MASTER'S DEGREE THESIS

Master of Science in Advanced Mathematics and Mathematical Engineering

Title: Online Coloring Problem with a Randomized Adversary and Infinite Advice.

Author: Elisabet Burjons

Advisor: Xavier Muñoz

Department: Matemàtica Aplicada IV

Academic year: 2014-2015

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Tesi de màster

Online Coloring Problem with a Randomized Adversary and Infinite Advice

Elisabet Burjons

Advisor: Xavier Muñoz

Matemàtica Aplicada 4

Als meus amics, els algorismes

Abstract

Keywords: on-line problems, advice complexity, k-server problem, randomized adversary, graph coloring

MSC2000: 200468W27, 68W20, 05C15, 05C85

Online problems are those in which the instance is not given as a whole but by parts named requests. They arrise naturaly in computer science. Several examples are given such as ski rental problem and the server problem. The performance of the online algorithms is analized in terms of the ratio between the cost of the algorithm and the cost of the optimal offline. This ratio is called the competitive ratio. Several models of online algorithms are described. They are deterministic algorithms, randomized algorithms and algorithms with advice (an oracle that provides information of the input).

In the k-server problem, k servers are given to serve requests (points) in a metric space. In a graph, the distance between two vertices is 1 if the vertices are adjacent. Given the k-star graph upper and lower bounds on the number of advice bits required for r-competitiveness are given. Any deterministic algorithm with at most l advice bits, has a competitive ratio of at least $r \leq \frac{k+2^l-1}{2^l}$. There is an algorithm with l bits of advice that has a competitive ratio $r \leq \frac{3}{2}l + \frac{(k-1)}{2^l}$.

Given a graph G, coloring the graph is giving a coloring of the vertices such that no adjacent vertices recieve the same color. The goal is to minimize the number of colors used. In the online coloring problem, defined in chapter 2, each request contains a vertex and all the edges from the vertex to previously presented vertices. The vertex must be colored before the next request is provided. We survey results on the problem for the deterministic, randomized and advice models.

Finaly, a new model is described for the online problems. In this randomized adversary model, the adversary (that builds the instances) is allowed to use random bits. In this way the information that the oracle can provide to the algorithm is limited. In the case of the online coloring, as an upper bound, there exists an algorithm that uses $\frac{(s+2)k}{2}$ colors in expectation when the adversary uses s random bits to build the instances and the input graphs are at most k-colorable. The lower bound given is tight but not general, it has only been proved in the case of minimum capable algorithms (algorithms able to attain the optimal coloring in some instance). It states: There exists an adversary using s random bits to generate k-colorable problem instances for which the number of colors required on average by any minimum capable algorithm is $c_k(s) \geq \frac{k}{2}(s+2)$. We describe also an attempt to prove the general lower bound by induction and a proof is given for the lower bound in the case where $s = 1$.

Notation

Alg	online algorithm
Opt	optimal algorithm
I	problem instance
Alg(I)	cost of Alg given instance I
Opt(I)	optimal cost of problem for instance I
n	instance length
$G,\ G(V,E)$	graph
V	set of vertices
v	vertex
E	set of edges
$G(I)$	online graph construced from instance I
$T,\ T(I)$	tree (online)
$\chi(G),\ k$	chromatic number
s	number of random bits
$c_k(s)$	average number of colors required to color a k-colorable instance with s random bits
K_k	complete graph with k nodes
p_0, p_1	permutations
$K_{k,k}$	complete bipartite graph with 2 sets of k nodes
$K_{k,k} \setminus \{p\}$	complete bipartite except perfect matching generated by the permutation p

Contents

Chapter 1
Introduction to Online Problems

1. Online Problems and Online Algorithms

Computational problems solved by algorithms are normally modeled in such a way that for each possible input of the problem, an algorithm processes it in order to output the best possible answer. In doing this the algoritms are usually designed and classified according to their performance in time, number of operations, optimality of the output, resources, memory ...

Instead, *Online problems* are interactive in the following sense: The input of an online problem is given in a sequence of input portions that we call *request sequence*. After each input portion or *request*, the algorithm is forced to give a partial output. So, the main difference between the usual computational problems, *offline problems*, and the online problems is the fact that the input of an online problem is not known a priori and the partial output given by the online algorithm is final and can not be changed.(See [**1**])

As one can imagine, this type of problems (online) and the *online algorithms* solving them can not be analyzed and classified in the same way as the offline ones according to the time performance etc. One tool used to evaluate the performance of an online algorithm is the *competitive ratio*.

As stated previously, an online algorithm is required to give a partial output for each request. Serving these requests incurs a cost and the overall goal is to minimize the total cost. The competitive ratio compares the cost paid by the online algorithm given an input sequence (Alg(I)) with the cost paid by the optimal offline algorithm given the same input sequence (Opt(I)) as follows.

Definition [**7**] An online algorithm (Alg) is *c-competitive* if there is a constant α such that for every finite input sequence I,

$$\mathrm{Alg(I)} \leq \mathrm{c} \cdot \mathrm{Opt(I)} + \alpha$$

We say then that the *competitive ratio* of the algorithm Alg is c. Usually, the competitive ratio of an algorithm is given in terms of the length of the input or

other meaningful variables for the problem. We realize that the competitive ratio of an algorithm is 1 if the algorithm is optimal. Slight changes on the definitions of cost and competitive ratio should be done in the case of maximization problems.

Example 1: The ski rental problem

[**21**] One of the examples that helps us the most in understanding the concept of online requests and competitive ratio is the Sky rental problem. Which states as follows.

Assume that you go on a vacation and are taking ski lessons. At the begining of each day you decide wether you continue taking lessons or stop and return from your vacation. You take this decision according to how much you have enjoyed the lesson, the weather, or if you have any broken bones. Each day you can decide wether to rent the skis for 1€ or to buy them for 10€. Of course once the skis have been bought then no more decisions are required.

As one can imagine the cost is computed in terms of how much money is spent in order to fulfill the input. The optimal offline strategy in this problem is, clearly, buy the skis if the vacation is longer than 10 days and rent otherwise. An online algorithm consists in deciding which day the skis are bought (eg. Alg = Rent until day 5 then buy day 6).

In order to compute the competitive ratio of an algorithm for the ski rental problem let us consider the following. In the case of Alg the worst possible instance would be if the trip is canceled the next day after buying the skis. In the previous example, the cost paid by Alg, $\text{Alg(I)} = 15$ whereas the optimal would have cost $\text{Opt(I)} = 6$, as discussed before, so $c = 15/6 = 2,5$. Thus providing a measure of how good the algorithm is. However, it is not always this simple to find the worst case instance and the best algorithm. The next section describes an approach that is often useful when computing competitive ratios for online algorithms.

2. Playing with an Adversary

One useful way to view the problem of analyzing online algorithms is to consider the problem as a game between and online player and a malicious adversary [**7**]. The online player runs an online algorithm on an input created by the adversary. The adversary, knowing the algorithm used by the online player, constructs the worst possible input so as to maximize the competitive ratio. It is, the adversary not only tries to make the input costly for the online algorithm but at the same time it also makes it inexpensive for the optimal offline algorithm.

Example 2: The ski rental problem (continued)

The adversary for the ski rental problem will make the trip stop the day after buying the skis. This is because after that, the algorithm does not have any additional

costs thus possibly decreasing the competitive ratio. So for any algorithm buying skis after k days the competitive ratio will be:

$$c = \frac{k+10}{min\{k+1, 10\}}$$

So, the algorithm minimizing the competitive ratio will be the one for $k = 9$ where we can see that $c = \frac{19}{10} = 1.9$. This means that, when applying this algorithm, the cost of the skis will never be over twice the optimal cost no matter the circumstances.

It is not always the case where the competitive ratio of a problem can be easily computed. This is why in most online problems upper and lower bounds for the competitive ratio give us important information about the problem.

3. Upper and Lower Bounds

Upper and lower bounds for the competitive ratio of online problems are often given in terms of a meaningful variable to the problem such as the input length. This is not the case in the ski rental problem, where the competitive ratio does not depend on any variable. However, as we will see in chapter 2 in the problem of Online coloring the input length and the chromatic number of the graph presented by the input are often meaningul to the achievable competitive ratio.

Upper bounds in the competitive ratio determine the *competitive ratio achievable by a given algorithm*, i.e for any posible adversary, which is the competitive ratio of the algorithm. The better the algorithm considered, the lower the upper bound is.

Lower bounds determine the *minimum competitive ratio achievable by any algorithm*, i.e. an instance (adversary) must be given in which any algorithm can not perform better than the lower bound. Usualy finding this bounds is more complicated. This is because one has to find an instance in which every online algorithm performs "significantly" worse than the optimal offline algorithm.

The goal in the analysis of online problems is finding upper and lower bounds to the competitive ratio. If both bounds match, then we have computed the competitive ratio of the problem. In some problems upper and lower bounds are found that do not match but that have the same order of magnitude in terms of the input length or some other relevant variable.

An example that may surface the relevance of bounding the competitive ratio is the server problem. It will be explained in section 6.

4. Randomizing the strategy

If we allow the online player to use randomness, smaller competitive ratios are attainable in most cases. The introduction of randomness in the game is natural

and not surprising given the important role of randomization in algorithms and game theory (see [**7, 15**]).

It is important to notice that the adversary loses power with the randomization of the algorithm. In our case, the adversary is aware of the strategy of the online algorithm. However, it does not know the outcome of the random bits, we call this type of adversary an *oblivious adversary*[**7**]. One way to think of it is imaginining the adversary has access to the code of the algorithm. It can not know the particular decisions of the algorithm if those are made at random. This makes impossible to design strategies as the one for the ski rental problem mentioned above (i.e. if the buying day is randomized, the adversary can not know which day to finish the trip). There are other types of adversaries (online adaptative, offline adaptative,...) discussed also in [**7**].

The competitive ratio must be redefined as now the performance of the algorithm is not deterministic. So, the modified definition of the competitive ratio is stated in [**7**] as follows.

Definition Let Alg be a randomized online algorithm. Based on the knowledge of Alg (in particular, the probability distribution(s) Alg uses) the adversary must choose a finite request sequence (instance) I in advance. Alg is c-competitive against an oblivious adversary if for every I,

$$\mathbb{E}[\mathrm{Alg(I)}] \leq \mathrm{c} \cdot \mathrm{Opt(I)} + \alpha$$

Where $\mathbb{E}[\cdot]$ is the expectation with respect to the random choices of Alg and α is a constant. We say that c is the *expected competitive ratio against an oblivious adversary.*

In section 6 we will see a theorem upper bounding the competitive ratio for online randomized algorithms in the k-server problem. Although it is not the case in the theorem presented, there are results in which the number of random bits used is relevant to the bound.

5. Reciving advice: The Oracle

Another setting considered in [**10**], consists of making certain bits of information available to the algorithm. The algorithm (Alg) receives information about the input sequence. This information allows the algorithm to perform better and achieve lower competitive ratios.

This model is useful in order to investigate how much information Alg is lacking with respect to Opt. Surprisingly, there are some problems such as ski rental problem where only one bit of information allows Alg to be as good as Opt as we will see. Clearly, this does not hold in general and this framework allows us to classify online problems according to how much information about the future input is needed in order to solve them optimally or with a specific competitive ratio. There are cases in which additional bits do not help at all to improve online

algorithms for specific problems as well as situations when very few advice bits cause a jump in the achievable competitive ratio [**6**].

Example 3: The ski rental problem (continued)

[**9**] This problem is very simple from the advice point of view, as a single bit of information can tell the algorithm wether to buy the skis the first day or rent every day. The optimal offline always does one or the other. So, we conclude that the advice complexity of the ski rental problem is 1.

Studying online problems from this point of view of getting tradeoffs between the quality of the solution and the size of advice provided a new instrument for measuring the hardness of online problems. Other important conceptual contributions are the development of a powerful method for proving lower bounds on the achievable competitive ratios of randomized online algorithms and new insights on the possible power of information [**6**].

This model can be viewed as a cooperation in between the oracle O which has unlimited computational power and the online algorithm Alg. The oracle sees the whole input of Alg in advance and writes bitwise information needed by Alg onto an *advice tape* before Alg reads any input. Then, Alg can accesss the bits from the advice tape sequentially in the same way a randomized algorithm accesses bits from the random tape.

The *advice complexity* of Alg on an input I is the number of advice bits Alg reads while processing this input. This is usualy considered as a function of the input size. It is also maximized over all inputs of length at most n in order to obtain the advice complexity of the algorithm.

It is important to note that the advice complexity is the number of bits used by Alg. This should be distinguished from the number of pieces of information. For example the integer n is encoded in $\log n$ bits. It may be the case where more than one bit of advice is read for one algorithm step.

In this model the bounds are on the advice complexity or on the tradeoff between advice complexity and competitive ratio. As we will see in section 6, the bounds on the advice model are quite diferent from the bounds for deterministic and randomized algorithms.

It may not seem intuitive but in fact there is a relation betwen randomization and advice and reads as follows:

THEOREM 5.1. [**5**] *Consider an online minimization problem P, $I(n)$ the set of all possible inputs of length n. Suppose that there exists a randomized online algorithm R with a wost-case expected competitive ratio of at most E. Then, for any fixed $\varepsilon > 0$, it is possible to construc a deterministic online algorithm A with advice that uses at most*

$$\lceil \log n \rceil + 2\lceil \log \lceil \log n \rceil \rceil + \log \left(\frac{\log(|I(n)|)}{\log(1+\varepsilon)} \right)$$

advice bits and that achieves a competitive ratio of $(1+\varepsilon)E$.

It basically states that advice bits are as helpful as random bits when it comes to the competitive ratio.

6. The k-Server problem in a k-star graph

A general definition of the problem [**23**] is the following:

Given a metric space $(X; d)$ and k servers, localized in some point of the space. The online algorithm receives requests for servers over time. Each request is a point in the metric that needs a server, and the question is which server should we send to that point. As in the other online algorithm a new request arrives only after having served the previous request. The goal in the (k-)server problem is to minimize the total distance travelled by servers.

Solving this problem for a general metric space is rather complicated. In fact the problem is conjectured to be k competitive and the problem is known to be at least k competitive as shown in [**20**]. The matching upper bound is yet to be proven. The best upper bound known to the date is the one in [**18**].

THEOREM 6.1. *The competitive ratio of the k-server problem in a general metric is at most* $2k-1$.

The power of randomization can be noted in many problems. In particular, for the Server problem a bound was found in [**2**] that improves on the deterministic bound in [**18**] whenever the number of points on the metric space n is sub-exponential in the number of servers k.

THEOREM 6.2. *There is a randomized algorithm for the k-server problem that achieves a competitive ratio of* $O(\log^2 k \cdot \log^3 n)$ *on any metric space on n points.*

In this section we present a particular case of the problem. Let SG_k be the star graph with k leaves. This graph consists in $k+1$ vertices. One of them, namely a, has degree k and is adjacent to the rest of vertices in the graph, the leaves, namely $\ell_1, \ldots, \ell_k$. See figure 6.

The distance in between two adjacent vertices of the graph is defined as 1. We are given k servers to attend requests within the graph. This means that at each step of the algorithm there is only 1 unattended vertex. This fact simplyfies the problem and permits us to establish paralelisms with the paging problem [**6**], which is in fact the k-server problem in a complete graph.

We present a lower bound and an upper bound on the number of advice bits for r-competitiveness. And also an upper bound randomized algorithm.

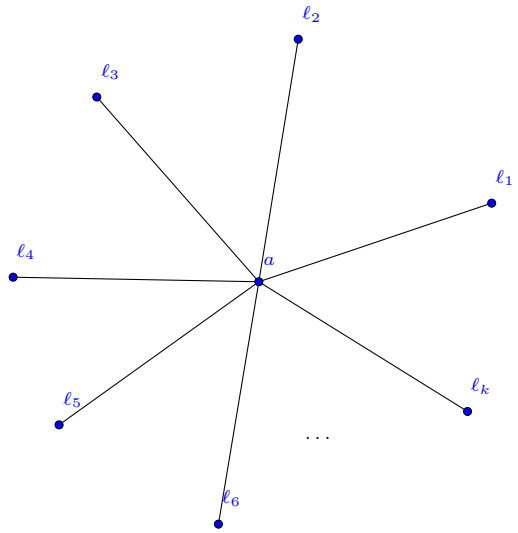

FIG. 1. Graph SG_k

6.1. Lower Bound on the advice complexity for r-competitiveness

In order to set a lower bound we will consider the set S_k of special finite request sequences of the type $a, i_1, a, i_2, a \ldots$, where i_j correspond to leaves. Note that S_k is infinite. Let S_k^t for $t = 1, 2, 3, \ldots$ denote the subset of S_k consisting of the sequences of the length $t \cdot 2k$. For every deterministic algorithm there is one request sequence from this set where the algorithm must make a movement at each step.

Now we build a tree T_k that visualizes the request sequences in S_k as follows. The sons of each vertex are all the possible requests that follow the sequence after that vertex. Starting with the root, it has one son corresponding to the request a, this son of the root has k sons, corresponding to the k possible leaves requested after the central vertex, each of those has only one son which corresponds to the request a after the leave request and so on. See figure 6.1.

We define the tree T_k^1 as the request tree of heigh $2k$ (it corresponds to the request sequences of length $2k$). And recusively we define the request tree T_k^n as the request tree of heigh $2kn$. This tree will have T_k^{n-1} as a root tree and then T_k^1 hanging from each of its leaves.

Given one concrete request sequence from S_k of length $2k$ (corresponding to a path of the request tree T_k^1) we know the optimal solution would take at most 2 steps in order to fulfill all the requests. This is because we have k servers and k requests

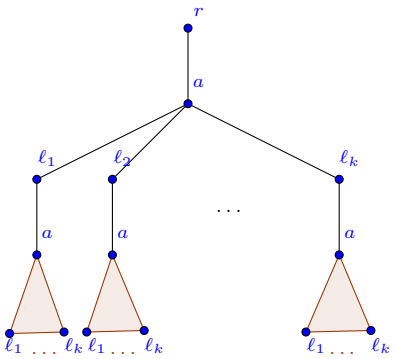

FIG. 2. Request tree T_k

correspond to the central vertex, and k requests correspond to leaves. So for each request sequence there is at least one leaf that is requested at most once. Thus, to serve the middle vertex, the optimal solution will take one server whose leaf will not be requested in the next $2k-2$ steps.

For each deterministic online algorithm A there exists a sequence in S_k such that A moves to each of the k a-requests exactly the server staying in the leaf that will be asked in the next request. In other words, the adversary always asks for that leaf that any server is attending, because it moved to a in the previous step. This is the worst possible case because one movement is needed at each step. We have then that under these circumstances any deterministic algorithm is at least k-competitive.

LEMMA 6.3. *For each sequence in S_k^t there is an optimal solution of cost at most $2t$ (at most $2t$ movements of servers are sufficient to fulfill al the requests of any sequence in S_k^t).*

Consider now the case in which one bit of advice is given. We take now the request tree T_k^2 of heigh $4k$, recall that this tree will have T_k^1 as a root tree and then again T_k^1 at each of its k^2 leaves. Considering that one advice bit is given, assign to each possible request sequence in T_k^2 (each leaf of T_k^2) a 0 or 1. Consider it as coloring the leaves of T_k^2 with two colors 0 and 1, where the coloring of the leaves corresponds to the assigned advice bits. Considering all the possible ways to do it, either one of the T_1^k's at the bottom has been assigned the same color to all its leaves or all trees T_k^1 have both colors.

If one of the bottom T_k^1's has been assigned the same color to all its leaves, we can consider it as the previous situation, where no advice on the bottom tree T_k^1 is given. In other words, the subtree given by leaves with the same color is proceed by a deterministic online algorithm given by the corresponding advice.Then, by the same reason as before each online algorithm will need $2k$ server movements to fulfill the bottom tree requests and at least 2 moves for the upper tree in the worst case.

If this is not the case, there is at least one leaf of each bottom tree T_k^1 has been advised color 0. Considering all the request sequences that have color 0 as advice, their prefixes completely color the leaves of the upper subtree T_k^1 of T_k^2. Then again each online algorithm A with the advice 0 must work for all prefixes of length $2k$ of the T_k^1 upper subtree of T_k^2 and so there exists $w \in T_k^2$ such that A moves a server exactly $2k$ times in order to fulfill the upper tree requests and at least 1 or 2 times for the bottom T_k^1.

However, in order to fulfill a sequence of $4k$ requests, the optimal solution makes 4 moves at most (we have $2k$ leaf requests so there is a leaf that is requested at most twice. We move to the center the server that will be requested the least in the leaf or the one that will be requested the last). It must also be said that the case in which the deterministic algorithm makes only one step for the bottom subtree, the optimal solution will make 3 moves at most so the competitive ratio is bounded by: $r \geq \min\{(k+1)/2, (2k+1)/3\} = (k+1)/2$. Hence, given one bit of advice, any deterministic algorithm will be at most $(k+1)/2$-competitive.

Now, we are able now to formulate the following induction hypothesis:

CLAIM 6.4. *Given the set S_k^n of all request sequences S_k of length $2nk$ (i.e. the tree T_k^n of heigh $2nk$), and n colors as advice to color the leaves, then any deterministic algorithm will make at least $2k + j_{n-1}$ steps to fulfill the request, where $j_{n-1} \in [1, 2(n-1)]$ is the optimal number of steps needed to fulfill a subset of request sequences of S_k^{n-1}.*

We already proved this claim for $n = 1$ and $n = 2$.

Now we want to prove it true for $n + 1$.

PROOF. Consider the request tree T_k^{n+1} of heigh $2(n+1)k$ and consider also that $n+1$ advice colors are given. In its root it has T_k^1, and a T_k^n starting at each leaf of the T_k^1 of T_k^{n+1}. Consider that each of the leaves is painted with one of the colors, i.e. assign an advice to each of the possible request sequences of T_k^{n+1}. There are two possible situations here:

(1) One of the lower subtrees T_k^n uses only n colors.
(2) Every one of the subtrees T_k^n uses all of the $n+1$ colors, meaning color 0 is advised in every subtree.

In the first case applying the induction hypothesis to the son tree T_k^n having at most n colors we get that each online algorithm with this advice distribution must perform at least $2k + j_{n-1}$ steps by induction hypothesis and one or two more steps to fulfill the upper subtree. So the algorithm makes $2k + j_n$ steps and the optimal makes at most $j_n + 2$ steps. So the bound for the competitive ratio is then:

$$r \geq \min_{j_n} \left\{ \frac{2k + j_n}{j_n + 2} \right\} = \frac{k+n}{n+1}$$

In the second case as color 0 is used in every son T_k^n of the root tree T_k^1 of T_k^{n+1}. So, for each path in the upper T_k^1 there exists a request sequence in S_k^{n+1} of length

$2k(n+1)$ with advice 0, whose prefix corresponds to be this path. Then using the same argument, each online algorithm A with the advice 0 must work for all the prefixes of length $2k$ of the upper subtree T_k^1, so there exists $w \in T_k^n$ such that A moves a server exactly $2k$ times in the first $2k$ steps.

In the lower subtree the deterministic algorithm will use at least the optimal number of steps which are $j_n \in [1, 2n]$ steps. Then the algorithm makes in total $2k + j_n$ steps, so the competitive ratio is:

$$r \geq \min_{j_n} \left\{ \frac{2k + j_n}{j_n + 2} \right\} = \frac{k+n}{n+1}$$

□

The same argument is valid for request sequences of length multiple of $2kn$.

THEOREM 6.5. *Considering the k-server problem in a k-star. Any deterministic algorithm with at most l advice bits, has a competitive ratio of at least:*

$$r \geq \frac{k + 2^l - 1}{2^l}$$

And now as an immediate corollary taking the reciprocal of the statement we are able to find the lower bound:

COROLLARY 6.6. *Given a deterministic algorithm such that:*

$$r < \frac{k + 2^{l-1} - 1}{2^{l-1}} \Rightarrow \textit{it uses at least } l \textit{ advice bits.}$$

6.2. Upper bound on the expected competitiveness of a Randomized Algorithm

We present a randomized algorithm and compute an upper bound for its expected competitive ratio.

In order to determine a bound we will consider again the set S_k of special finite request sequences of the type $a, i_1, a, i_2, a \ldots$, where i_j correspond to leaves.

This can be done because if it was the case that there is more than one leaf in between two center requests, any algorithm would need at most one move to fulfill all the leaf requests, so in the end when the steps made by the algorithm are compared to the optimal number of steps, these leaf requests would not provide any relevant information.

Consider the following marking algorithm based on [**11**] divided in phases:

- The phase P starts with all the servers unmarked, $k-1$ of them in the leaves and one in the center. The first request is for the unattended leaf f.
- A server is marked when its leaf is requested.
- Once a server is marked it is not moved from its leaf again during the phase.

- The phase P ends when $k-1$ servers are marked and the non marked server is in the center.The new phase begins when the unattended leaf is requested. At the end of each phase all servers are unmarked.

During the phase, the leaf f is requested at the begining, and afterwards there are exactly $k-2$ requests for unmarked leaves diferent than f.

When the center node a is requested and there is no server on the center, the algorithm picks one of the unmarked servers uniformly at random and takes that server to the center. When a new leaf is requested, if there is a server on the leaf, the server is marked. If there is no server on that leaf, the server in the center is moved to the leaf and the server is marked.

So when the n-th new leaf is requested, ($n-1$ marked leaves), the probability that the leaf has no server is $\frac{1}{k-n-1}$. Knowing that there will be $k-1$ requests for new leaves at the end of the phase, we can compute the expected number of movements in one phase:

$$\mathbb{E}(\text{steps in phase } P) = 2 + 2 \cdot \left(\frac{1}{k-1} + \frac{1}{k-2} + \ldots + \frac{1}{2} \right) = 2 + 2 \cdot (H_{k-1} - 1)$$

where the first 2 stands for the first two movements and H_k stands for the k-th harmonic number. Recall that the phase starts when the unoccupied leaf is requested and one server of the $k-1$ unmarked ones must be moved to the center. Then two movements are required every time an unocupied leaf is requested and another unmarked server must be moved to the center.

So the expected number of steps is $\mathbb{E}(P) = 2H_{k-1} \simeq 2 \log k$ when aproximating the $(k-1)$-th Harmonic number.

6.3. Upper Bound on the advice for r-competitiveness

The proof of the upper bound in this section is based on the upper bound for the paging algorithm in [**6**].

In order to build this upper bound we will consider again the special set of finite request sequences S_k defined in the previous subsection 6.2.

This can be done because if there were more than one leaf request between two center requests, any algorithm would need at most one move to fulfill all the leaf requests, so in the end when the steps made by the algorithm are compared to the optimal number of steps, these leaf requests would not provide any relevant information.

THEOREM 6.7. *Given the k-server problem in a k-star, and requests sequences of the type S_k. There is an algorithm with l bits of advice ($2^l < k$) with competitive ratio $r \leq \frac{3}{2}l + \frac{(k-1)}{2^l}$.*

PROOF. Defining $c = 2^l$, the goal is to build c deterministic algorithms $A_1, \ldots, A_c$ and argue that for each input there is one algorithm that does not make too many steps.

Each algorithm will be a marking algorithm. We recall from the previous subsection that marking algorithms do not make any moves for marked leaves. So let us consider that requests for marked leaves do not happen. Then, each phase consists of a request sequence in S_k containing requests for $k-1$ diferent leaves.

The requests are processed in $l+1$ rounds. For each round the algorithms are divided in groups and all the algorithms in one group perform the same during the round. Round 0 processes the requests $i_0, a, i_1, \ldots, i_{k-1-2^l}, a$, so it lasts $2(k-2^l)$ steps. Round u processes the requests $i_{k-2^{l-u+1}}, a, \ldots, i_{k-1-2^{l-u}}, a$ and lasts 2^{l-u} steps.

At each round the algorithms are partitioned in 2^{l-u} groups $G_0, \ldots, G_{2^{l-u}-1}$. At the begining of round 0 each algorithm is on a group alone and at the end of round $u > 0$ the groups G_i and $G_{i+2^{l-(u+1)}}$ are merged into a single group G_i.

Now, consider the requests i_t, a that are processed in round u. Processing i_t only consists in moving the server from the middle to i_t if the leaf is unattended and marking the leaf. After processing i_t there are $t+1$ marked leaves and $k-t-1$ unmarked ones. We keep an ordering of the unmarked leaves as $f_0, \ldots, f_{k-t-2}$ and assign the leaf f_i to the group G_i. It is possible to do that because $(k-t-2)-2^{l-u} \geq 0$, as $t \leq k-1-2^{l-u}$. There can be some algorithms $A_j \in G_i$ that have no server on the center after processing i_t. These algorithms must move a server to the center in order to process the request a. Each $A_j \in G_i$ that requires this move, will move to the center the server in the unmarked leaf f_i assigned to G_i.

After descriving the algorithms, we are going to find the upper bound of the number of server moves made by all the algorithms during one phase.

To begin with, each algorithm must make two movements at the begining of the phase, to fulfill the first couple of requests (recall that the phase starts with a request to the unattended leaf f), this adds to $2 \cdot 2^l$ movements in total (two for each algorithm).

We also count 2 movements every time that an algorithm gets assigned a new leaf (this is not tight as some of the algorithms in one group maybe have not needed any movement during the round). There are two instances in which an algorithm (or group of algorithms) will be assigned a new leaf. When the leaf assigned to the group is requested and when merging the groups G_i and $G_{i+2^{l-u}}$ (every algorithm in the second group gets assigned the leaf of the group G_i).

Each of the requests in round u can change the assigned leaf for one group of algorithms. There are 2^u algorithms in each group and 2^{l-u} requests in the round if $u > 0$ and $k-2^l-1$ requests in round 0 ($k-2^l-1$ assignments). And, at the begining of each of the l rounds half of the algorithm gets assinged a new leaf ($l \cdot 2^l$ assignments). Hence, the possible movements made by all the algorithms on this account are $2 \cdot (2^l \cdot l) + 2 \cdot (k-2^l-1)$.

Finally by adding up all the contributions, the movements made by all the algorithms are upper bounded by $3 \cdot 2^l \cdot l + 2(k-1)$. Since there are 2^l algorithms, there must be at least one of them that makes at most

$$3 \cdot l + \frac{2(k-1)}{2^l}$$

moves on the instance.

On the other hand, any optimal algorithm makes at least 2 movements. □

To sum up, the reciprocal of the theorem stated in this section provides an upper bound for the number of advice bits used by a deterministic algorithm with a lower bounded competitive ratio.

COROLLARY 6.8. *Given that any deterministic algorithm has a competitive ratio:*

$$r > \frac{3}{2}l + \frac{(k-1)}{2^l} \Rightarrow \textit{they use less than } l \textit{ advice bits}$$

To sum up, we have introduced the k-server problem. Some interesting bounds for the problem were summarized. We introduced the particular case of the k-server problem in a k-star. For this problem we have found an upper and a lower bound for the competitive ratio af an algorithm that uses l advice bits and the reciprocal of these statements provides us with upper and lower bounds for the number of advice bits used by deterministic algorithms with the competitive ratio lower and upper bounded respectively. A randomized algorithm was described that provided an upper bound for the expected competitiveness. This algorithm belongs to the class of marking algoritms. Introducing this algorithms helped us understand the marking algortihms described to prove the upper bound on the advice.

Chapter 2
Online Coloring: Previous Results

1. Introduction: Online Coloring

Given a graph $G(V, E)$ where V is the set of vertices of the graph and E is the set of edges, the graph coloring problem consists of coloring the vertices of G in such a way that no two adjacent vertices receive the same color. A coloring of a graph $G(V, E)$ is a function $g : V \to \mathbb{N}^+$ that satisfies this property. The goal is coloring V with the minimum number of colors. For any given graph G the minimum number of colors in which it can be colored is its *Chromatic number* $\chi(G)$.

The online coloring problem is one of the most studied online scenarios. Here, the vertices of the graph are revealed one after the other, together with all the edges conecting them to vertices that have already been presented. Each vertex must be colored before the next one is presented. As it happens in every online setting, once a vertex has been assinged the color it can not be changed. The goal is to use the minimum number of colors to color the graph.

In this setting the competitive ratio of an algorithm Alg that colors the problem instance I is the quotient between the number of colors used to color the graph generated by the request sequence $\chi_{\text{Alg}}(G(I))$ and the chromatic number of the said graph $\chi(G(I))$.

An introduction to the problem of online graph coloring can be found in [**17**].

2. General Results

Unlike the sky rental problem discussed in the previous chapter, the online coloring problem does not have a bounded competitive ratio, in fact the following theorem stands:

THEOREM 2.1. [**3**] *For every online algorithm Alg and every integer t, there exists a problem instance I constructing a tree $T(I)$ such that $\chi_{\text{Alg}}(T(I)) > t$. Moreover, $T(I)$ has only 2^t vertices.*

The proof is by induction on t. One constructs $T(I_i)$ for every $i = 1, \ldots, t-1$ such that each tree has 2^i vertices and the number of colors used is i in each of them. Then it is possible to choose a vertex x_i in each $T(I_i)$ such that each x_i has a distinct color. Then a final vertex v_{2^t} is requested adjacent to each x_i and it must receive the t-th color.

This graph would work for every algorithm but not all online algorithms require, as adversaries, graphs with this many vertices. For example, in the case of the greedy algorithm Gr (that colors the vertices with the first possible color available) it is possible to construct a bipartite graph instead with $2t$ vertices requiring t colors. We only have to consider the complete graph except the matching generated by the pairs of vertices $\{v_i, v_i'\}$. We denote this graph as $K_{t,t} \setminus \{id\}$ as the missing matching is generated by the identity permutation of vertices. And we present the vertices in the following order $v_1, v_1', v_2, v_2', \ldots, v_t, v_t'$.

An upper bound exists that matches the lower bound presented.

THEOREM 2.2. [**19**] *There exists an online algorithm Alg such that for every probem instance I corresponding to a 2-colorable graph $G(I)$ on n vertices, $\chi_{\text{Alg}}(G(I)) \leq 2\log(n)$.*

The algorithm is the following. Consider a new vertex v_i. There is a unique partition (I_1, I_2) of the connected component of v_i in the graph $G(I_i)$ where $v_i \in I_1$. Alg assigns to v_i the least color not assigned to any vertex in I_2.

Observe that if Alg assigns v_i color $k+2$, then Alg must have assigned $k+1$ to some vertex of I_2 and k to some vertex $v_p \in I_2$. Thus, Alg must have assigned k to some vertex $v_q \in I_1$. Since Alg assigned v_p and v_q the same color, they are in separate components of $G(I_r)$ where $r = \max\{p, q\}$. Thus, by induction, each of these components must have size $2^{k/2}$ and $i > 2(2^{k/2}) = 2^{(k+2)/2}$.

This two theorems bound the competitiveness of bipartite graphs. For general graphs with cromatic number k, an improvement of the proof of theorem 2.1 allowed to prove the lower bound of $\Omega(\log^{k-1} n)$. The theorem is as follows.

THEOREM 2.3. [**25**] *For every online algorithm Alg and all integers k and n, there exists an online graph $G(I)$ on n vertices such that $\chi(G(I)) \leq k$ and $\chi_{\text{Alg}}(G(I)) \geq \left(\frac{\log n}{4k}\right)^{k-1}$.*

This lower bound was improved in [**14**]. This is the best general lower bound known up to date.

THEOREM 2.4. [**14**] *For a general graph with n vertices, the competitive ratio of any deterministic online coloring algorithm is at least $2n/\log^2 n$.*

An alternative proof of the theorem can be found in [**22**]. It contains also a good review on the known bounds of the problem of online coloring in both the deterministic and randomized settings.

The best general upper bound known is sublinear. It does not match the upper bound and can be found in [**19**]. It states.

THEOREM 2.5. [**19**] *There exists an online algorithm Alg such that for every integer k and every online k-colorable graph $G(I)$ on n vertices, $\chi_{\mathrm{Alg}}(G(I)) \leq \left(\frac{kn}{\log^* n}\right)(1+o(1))$. Where $\log^* n$ is the smallest i for which the value of taking the logarithm of n recursively i times is at most 1.*

The original proof of this theorem used an algorithm constructed recursively. It is interesting to remark that the same kind of recursive construction was used in [**26**]. The algorithm presented in [**26**] is an offline approximate coloring algorithm that is polynomial. It is remarkable to observe that an offline technique became useful for the corresponding online problem.

As in the k-server problem, in the deterministic model of the online coloring problem the upper and lower bounds for the competitive ratio are still not tight.

3. Results for Randomized Algorithms

The upper and lower bounds known for the randomized model of the online coloring problem are not tight either.

A first upper bound algorithm, presented in [**25**], achieved an expected competitive of $O(n/\sqrt{\log n})$. The algorithm presented is very similar in style to the sublinear algorithm presented in [**19**], except that randomization is added. The algorithm recduces the problem and then applies the algorithm recursively until an easy base case is encoutered.

Another algorithm with a better expected competitive ratio was presented in [**13**].

THEOREM 3.1. *There eixsts a randomized online coloring algorithm such that has an expected competitive ratio of $O(n/\log n)$*

This is the best known upper bound for the randomized coloring algorithm. The best known lower bound does not match the upper bound and was presented in [**14**]. It states.

THEOREM 3.2. *The performance ratio of any randomized online coloring algorithm is at least $n/(16\log^2 n)$.*

It is interesting to note that the lower bound for randomized algorithms difers from the one for deterministic algorithms only in a constant. Both lower bounds were presented in the same paper and are the best ones known up to date for general graphs.

4. Results for Algorithms with Advice

No general results are known for coloring algorithms with advice. Bounds have been found for particular classes of graphs such as paths [**12**], spider graphs [**16**] maximal outerplanar graphs and chordal graphs [**24**]. There exist also results for bipartite graphs [**4**] and 3-colorable graphs [**24**].

Recall that in the case of bounds for algorithms with advice we are bounding the number of advice bits necessary for optimality or for r-competitiveness. This makes the results for problems with advice very diferent from results for randomized or deterministic algorithms where the competitive ratio is bounded. In this case we are interested in the tradeoff between the advice and the competitiveness. The results bound the number of advice bits necessary for a certain competitive ratio or the competitive ratio achievable with a certain number of bits.

For bipartite graphs the results are the following:

THEOREM 4.1. [**4**]

There exists an online algorithm for bipartite graphs which needs at most $n-2$ advice bits to be optimal on every instance of length n.

An almost tight lower bound for the upper bound given.

THEOREM 4.2. [**4**]

Any determinsitic online algorithm for bipartite graphs needs at least $n-3$ advice bits to be optimal on every instance of length n.

And an upper bound on the number of advice bits necessary for a constant competitive ratio.

THEOREM 4.3. [**4**]

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

And as a corollary they extend the upper bound to a non constant competitive ratio (depending on n).

COROLLARY 4.4. [**4**]

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

For 3-colorable graphs the results are:

THEOREM 4.5. [**24**]

For any n there exist a 3-colorable graph instance on n vertices for online coloring such that every deterministic online algorithm needs at least $\log_2 3 \cdot (n-4) - 1$ advice bits to be optimal.

And an upper bound.

THEOREM 4.6. [**24**]

For any 3-colorable graph instance on n vertices for online coloring there exists a deterministic online algorithm needs at most $1.5863 \cdot (n-3) + 46$ advice bits to be optimal.

There is also a similar theorem giving an upper bound on the number of colors needed to color a 3-colorable graph in at most 4 colors. The number of bits needed per request is also greater than 1.

No more general results exist for advice coloring up to our knowledge. However, we think that, a deterministic algorithm needs a lot of information about the graph instance in order to be optimal. This contrasts with the advice complexity of problems such as the ski rental discussed in the introduction.

5. Results for Non-General Graphs

There are several families of graphs such as trees, chordal graphs, etc. that have been analyzed. For some of this families tight upper and lower bounds exist for the competitiveness.

For example, cromatic bounded graph classes are graph classes where a constant upper bound exists for the competitiveness. There exists an algorithm that colours any graph of the class in a constant number of colors (not depending on the input size). In [**17**] we can find a summary of results for online cromatic bounded classes of graphs.

Also [**8**] contains bounds on the competitiveness of online coloring algorithms for some graph classes.

In the table 5 we summarize the results presented in this chapter.

Bounds		Deterministic	Randomized	Advice
General	Upper	$r \leq \left(\frac{n}{\log^* n}\right)(1+o(1))$	$r \leq O(n/\log n)$	-
	Lower	$r \geq \frac{2n}{\log^2 n}$	$r \geq n/(16\log^2 n)$	-
Bipartite	Upper	$r \leq 2\log n$		$n-2$ advice bits for optimality, $\frac{n}{\sqrt{2^{k+1}}}$ for k-colorability and $O(\sqrt(n))$ for $\lceil(\log n)\rceil$ colorability.
	Lower	$r \geq \frac{1}{2}\log n$		$n-3$ bits for optimality
3-Col	Upper			$1.5863 \cdot (n-3)+46$ bits for optimality.
	Lower	$r \geq \frac{1}{2}\log n$		$\log_2 3 \cdot (n-4)-1$ bits for optimality

FIG. 1. Bound summary

In the following chapter we present a new model for online algorithms. Then we find an upper bound and a restricted lower bound for the online coloring problem in this model. As we comented in the previous section, we think that the amount of advice bits needed for optimality in the online problem is very large. With this

model, we look at it from another point of view. The algorithms are given infinite advice but the adversaries are allowed to use s random bits. In some sense, we measure the amount of information the algorithm is missing with this bits and we bound the competitiveness of the algorithms with respect to s.

Chapter 3
Randomizing the Adversary of the Coloring Problem

1. Randomized Adversary

In this chapter we introduce a more general model (than the advice model) in which we allow the adversary to use random bits. In spite of the fact that randomization can be very helpful for online algorithms in the classical model of online computations, in the classical model random bits are not helpful for the adversary at all. Since the competitive ratio is defined in the worst-case manner, it is sufficient to produce deterministically one hard input for each online algorithm. In our model of advice complexity randomization can increase the power of the adversary. If the adversary constructs a set of problem instances from which the adversary can choose one of them randomly, and the advisor does not know these random bits in advance, then the advisor can only tell the set of possible problem instances or a probability distribution over the set of problem instances if not uniform, but not the whole problem instance that will be presented to the algorithm. In this scenario we measure the expected competitive ratio over the hardest set of problem instances. A new parameter in the game is the number of random bits of the adversary with respect to the achievable competitive ratio if the advisor is allowed to give an unbounded number of advice bits. A more advanced study may counter tradeoff or the number of advice bits, the number of random bits of the adversary and the competitive ratio.

There is also another view on the scenario. In the classical model without any advisor the adversary is not allowed to construct one of the hardest problem instances for the given online algorithm. The power of the adversary will be reduced by asking him, for a given positive integer s, to generate a set of 2^s problem instances. Then , the competitive ratio of the algorithm with respect to s is the maximum of the expected competitive ratios over all sets of 2^s instances, when the particular problem instances are chosen from the set constructed by a uniform probability distribution. This measure of the quality of online algorithms is reasonable because in the classical model an online algorithm is poor if there exists 1 hard problem instance for it. Here, investigating the achievable competitive ratio with respect to

s (or the size of the set of problem instances) can provide more insight on the hardness of concrete online problems. For instance, if the competitive ratio improves essentially with growing s, then the online problem can look easier that at the first glance in the classical model.

2. Online coloring with randomized adversary

In this chapter we apply this concept on the classical online coloring problem. Where vertices are coming one after the other with edges leading to the previous vertices only. As seen in the previous chapter, in the classical model the competitive ratio is unbounded, it is a sublinear function growing with the number of vertices n. This holds even for trees where this function is in $O(\log_2 n)$ [**16**]. In contrast to this we show that, with this new model, for all k-colorable graphs for $k \geq 2$, the expected competitive ratio for sets of 2^s problem instances is at most $(s+2)k/2$.

Note, that here the sets of problem instances are known to the online algorithm (infinite advice), that can choose the appropriate working strategy with respect to the given set of 2^s inputs. If this set is unknown we have here the same unbounded competitive ratio as for the classical online problem because for each online algorithm one can take the hardest problem instance and add a few dummy vertices in order to get 2^s problem instances.

For this upper bound $(s+2)k/2$ we will reach a tight lower bound when one restricts to online algorithms for which the probability of using minimum number of colors is not zero.

We also conjecture that the lower bound holds for every online algorithm and give a few arguments to support this conjecture. In particular we prove the lower bound for $s = 1$.

2.1. Upper Bound

The competitive ratio of the algorithms in this setting is linear with respect to the number of random bits available to the adversary. A practical way of computing competitive ratios in this setting is to look for the expected number of colors used by the algorithm in order to satisfy a problem instance. Knowing the chromatic number of the problem instance and the expected number of colors used computing the quotient between them we obtain easily the competitive ratio.

THEOREM 2.1. *Given an adversary that uses s random bits to generate inputs and all of its inputs are at most k-colorable, there exists an online algorithm with unbounded advice, that colors the input using $\frac{(s+2)k}{2}$ colors in expectation.*

PROOF. Given the set S of 2^s possible problem instances (defined by their request sequence of vertices) we can construct a rooted tree T_S where a node divides itself into two or more son nodes whenever a next request in the sequence enables to distinguish between two or more groups of different problem instances. The root represents the whole set of 2^s problem instances. For example, the first node never

gives any information about the problem instance because it is the same for all graphs. However, the second node may or may not be adjacent to the first node presented. So, looking at the first two nodes of all the request sequences, two groups of problem instances can be derived from that piece of information.

Once constructed, the tree of the set of problem instances (instance tree) will have at most 2^s leaves. Now, the algorithm colors the request sequence given as follows.

First of all choose one leaf ℓ in the instance tree according to the following criteria. For each node v in the path from the leaf ℓ to the root of the tree, the subtree T_v rooted in the node v has a greater or equal number of leaves than any of its sibling subtrees T_w. Observe that in a balanced tree any leaf chosen is good according to the criteria so, there may be more than one possible choice for the leaf. This leaf corresponds to a possible problem instance I_ℓ.

Start coloring the incoming vertices (requests) as if the graph given as input were the one corresponding to the selected leaf with k colors.

While the actual problem instance I coincides with the problem instance I_l we will be following the path from the root to ℓ. If a request R comes up that does not coincide with the path it will mean that the unknown actual problem instance does not correspond to I_ℓ. In this case the problem instance is one that corresponds to a leaf in the subtree T_R of problem instances that coincide with I up to the request R. A leaf $\hat{\ell}$ in T_R is selected according to the same criteria previously stated and k new colors are used to color the new incoming requests according to the coloring of the graph corresponding to $I_{\hat{\ell}}$.

Coloring in this way $(t+1)k$ colors are used at most to color the graph where t is the number of switches of leaves until the right leaf is reached. We have to prove that using this method not more than $s/2$ switches are performed on average. In fact, we will prove in the following lemma 2.2 that not more than $\frac{s2^s}{2}$ switches are required in order to fulfill all possible 2^s request sequences.

Knowing that in order to fulfill all 2^s possible request sequences not more than $s2^s/2$ switches are required, we can easily conclude that on average no more than $i/2$ switches are made, so in expectation no more than $\frac{sk}{2} + k = \frac{(s+2)k}{2}$ colors are used.

□

LEMMA 2.2. *Let S be any set of L problem instances, and let T_S be the corresponding tree with L leaves. Let $\#(T_S)$ be the number of switches in T_S. In T_S at most $\frac{L \cdot \log L}{2}$ switches are performed in order to fulfill all L possible request sequences with the previously described algorithm.*

PROOF. By induction on the number of leaves of the instance tree L. For $L = 2$ the only possible rooted tree is the root v and two leaves v_1, v_2 hanging from v. Either choice is possible for ℓ is equally valid so, choosing v_1 there will be no switches for the first problem instance and one switch for the second, making a total number of switches 1 as expected.

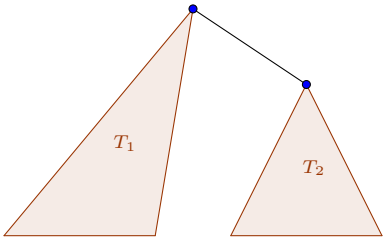

FIG. 1. Tree configuration

Suppose the lemma is valid for any tree with less than L leaves. For a tree with L leaves, the root will have two or more son vertices, pick the one which has the smallest subtree and name it T_2 and name the rest of the tree with the original root T_1 (figure 2.1). The selected leaf will always be in T_1 (as T_2 is the smallest subtree with a son root of T_S) the total number of switches is $\#(T_S) = \#(T_1) + \ell_2 + \#(T_2)$. This corresponds to the switches inside T_1, the switches inside T_2 and the ℓ_2 extra switches in the cases where problem instance is amongst the leaves of T_2. And using the induction hypothesis:

$$\begin{aligned}
\#(T_S) &= \#(T_1) + \ell_2 + \#(T_2) \\
&\leq \frac{\ell_1}{2}\log \ell_1 + \frac{\ell_2}{2}\log \ell_2 + \ell_2 \\
&= \frac{\ell_1}{2}(\log \ell_1 + \frac{\ell_2}{\ell_1}) + \frac{\ell_2}{2}(\log \ell_2 + 1) \\
&= \frac{\ell_1}{2}(\log 2^{\frac{\ell_2}{\ell_1}}\ell_1) + \frac{\ell_2}{2}(\log 2\ell_2)
\end{aligned}$$

Where we have rewritten the terms in order to have them as a sum of two logarithms. Now we use the fact that $\ell_2 \leq \ell_1$, and the inequality $2^x \leq 1 + x$ for $0 \leq x \leq 1$ for $x = \frac{\ell_2}{\ell_1} \leq 1$ in the first term and we conclude:

$$\begin{aligned}
\#(T) &\leq \frac{\ell_1}{2}(\log 2^{\frac{\ell_2}{\ell_1}}\ell_1) + \frac{\ell_2}{2}(\log 2\ell_2) \\
&\leq \frac{\ell_1}{2}(\log L) + \frac{\ell_2}{2}(\log L) \\
&= \frac{L}{2}\log L
\end{aligned}$$

Observe that if all of the son subtrees of the root are equal in number of leaves we can still choose the selected leaf amongst T_1 or in the other way around, select a subtree where the chosen leaf is not to be T_2. □

Dividing by k the expected number of colors we see that the competitive ratio of the online coloring problem in this setting is upper bounded by $\frac{s+2}{2}$.

Note that the upper bound on the competitive ratio does not depend of the number of vertices n the instance graph has. It is a remarkable diference that arises in this model in contrast to the deterministic and randomized models. This is because, as infinite advice is given to the algorithm, it does not matter how many vertices the adversary presents. This makes itself more evident in the lower bound. In the lower bound one can see that the adversary gains power from the fact that it can construct 2^s instances and present one or another acording to the result of the random bits.

The reasoning that has allowed us to prove this upper bound could also be used in other problems where the ratio is given by 'mistakes', no matter in which moment. A problem of this kind may be paging [**6**].

2.2. Restricted Lower Bound for Bipartite Graphs

The lower bound that we are going to present in this section is not general. First of all we present an adversary, i.e. a construction of 2^s instances. Then we lower bound the expected competitive ratio of the online algorithms against this adversary. The restriction does not come from the adversary construction but from the posterior analysis of the lower bound. In fact we bound the expected competitive ratio only of some algorithms. We call this algorithms minimum capable algorithms. These are capable of coloring the instance graph in the minimum number of colors possible even if it is only for some specific result of the random bits. It seems reasonable to us that the same adversary and lower bound should stand for all algorithms. This is because the other algorithms would use extra colors in unnecessary situations. However, the case analysis this type of algorithms require is greater and we have not been able to prove the lower bound without the restriction.

In order to find lower bounds for the expected number of colors required, let us consider the graph G_s defined as follows. Let the vertex set be $V(G_s) = \{x_0x_1\cdots x_m : x_j \in \mathbb{Z}_2, 0 \le j \le m \le s\}$ and in which there is an edge $(x_0x_1\cdots x_{m_1}, y_0y_1\cdots y_{m_2})$ whenever $m_1 \le m_2$, $x_{m_1} = \overline{y_{m_2}}$ and either $m_1 < m_2$ or $x_0x_1\cdots x_{m_1-1} = y_0y_1\cdots y_{m_1-1}$. A vertex of the form $x_0\cdots x_m$ is said to be of level $s-m$ or a L_{s-m}-vertex. Fig 2 shows graph G_2.

It is clear that G_s is bipartite, since vertices ending in 0 form an independent set and so do vertices ending in 1. Let us call those bipartite sets G_s^0 and G_s^1 respectively. It is also easy to check that graph G_s can be defined recursively from to copies of G_{s-1}, namely $G0_{s-1}$ and $G1_{s-1}$ by adding two new vertices 0 and 1 as shown in Figure 2.2. Vertex 1 is adjacent to all vertices in $G0_{s-1}^1$ and $G1_{s-1}^1$ and also adjacent to vertex 0. Similarly with vertex 0. Finally, in order to preserve the same label definition, just add prefix 1 to vertices in $G1_{s-1}$ and 0 to vertices in $G0_{s-1}$.

Given a sequence of s bits, let us define $Z_{b_1,\cdots,b_s}$ to be a function transforming the labels of the vertices in $V(G_s)$ as follows: $Z_{b_1,\cdots,b_s}(x_0x_1\cdots x_m) = z_0\cdots z_m$ with $z_i \equiv x_i + \sum_{j=0}^{i-1} x_j b_{s-j} \mod 2$. The outcome of this function is the swapping of the labels of some given vertices depending on the value of the bits $b_1,\cdots,b_s$. Notice that bit b_i affects to vertices in levels smaller than i. This swapping of labels is shown in Figure 2.

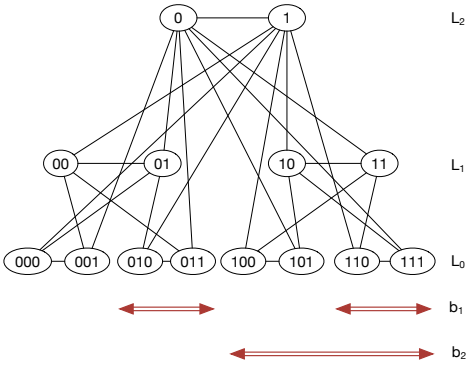

FIG. 2. Graph G_2.

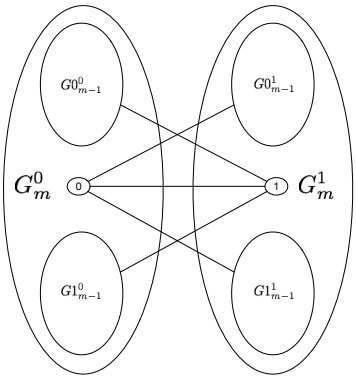

FIG. 3. Recursive definition of G_m

In order to determine a lower bound for the expected number of required colors let us consider an adversary that will produce 2^s different instances. All those instances will output the same graph G_s but showing its vertices in different order depending on the values of s bits $b_1, \cdots, b_s$. In all instances vertices will be shown by levels, starting from level 0, and the order in which vertices appear in each level is the lexicographic order of the images of their labels through function $Z_{b_1,\cdots,b_s}$.

It is easy to check that due to the symmetry of the graph, the algorithm cannot know the value of bit b_i before vertices in level i start to appear.

Notice that in order to color the graph with 2 colors the algorithm must guess correctly all bits $b_1, \cdots, b_s$ in all nodes where each bit is involved. Therefore an algorithm will be able to color the graph with minimum number of colors in at least one of the instances, if it follows the following two rules:

- Never use a new color if it is not strictly necessary at that point.
- Use the same strategy in all nodes.

An algorithm that is able to color the graph with minimum number of colors in at least one of the instance will be called a *minimum capable algorithm*. We will restrict ourselves to minimum capable algorithms, since those restrictions simplify to a large extent the analysis of the algorithm.

THEOREM 2.3. *There exists an adversary using s random bits to generate 2-colorable problem instances for which any minimum capable algorithm requires at least $c(s) \geq (s+2)$ colors on average.*

PROOF. We will use the adversary explained above to proof this result by induction.

For $s = 0$ there is only one instance. Recall that $G_0 = K_2$ and any algorithm will require exactly two colors. Therefore the result holds for $s = 0$.

In order to compute the number of colors required on average to color graph G_s let us consider the recursive definition of G_s depicted in Figure 2.2.

Let A_0 and A_1 be the color sets used in subgraphs $G0^0_{s-1}$ and $G0^1_{s-1}$ respectively, and let B_0 and B_1 be the color sets used in subgraphs $G1^0_{s-1}$ and $G1^1_{s-1}$. Because of the restrictions of minimum capable algorithms, the same strategy was used to color both subgraphs $G0_{s-1}$ and $G1_{s-1}$ and therefore not only the same colors are used in both subgraphs (i.e. $A_0 \cup A_1 = B_0 \cup B_1$) but also the same distribution of colors among their bipartite sets.

On the other hand since bit b_s was unknown before starting to appear vertices of level s, two possible situations arise:

- Bit b_s is guessed correctly: In this case $A_0 = B_0$ and $A_1 = B_1$ and vertices 0 and 1 (level s) may be colored with the same colors used in vertices in level $s-1$. In such a case, no new colors are needed.
- Bit b_s is guessed wrong: In this case $A_0 = B_1$ and $A_1 = B_0$ and vertices 0 and 1 need to use two new colors.

Finally, since probability of guessing bit b_s correctly is $\frac{1}{2}$, supposing that $c(s-1) \geq (s-1)+2$, we can conclude that $c(s) \geq \frac{1}{2}(c(s-1)+2) + \frac{1}{2}c(s-1) \geq (s+2)$ as expected. □

2.3. Restricted Lower Bound for k-colorable graphs, k even

The previous result is for bipartite graphs. In this section we are able to generalize the previous aversary and extend the previous result to prove a lower bound for k-colorable graphs. However, the bound is only tight in the case were k even. In the following section, a tight bound will be proved but a diferent adversary will be required.

First of all, theorem 2.3 can be extended for k-colorable graphs, as it is shown in the next Theorem:

THEOREM 2.4. *There exists an adversary using s random bits to generate k-colorable problem instances for which the number of colors required on average by any minimum capable algorithm is given by the expression:*

$$c_k(s) \geq \begin{cases} \frac{k}{2}(s+2) & \text{if } k \text{ even} \\ \lfloor \frac{k}{2} \rfloor (s+2) + 1 & \text{if } k \text{ odd} \end{cases}$$

.

PROOF. If k is even ($k = 2m$), m identical copies of the bipartite problem instance I^s (Theorem 2.3), namely I_i^s for $i = 1 \dots m$ are presented as follows: Each vertex in I_i^s is adjacent to all the vertices in I_j^s for all $i \neq j$ (in this way a color used in I_i^s can not be used in any other I_j^s for all $i \neq j$). Moreover the problem instance presents first all vertices of level t in all I_i^s before presenting any vertex of level $t+1$. In this way for each I_i^s 2 colors are necessary and sufficient so the chromatic number of the final graph is k. By Theorem 2.3 $(s+2)$ colors required on average for each I_i^s. The result is straightforward taking into account that colors used in different sets I_i^s must be different.

For $k = 2m+1$ proceed as in the case $k = 2m$ and let the adversary present, as a last round, a vertex adjacent to all other vertices presented. Clearly, the graph will be $k+1$ colorable and one additional color will be required in all cases. □

We see that the lower bound of the competitive ratio of the online coloring problem in this setting is $(s+2)/2$ for even k and $(s+2)/2 + \frac{1}{k}$ for odd k. This matches the upper bound in the even case as we expected.

Observe that in the odd case the lower bound does not match the upper bound. In the next section we are able to provide a proof through another set of instances that construct a diferent graph. As mentioned before, with this proof the lower bound is tight even for odd k. This proof has the same restricitions as the one presented in the previous sections.

2.4. Restricted Lower Bound for k-colorable graphs, $k \geq 3$

In this section we present a lower bound that matches the upper bound also in the odd cases but that does not work in the bipartite case. In this case we do not extend the adversary for the bipartite case, so we have to descrive another adversary that allows us to prove the lower bound. It is remarkable that despite the adversary being diferent, the analysis is also complicated for non-minimum capable algorithms. This shows us that the dificulty in the analysis lies in the nature of the algorithms and not in the chosen adversary.

Recall that a coloring of a graph $G(V, E)$ is a function $f : V \to \mathbb{N}^+$. We define $K_{k,k} \backslash \{p_1, \dots p_\ell\}$ as the bipartite graphs with 2 sets V_1, V_2 of k vertices v_i^1, v_i^2 for $i = 1 \dots k$ and all the edges between V_1 and V_2 except for the matchings corresponding to the permutations $p_1, \dots, p_\ell$ from V_1 to V_2, i.e. the edge $(v_i^1, v_{p_1(i)}^2)$ is not in the graph. We must remark that this notation has in some sense a directionality for

the edges, so, when taking this graph we are going to indicate from which vertex set it comes.

THEOREM 2.5. *There exists an adversary using s random bits to generate k-colorable problem instances with $k \geq 3$ for which the number of colors required on average by any minimum capable algorithm is given by the expression:*

$$c_k(s) \geq \frac{k}{2}(s+2)$$

.

PROOF. As in the case of the previous lower bound first of all we are going to describe the problem instances (the adversary) and compute the average number of colors required to color it by any minimum capable algorithm in order to find the bound.

First of all, for every possible sequence of random bits, a complete graph with k vertices, K_k^0, is presented (as no random bit is involved in its presentation). We name its vertices v_i for $i = 1, \ldots, k$. We will assume without loss of generality that any algorithm will color its vertices as $f(v_i) = i$.

Let p_0 and p_1 be permutations on k elements where $p_0 : [k] \to [k]$ is the identity and $p_1 : [k] \to [k]$ is unicyclic.

We present two more k-complete graphs. The second k-complete, $K_k^{0'}$ with vertices v_i', is connected with K_k^0 by $K_{k,k} \setminus \{p_0, p_1\}$ from K_k^0 to $K_k^{0'}$. Observe that $K_k^{0'}$ is colorable with the first k colors in two ways $f(v'(i)) = p_0(f(v_i)) = i$ or $f'(v'(i)) = p_1(f'(v_i))$. Where f and f' are the two possible coloring functions and $f(v_i) = f'(v_i) = i$.

The third k-complete, $K_k^{0''}$ and vertices v_i'', is connected to K_k^0 by $K_{k,k} \setminus \{p_0, p_1\}$ from K_k^0 and to $K_k^{0'}$ by the matching p_0.(see figure 2.4)

Observe that the graph is k-colorable since the coloring f can be extended to $f(v_i'') = p_1(f(v_i)) = p_1(i)$ and the coloring f' can be extended to $f'(v_i'') = p_0(f'(v_i)) = i$.

Observe also that in order to color the graph with k colors only this two colorings are possible since any other permutation of colors for $K_k^{0'}$ and $K_k^{0''}$ are forbiden by the edges $K_{k,k} \setminus \{p_0, p_1\}$ from K_k^0 to the other two complete graphs.

Now the first random bit comes into play. Another k-complete graph is presented. We name this new graph K_k^1 and vertices v_i^1. The adjacencies are:

- $K_{k,k} \setminus \{p_0, p_1\}$ from K_k^0.
- If the first random bit is 0:
  - the matching p_1^{-1} from $K_k^{0'}$.
  - the matching p_1 from $K_k^{0''}$.
- If the first random bit is 1:
  - the matching p_1 from $K_k^{0'}$.
  - the matching p_1^{-1} from $K_k^{0''}$.

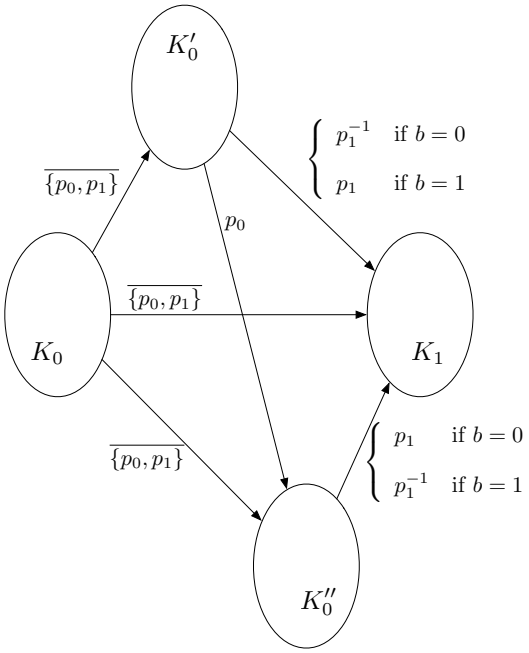

FIG. 4. Graph G_1.

Let f_0 and f'_0 be the colorings of K_k^1 for the event where the first random bit is 0. And f_1 and f'_1 be the colorings if the first random bit is 1. Let f_* coincide with f and f'_* coincide with f' in K_k^0, $K_k^{0'}$ and $K_k^{0''}$.

We can observe that the edges in $K_{k,k} \setminus \{p_0, p_1\}$ connecting K_k^0 to v_i^1 forbid every color between 1 and k except for $p_0(i)$ and $p_1(i)$.

In the event that the first random bit is 0, the matching p_1^{-1} from $K_k^{0'}$ for f_0 forbids the color $p_1^{-1}(f(v'_i)) = p_1^{-1}(i)$ that was already forbiden and for f'_0 it forbids the color $p_1^{-1}(f'(v'_i)) = p_1(i)$. And the matching p_1 from $K_k^{0''}$ for f_0 only forbids $p_1(f(v''_i)) = p_1^2(i)$ that, again was already forbiden and for f'_0 forbids $p_1(f'(v''_i)) = p_1^{-1}(p_1(i)) = p_0(i)$. So $f_0(v_i^1)$ can be either $p_0(i)$ or $p_1(i)$ but k new colors are necessary for f'_0.

In the event that the first random bit is 1, the matching p_1 from $K_k^{0'}$ for f_1 forbids the color $p_1(f(v'_i)) = p_1(i)$ and for f'_1 forbids the color $p_1(f'(v'_i)) = p_1^2(i)$ that was already forbiden. And the matching p_1^{-1} from $K_k^{0''}$ for f_1 forbids $p_1^{-1}(f(v''_i)) = p_1^{-1}(p_1(i)) = p_0(i)$ and for f'_1 forbids $p_1^{-1}(f'(v''_i)) = p_1^{-1}(i)$ that, again was already forbiden .So, k new colors are necessary for f_1 but $f'_1(v_i^1)$ can be either $p_0(i)$ or $p_1(i)$.

We see that there are two cases in which two possible colorings are admited with colors from 1 to k. In order to restrict this we will add the matching p_1 from K_k^0

to K_k^1. This forces $f_0(v_i^1 = p_0(i)$ and $f_1'(v_i^1) = p_0(i)$. It will be useful for the future analysis of the lower bound. Notice that this does not affect the other colorings.

We continue to construct the problem instances inductively in the following faishon. For the random bit $\ell \leq s$ a K_k^ℓ is presented, it has the edges $K_{k,k} \setminus \{p_0\}$ from K_k^n for $n = 0, \ldots, \ell - 1$ and the matchings p_1 and p_1^{-1} from $K_k^{n-1'}$ and $K_k^{n-1''}$ acording to the result of the $(n-1)$-th random bit.

In this way, if in the step n no previous colors could be used to color K_k^n, then these colors cannot be used either to color K_k^ℓ. This means that for any error in guessing the 'correct' coloring for $K_k^{n'}$ and $K_k^{n''}$ the adversary ensures that in the following steps those k colors can not be used again.

Once K_k^ℓ is presented, $K_k^{\ell'}$ and $K_k^{\ell''}$ are also presented (if $\ell < s$). Both of them have the edges of $K_{k,k} \setminus \{p_0, p_1\}$ from K_k^n for $n = 0, \ldots, \ell - 1$ and p_0 between them.

Note that this describes the 2^s instances as for each random bit 2 distinct instances are presented.

Finally, knowing how the adversary presents the 2^s possible instances and that for each "mistake" any minimum capable algorithm will need k extra colors, we can count the average number of colors used in coloring this set of instances. If ℓ bits are guessed wrong, at least $(\ell+1)k$ colors will be used to color the instance. Thus:

$$c_k(s) \geq \frac{\sum_{\ell=0}^{s} \binom{s}{\ell}(\ell+1)k}{2^s} = \frac{k}{2}(s+2)$$

□

Observe that this proof does not work for bipartite graphs as there is only one possibility for $p_1 \neq id$ and in that case $p_1^{-1} = p_1$.

Notice also that the restriction of minimum capable algorithms is also needed in the proof since we rely on the fact that for each mistake k colors need to be added. If extra colors were used in some step, it would be possible for only partial mistakes to occur and we can not count how many new colors would be needed in expectation for the next bit.

2.5. Further observations on the lower bound

As previously mentioned,we are convinced that the result is still true for general algorithms and not only for minimum capable algorithms, but the analysis is much harder:

CONJECTURE 2.6. *There exists an adversary using s random bits to generate 2-colorable problem instances for which any algorithm requires at least $c(s) \geq (s+2)$ colors on average.*

We have tried to prove the conjecture 2.6 by proving the following conjecture by induction:

Let $B = b_1, \ldots, b_s$ be a sequence of s random bits, and B^c be the complementary sequence.

CONJECTURE 2.7. *Given the adversary presented in section 2.2, any algorithm using ℓ colors to color the problem instance $I(B)$ will use at least $2(s+2)-\ell$ colors to color $I(B^c)$.*

If conjecture 2.7 could be proved, then conjecture 2.6 would be proved. As, according to this conjecture, for each pair of bit sequences B and B^c any algorithm uses at least $2(s+2)$ colors making a total average of $c_2(s) = 2^{s-1} \cdot (2(s+2))/2^s = s+2$ colors.

We did not manage to complete the proof by induction. However, it is interesting to state it because it reduces significatively the amount of cases to look at, which makes us think it is not a bad approach.

We consider induction on the number of random bits s. The result holds for $s = 0$. Suppose that the conjecture holds for every number of random bits $< s$.

Following notation from section 2.2. Let $G_s(B)$ be the graph constructed from the instance $I(B)$ and let $A0_0(B)$ be the set of colors used to color $G0^0_{s-1}(B)$. Analogously let $A0_1(B)$, $A1_0(B)$ and $A1_1(B)$ be the set of colors used to color $G0^1_{s-1}(B)$, $G1^0_{s-1}(B)$ and $G1^1_{s-1}(B)$ respectively.

We name $|A0_0(B) \cup A_1 0(B)| = a$ and $|A1_0(B) \cup A1_1(B)| = \hat{a}$. The number of colors used to color $G_s(B)$ is $\alpha = |A0_0(B) \cup A0_1(B) \cup A1_0(B) \cup A1_1(B)|$ and the number of colors used to color $G_s(B^c)$ is $\alpha^c = |A0_0(B^c) \cup A0_1(B^c) \cup A1_0(B^c) \cup A1_1(B^c)|$. By induction hypothesis we know that $|A0_0(B^c) \cup A0_1(B^c)| \geq 2((s-1)+2) - a$ and $|A1_0(B^c) \cup A1_1(B^c)| \geq 2((s-1)+2) - \hat{a}$.

Let $\alpha^{0,1}$ be the number of extra colors used for the vertices 0 and 1 (the last vertices presented) in both B and B^c. $\alpha^{0,1}$ takes values between 0 and 4.

Now we have to prove that $\alpha + \alpha^c + \alpha^{0,1} \geq 2(s+2)$.

First of all $\alpha \geq \max\{a, \hat{a}\}$. Moreover, $\alpha^c \geq \max\{2((s-1)+2)-a, 2((s-1)+2)-\hat{a}\}$. Without loss of generality we can assume that $a \geq \hat{a}$ and write $\alpha \geq a$ and $\alpha^c \geq 2((s-1)+2) - \hat{a}$ so,

$$\alpha + \alpha^c \geq a + 2((s-1)+2) - \hat{a} = 2(s+2) + (a - \hat{a}) - 2$$

If $a \geq \hat{a} + 2$ then we are done. Take into account that if the set $A0_0(B) \cup A0_1(B)$ and $A1_0(B) \cup A1_1(B)$ or $A0_0(B^c) \cup A0_1(B^c)$ and $A1_0(B^c) \cup A1_1(B^c)$ difer in 2 or more colors then we are also done. The following cases are left:

- $a - \hat{a} = 1$ and $A0_0(B) \cup A0_1(B) = A1_0(B) \cup A1_1(B) + 1$ color, $A0_0(B^c) \cup A0_1(B^c) + 1$ color $= A1_0(B^c) \cup A1_1(B^c)$. We expect that in this case a new color is needed to color the vertex 0 or 1 in $G_s(B)$ or $G_s(B^c)$ but we have not been able to prove it.
- $a - \hat{a} = 0$ and $A0_0(B) \cup A0_1(B) = A1_0(B) \cup A1_1(B)$, $A0_0(B^c) \cup A0_1(B^c) = A1_0(B^c) \cup A1_1(B^c)$. In this case we expect to see that two new colors are

needed to color the vertices 0 or 1 in $G_s(B)$ or $G_s(B^c)$. We have not been able to prove this either.

- $a - \hat{a} = 0$ and the sets $A0_0(B) \cup A0_1(B)$ and $A1_0(B) \cup A1_1(B)$, and the sets $A0_0(B^c) \cup A0_1(B^c)$ and $A1_0(B^c) \cup A1_1(B^c)$ difer in at most one color. In this case we expect to see that one new colors is needed to color the vertices 0 or 1 in $G_s(B)$ or $G_s(B^c)$. We have not been able to prove this either.

The main dificulty in aproaching these three remaining cases is the fact that we do not have any information in how the algorithm can use the colors in the sets. In fact we are trying to see that these colors are needed by any algorithm.

We are going to prove the lower bound in the case $s = 1$ in order to see why do we think that the lower bound holds. In the case $s = 1$ the graph G_1 is quite simple, and corresponds to figure 2.5.

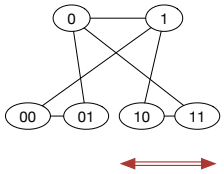

FIG. 5. Graph G_1.

According to the lower bound in average 3 colors are used in order to color this graph. Any algorithm using 3 or more colors to color the first 4 vertices has already reached the average. Any other coloring choice is minimum capable so on average performs as the lower bound states. This is because the choice of colors for 10 and 11 can be right with respect to the choice for 00 and 01, so no new colors are used for the vertices 0 and 1. Or it can be wrong and 2 new colors are needed for the vertices 0 and 1 thus ataining the average of $4 + 2/2 = 3$ colors.

In a similar way the case where $s = 2$ can be proven but a lot more cases need to be considered.

So, despite the fact that we have not been able to prove the lower bound for any algorithm, we have reason to believe that it is true, and we leave it as a conjecture.

Chapter 4
Conclusions

We defined the concept of online problems and a way to analize their performance with the competitive ratio. We saw diferent models for online algorithms and how they were diferent in terms of the bounds for the competitive ratio. The deterministic, randomized and advice models. The sky rental problem and the server problem were presented as examples together with some known bounds.

Then we considered the k-server problem in a k star and presented an upper and lower bound for the advice bits used in terms of the competitive ratio and an upper bound for the expected competitive ratio for the randomized model. The k-server problem in a k-star is very reminiscent of the paging problem. The methods in [**6**] could be applied for the proofs. The results though, gave us an insight in the online problems and the kind of bounds one gets with different settings such as deterministic algorithms, randomized algorithms and algorithms with advice. Some open questions that this case of the k-server problem presents is: What is the competitive ratio when there number of servers is less than then number of leaves? What could be said for the bounds if the graph is not a k star but a tree with k leaves?

In the second chapter we defined the online coloring problem and we reviewed the existing bounds for the models of deterministic online coloring, randomized online coloring and online coloring with advice. We saw that the existing bounds for the deterministic online coloring problem are not tight. Nor are the bounds for the randomized version of the problem. No general bounds are known for the advice model of the problem. Only bounds for the bipartite and 3-colorable graphs are known. This leads us to think that maybe another model will be succesful to provide bounds for the coloring problem in a setting where the algorithm does have information about the adversary.

In chapter 3, we presented a new model for on-line algorithm in which the on-line algorithm may use infinite advice bits but the adversary may use s random bits. This model gives us an insight in what happens when some information is missing. This is interesting to analyze in problems where few advice bits are not useful to bound the competitive ratio. In the case of online coloring for example, a lot of information is needed in order to achieve good bounds for the competitive ratio.

We studied the competitive ratio for the online coloring problem with this new model and it happens to be upper bounded by $1 + s/2$. We also found a couple of lower bounds which are valid with certain (reasonable) restrictions for the on-line algorithm. That lower bounds meet the upper bound for graphs that can be colored with an even number of colors. We conjecture that the upper bound is tight for any possible on-line algorithm and for any chromatic number of the presented graphs. We even present a partial proof by induction where it can be seen that the cases in which the lower bound may fail to be true are limited. Moreover, we see a proof for the lower bound in the case where $s = 1$.

This new model opens some new questions to be answered: What is the competitive ratio in this model with limited advice bits? How do other classical on-line problems behave with this new model?

References

[1] Susanne Albers. Online Algorithms: A Survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[2] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph (Seffi) Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 267–276, Washington, DC, USA, 2011. IEEE Computer Society.

[3] D Bean. Effective coloration. *J.Symbolic Logic*, 41:469–480, 1976.

[4] Maria Paola Bianchi, Hans Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. Online coloring of bipartite graphs with and without advice. *Algorithmica*, 70(1):92–111, 2014.

[5] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *Automata, languages and programming. Part I*, volume 6755 of *Lecture Notes in Comput. Sci.*, pages 207–218. Springer, Heidelberg, 2011.

[6] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Algorithms and Computation*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer Berlin Heidelberg, 2009.

[7] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*, volume 2. Cambridge University Press, 1998.

[8] Iwona Cieslik. *On-line graph coloring*. PhD thesis, Jagiellonian University, 2006.

[9] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin Heidelberg, 2008.

[10] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theoretical Informatics and Applications*, 43:585–613, 7 2009.

[11] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[12] Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *Proceedings of the 6th International Conference on Language and Automata Theory and Applications*, LATA'12, pages 228–239, Berlin, Heidelberg, 2012. Springer-Verlag.

[13] Magnús M. Halldórsson. Parallel and on-line graph coloring. *J. Algorithms*, 23(2):265–280, May 1997.

[14] Magnus M. Halldórsson and Mario Szegedy. Lower bounds for on-line graph coloring. *Theoretical Computer Science*, 130(1):163–174, 1994.

[15] J. Hromkovič. *Design and analysis of randomized algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2005. Introduction to design paradigms.

[16] Lucia Keller. *Complexity of optimization problems: Advice and approximation*. PhD thesis, ETH, Zürich, 2014.

[17] H. A. Kierstead and W. T. Trotter. On-line graph coloring. In Lyle A. McGeoch and Daniel D. Sleator, editors, *On-line algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 247–258. AMS–DIMACS–ACM, 1992.

[18] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, September 1995.

[19] László Lovász, Michael Saks, and W.T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1â3):319 – 325, 1989.

[20] Mark Manasse, Lyle McGeoch, and Daniel Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 322–333, New York, NY, USA, 1988. ACM.

[21] Claire Mathieu. Online Algorithms: Sky rental. `http://cs.brown.edu/~claire/Talks/skirental.pdf`, 2008.

[22] Avery Miller. Online Graph Colouring. *Online*, pages 1–16, 2004.

[23] Joseph Naor. Lecture: Online algorithms and the k-server problem. `http://www.cs.princeton.edu/~zdvir/apx11slides/naor-scribe.pdf`, 2011.

[24] Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice Complexity of the Online Vertex Coloring Problem. 2012.

[25] Sundar Vishwanathan. Randomized online graph coloring. *Journal of algorithms*, 13(4):657–669, 1992.

[26] Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *J. ACM*, 30(4):729–735, October 1983.