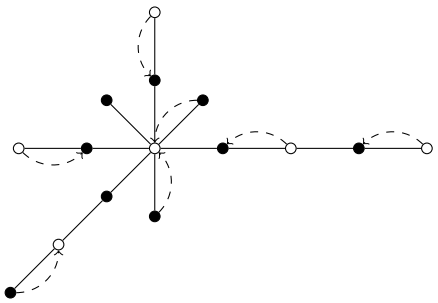

Figure 4: Deconstructing a spider graph. Black vertices have color 1, white have color 2. Each arrow points from a possible vertex v to the corresponding vertex u.

(The center vertex is never considered to be special, as we already know its color.) The third part of the advice string will be used to encode the correct coloring of all special vertices. However, we will not do it in the naive way that needs one bit per vertex.

Consider any special vertex v, and let u be the neighbor of v that is closer to the center. Note that u is never a special vertex. The vertex u may be the center vertex if and only if the correct color of v is 1. Hence if v_1 and v_2 are two special vertices with correct color 2, their corresponding neighbours u_1 and u_2 are always distinct. This is shown in Figure 4.

The oracle will take the instance I and transform it into a corresponding partially labeled 1-2-spider graph I'. To do this, we process all vertices of I in the order in which the online algorithm will see them. Whenever we encounter a special vertex that should receive the color 2, we will add a leg of length 2 to I', and whenever we encounter a special vertex that should receive the color 1, we will add a leg of length 1. The end vertices of legs in I' are labeled in the order in which they were added.

According to the above observation, the resulting instance I' will never

have more than n vertices. If it has less, we add additional legs of length 1 until the number of vertices is correct. We will take the instance I' and transmit it to the online algorithm using advice. (To transmit I' we use the usual trick: all possible instances can be numbered in lexicographic order.) The online algorithm can decode the instance I', find its correct coloring, and then use that coloring to color the special vertices in its actual instance.

By Lemma 18, there are F_n possible instances I', hence we need $\lg F_n$ bits to describe any of them to the online algorithm. This gives us a total of $3 \lg n + \lg F_n = \lg \phi \cdot n + 3 \lg n + O(1)$ bits of advice. □

Note that the statement of Lemma 19 can be slightly improved by using a better encoding to transmit the value of n, for instance by using the Elias Delta code [16].

9. Conclusion

In this paper we barely scratched the surface of the analysis of online graph coloring in terms of its advice complexity. We strongly believe that once there is a significant progress in this direction, useful patterns will emerge. For instance, we are convinced that it should be possible to find and describe some relation between the advice complexity of particular variants, the competive ratio we can obtain in the online setting, and the approximation ratio we can obtain in polynomial time in the offline setting. However, we are still far away from this goal, much more research in this new interesting area is necessary before we can aim for such results.

Another open line of research is a more fine-grained analysis. In this paper, we only focused on solving the particular online problems optimally.

However, as we already outlined in our classification, for all sufficiently complex graph classes it also makes sense to investigate the trade-off between the amount of advice available and the quality of the competitive ratio we are able to guarantee (or, with a randomized algorithm, obtain in expectation). This more detailed analysis may then also lead to new, interesting offline approximation algorithms.

Acknowledgement

The authors are thankful to Juraj Hromkovič for bringing this problem to their attention, to Maria Paola Bianchi for helpful discussions that lead to the paper [1] and especially to Hans-Joachim Böckenhauer for his kind help, insightful discussions throughout the work on this paper, and its very careful proofread.

References

[1] M. Forišek, L. Keller, M. Steinová, Advice Complexity of Online Coloring for Paths, in: Language and Automata Theory and Applications (LATA 2012), 2012, pp. 228–239.

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

[3] J. Hromkovič, Design and Analysis of Randomized Algorithms, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2005.

[4] S. Dobrev, R. Královič, D. Pardubská, How much information about the future is needed?, in: Geffert et al. [17], pp. 247–258.

[5] Y. Emek, P. Fraigniaud, A. Korman, A. Rosén, Online computation with advice, in: Albers et al. [18], pp. 427–438.

[6] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke, On the advice complexity of online problems, in: Dong et al. [19], pp. 331–340.

[7] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, On the advice complexity of the k-server problem, in: Aceto et al. [20], pp. 207–218.

[8] J. Hromkovič, R. Královič, R. Královič, Information complexity of online problems, in: MFCS, Vol. 6281 of Lecture Notes in Computer Science, Springer, 2010, pp. 24–36.

[9] D. Komm, R. Královič, Advice complexity and barely random algorithms, in: Černá et al. [21], pp. 332–343.

[10] M. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, L. Keller, Online coloring of bipartite graphs with and without advice, in: Proceedings of COCOON 2012, 2012, (to appear).

[11] M. M. Halldórsson, M. Szegedy, Lower bounds for on-line graph coloring, in: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms, SODA '92, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992, pp. 211–216.

[12] I. Cieślik, On-line graph coloring, Ph.D. thesis, Jagiellonian University Kraków (2006).

[13] N. G. de Bruijn, Asymptotic Methods in Analysis, Dover, 1981.

[14] E. R. Canfield, C. Pomerance, On the problem of uniqueness for the maximal Stirling number(s) of the second kind, Integers 2, paper A1.

[15] J. A. Gallian, A dynamic survey of graph labellings, Electron. J. Combin., Dynamic Surveys(6):95pp.

[16] P. Elias, Universal codeword sets and representations of the integers, IEEE Transactions on Information Theory 21 (1975) 194–203. doi:10.1109/TIT.1975.1055349.

[17] V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, M. Bieliková (Eds.), SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19–25, 2008, Proceedings, Vol. 4910 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2008.

[18] S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, W. Thomas (Eds.), Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5–12, 2009, Proceedings, Part I, Vol. 5555 of Lecture Notes in Computer Science, Springer-Verlag, 2009.

[19] Y. Dong, D.-Z. Du, O. H. Ibarra (Eds.), Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings, Vol. 5878 of Lecture Notes in Computer Science, Springer-Verlag, 2009.

[20] L. Aceto, M. Henzinger, J. Sgall (Eds.), Automata, Languages and Programming, 36th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4–8, 2011, Proceedings, Vol. 6755 of Lecture Notes in Computer Science, Springer-Verlag, 2011.

[21] I. Černá, T. Gyimóthy, J. Hromkovič, K. G. Jeffery, R. Královič, M. Vukolic, S. Wolf (Eds.), SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings, Vol. 6543 of Lecture Notes in Computer Science, Springer, 2011.