Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Selected Topics from Advice Complexity

Diploma thesis

2014 **Michal Petrucha**

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Selected Topics from Advice Complexity

Diploma thesis

Study programme:	Informatics
Field of study:	2508 Informatics
Department:	Department of Computer Science
Supervisor:	RNDr. Michal Forišek, PhD.

Bratislava, 2014 **Michal Petrucha**

Abstrakt

Sem patrí abstrakt v slovenčine.

Kľúčové slová: nejaké sem treba doplniť

Abstract

This is where the abstract in English will go.

Key words: and some key words as well

Acknowledgements

I want to thank Valve for their Holiday sales on Steam which gave me a lot of options to procrastinate instead of working on this thesis.

Contents

List of Figures

List of Tables

Introduction

This is the place for an introduction...

Chapter 1

First chapter

[TODO: **Name this chapter!**]

[TODO: **Write this introduction**] This is the sectionless introduction to the first chapter. Probably a word or two about how we are going to give a brief overview of what online problems and advice complexity are and the appropriate definitions upon which this thesis is based.

1.1 Online problems

One of the countless ways to categorize algorithmic problems is into *offline* and *online problems*. Offline problems are those where the algorighm can access the whole input instance before yielding the output. On the other hand, the instance of an online problem is revealed to the algorithm in smaller pieces and after each piece a partial solution has to be produced. This partial solution cannot be changed later.

A slightly different way of looking at online algorithms is that the algorithm waits for an input query, processes it and outputs an answer to this query immediately. Then it waits for another query and repeats the process until there is nothing more to do.

Solving a problem online is obviously more difficult than solving the same

instance knowing the whole input at once. For many problems it is even impossible to compute the optimal partial solutions without the knowledge of the rest of the input sequence. Therefore we define a *competitive ratio* of an algorithm, which is the quotient of the cost of the solution produced by the online algorithm and the cost of the optimal solution. An optimal solution is one produced by an optimal offline algorithm. Since the competitive ratio can depend on the input instance, we study the worst competitive ratio an algorithm achieves.

We consider randomized online algorithms as well. In this case we examine the expected competitive ratio.

Let us describe a few examples of simple online problems to give a better idea of what they are about. A very simple online problem is ski rental. Suppose we are going to take an unknown number of ski trips and we do not own a pair of skis. Renting a pair of skis for a single trip costs 1, buying one costs s. The input consists of a sequence of queries "take a ski trip" and after each query an answer is expected that is either "rent", "buy" or "use skis already bought". In [Kar92] it is proved that to minimize the competitive ratio the algorithm needs to rent for the first $s-1$ rounds and then buy a pair of skis; this way, the competitive ratio is $\frac{2s-1}{s} \approx 2$.

Another classic online problem is the paging problem. Assume a two-level memory divided into uniform, fixed-size pages. Let k be the number of pages that can fit within the fast memory. The input consists of n queries, each specifying a page we want to access. This page needs to be loaded into the fast memory thus replacing a page called a victim (unless it is there already). The goal is to minimize the number of page faults, i.e. the number of times we need to load a page from the slow memory into the fast level.

In [ST85] the authors show that for any deterministic online algorithm solving the paging problem it is possible to construct an instance using $k+1$ pages where the online algorithm will produce a page fault on each request

by always choosing the page that is not in the fast memory. However, an offline algorithm can decrease the number of page faults by at least a factor of k, therefore the competitive ratio of any online paging algorithm is at least k.

In addition, in [ACN96] the authors describe a randomized online algorithm for the paging problem whose competitive ratio is H_k.

1.2 Advice complexity

In the previous section we showed that there are problems which cannot be solved optimally with an online algorithm. This means that having access to the whole of the input sequence can help the algorithm to provide better partial results. However, sometimes it may not be necessary to access the whole input sequence in order to compute the optimal solution, in some cases a significantly smaller amount of information is required.

That is why a computational model of *online algorithms with advice* has been introduced in [DKP08]. In this model, the online algorithm is assisted by an oracle with access to the whole input sequence. The oracle has unlimited computational power and provides the online algorithm with information about the input sequence that it requires. We define the *advice complexity* of an online algorithm as the minimal number of bits it needs to read from the oracle in order to solve the problem optimally. The advice complexity of an online problem is then defined as the lowest advice complexity of online algorithms solving it.

There have been multiple formal definitions of this model with various drawbacks. [DKP08] contains a definition in which the online algorithm has access to a finite binary advice tape. That means, however, that additional information can be encoded into the length of the advice tape. In [Eme+09] the authors define a slightly different model where the online al-

gorithm receives the same amount of information in each round. This makes it impossible to use a sublinear amount of advice. The model used in this thesis has been defined in [B+09]; this model uses an infinite advice tape and we measure the amount of bits the algorithm accesses.

To illustrate the effect of advice we show the amount of advice required to solve the two aforementioned online problems optimally. The ski rental problem is trivial to solve using a single bit of advice – this bit tells the algorithm whether there will be at least s queries. The online algorithm reads this before answering the first query and it knows immediately whether to buy a pair of skis or just rent them on each trip.

The paging problem is slightly more complex to solve optimally using advice. Following the proof in [DKP], this can be done using n bits of advice. The oracle calculates one optimal solution to the input instance and assigns a single bit to each request. This bit indicates whether the page will be accessed again before it is replaced by another one in the optimal solution, such pages are called active; if the page will not be accessed again, it is passive. The online algorithm then just picks a passive page as the victim on each page fault.

Thus far we only covered the amount of advice required to obtain the optimal solution using an online algorithm. However, it is also useful to examine the amount of advice required to obtain a certain competitive ratio and the tradeoff between these two. In this thesis we will study this aspect as well.

Another possible area of research is the amount of advice required to solve a *partially online problem*. This is a special case of an online problem where only a part of the input instance is served in pieces and at some point the whole rest of the input is served in a single piece.

1.3 Online graph coloring

The main focus of this thesis is on the problem of online graph coloring. Obviously, the difficulty of this problem depends greatly on any assumptions we make on the input instance, e.g. restrictions on the class of graphs, e.g. trees, bipartite graphs, cycles or a relationship between the number of vertices and the number of edges, or the order in which their vertices are revealed to the online algorithm. All these assumptions provide the algorithm with additional information. This means that by comparing the advice required to solve these special cases to the advice complexity of the general case we can quantify the amount of information provided by a particular set of assumptions.

An online graph coloring algorithm works roughly as follows. In each turn a single vertex of the input graph is revealed to the algorithm and it has to assign a color to this vertex. More precisely, assuming the vertices of a graph are ordered in a sequence, in t-th turn the algorithm has the knowledge of the subgraph induced by the first t vertices in this sequence. That means, all edges are revealed as soon as both of their ending vertices are known.

The order in which vertices are revealed is referred to as the *presentation order*. In the most general case, the vertices will appear in a fully arbitrary order. We can restrict this to a connected presentation order, which means that in each turn the vertex currently revealed is connected to at least one vertex revealed previously. This can be restricted even further to the order in which a depth-first search (DFS) or a breadth-first search (BFS) will visit vertices. Another common presentation order is when the sequence of vertices is sorted by their degrees.

In this thesis, each time we study a particular online graph coloring problem, we specify explicitly both the class of graphs and the presentation order. For instance, OnlineColoring(bipartite, connected) denotes that the

problem is restricted to bipartite graphs and their vertices are revealed in a connected order. As a special case, OnlineColoring(any, any) denotes the most general version of the problem where no assumptions are made at all.

1.4 Formal definitions and notations

This section will contain the required definitions. [TODO: **ADD THEM!!!**]

This thesis follows the definitions from [FKS12]. They are provided in this section for reference.

To denote the solution produced by an algorithm A for an instance I we use $A(I)$. The cost of a solution S will be denoted by $C(S)$. An optimal solution for I will be denoted by $Opt(I)$. We will use $E[X]$ to denote the expected value of a random variable X.

Definition 1.1. *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 previous definition can easily be extended to randomized algorithms. For each instance I we require $E[C(A(I))] \leq c \cdot C(Opt(I)) + \alpha$. We say that the expected competitive ratio of A is the smallest value of c satisfying the above inequality.

Definition 1.2. *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 $A^\varphi(X) = Y$.*

As stated earlier, the computation of A can be interpreted as a series of turns, where in the i-th turn the algorithm reads x_i and yields y_i using all the information read so far and possibly some additional bits from the advice string φ. It is worth noting that the definition does not restrict the computational power of A.

Definition 1.3. *The advice complexity of A is a function s such that $s(n)$ is the smallest value such that for each input sequence of size n there is an advice string φ such that the algorithm A examines at most the first $s(n)$ bits of 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 1.4. *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 $C(A^{\varphi}(I)) \leq c \cdot C(Opt(I)) + \alpha$.*

The previous two definitions suggest another way to look at algorithms with advice. They have one similarity with the traditional definition of a non-deterministic Turing machine: in both cases the definitions allow the existence of an unlimited number of possible computations for a single input instance and all except "the right one" are disregarded. In the case of a non-deterministic Turing machine this means that in case there are multiple possibilities for the next configuration, we always choose the one that leads to an accepting state. In the case of online algorithms with advice this means that although there is an infinite number of advice strings, we always choose the one that leads to the optimal solution (or the required competitive ratio) with the smallest number of accessed bits.

This observation indicates that there is a direct connection between the number of required advice bits and the number of non-deterministic decisions a non-deterministic algorithm would have to make in order to find the

expected solution.

Definition 1.5. *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.*

When anlyzing a variant of OnlineColoring, we always need to specify the class of graphs it is restricted to and the presentation order. We denote this using OnlineColoring(X, Y) where X is the class of graphs G will belong to and Y is the presentation order. For the class of graphs we will use its common name (e.g., "bipartite", "planar") with the special class called "any" meaning that there is no restriction on G at all. For the presentation order we will use "connected", "bfs", "dfs" and "max-degree" with meanings as discussed earlier and, again, "any" with the meaning that the vertices may be presented in a fully arbitrary order.

The value of n is not known to the online algorithm beforehand. The reason for this is that it would provide the algorithm with additional information about the input instance which may (and in some cases does) affect the advice complexity of the problem.

Chapter 2

Known results and related work

[TODO: **Write this chapter introduction**]

In this chapter we give an overview of currently known results and advances in the field of advice complexity.

2.1 Common analysis and proof techniques

Despite the fact that the computational model of online algorithms with advice has been only concieved a few years ago it is already possible to notice the emergence of common techniques to analyze online problems and find lower and upper bounds for their advice complexity.

To find an upper bound the most straightforward method is, same as with other complexity metrics, to find an algorithm which solves the problem and then determine its advice complexity. Any optimal algorithm cannot then have any worse advice complexity. While this method is obvious, it is often the most demonstrative one.

One of the most basic approaches to find the lower bound on the advice complexity of a particular online problem is to find a set of instances with the following properties:

(i) for a given non-negative integer k the prefixes $(x_1^{(i)}, \ldots, x_k^{(i)})$ of instances $I^{(i)}$ are equal, i.e., for two instances $I^{(i)} \neq I^{(j)}$, for each l such that $1 \leq l \leq k$, the members $x_l^{(i)}$ and $x_l^{(j)}$ are equal

(ii) for each pair of instances $I^{(i)} \neq T^{(j)}$ there are no optimal solutions $Opt(I^{(i)}) = (y_1^{(i)}, \ldots, y_{n_i}^{(i)})$, $Opt(I^{(j)}) = (y_1^{(j)}, \ldots, y_{n_j}^{(j)})$ such that

$$(y_1^{(i)}, \ldots, y_k^{(i)}) = (y_1^{(j)}, \ldots, y_k^{(j)})$$

In other words, we find a set of instances such that the algorithm can't possibly distinguish the prefixes of these instances but for each instance a unique solution needs to be yielded in the prefix already. To achieve this, the advice string must necessarily be used. If the size of this set of instances is m, at least $\log_2 m$ advice bits need to be accessed which gives a lower bound on the advice complexity of the problem.

This technique is used in various proofs in [FKS12]. These will be discussed in more detail in the following sections.

In [B$\ddot{+}$12] the authors use reductions to a simpler problem that is easier to analyze as a method to prove lower bounds. Specifically, they picked the string guessing problem in two variants.

Definition 2.1 (String Guessing with Known History)**.** *The string guessing problem with known history over an alphabet Σ of size $q \geq 2$ (denoted as q-*SGKH*) is defined as follows. The input instance $I = (n, d_1, \ldots, d_n)$ consists of an integer n specifying the length of the instance and a sequence of n characters, where $d_i \in \Sigma, 1 \leq i \leq n$. Let A be an online algorithm that solves q-*SGKH*, then $A(I) = (y_1, \ldots, y_n, \_)$, where $y_i \in \Sigma$. We define the cost of a solution as the Hamming distance between the sequence $(y_1, \ldots, y_n)$ and the sequence $(d_1, \ldots, d_n)$, i.e. the number of wrongly guessed characters.*

Definition 2.2 (String Guessing with Unknown History)**.** *The string guessing problem with unknown history over an alphabet Σ of size $q \geq 2$ (denoted*

as q-SGUH*) is defined as follows. The input instance* $I = (n, ?_2, \ldots, ?_n, d)$ *consists of an integer* n *specifying the length of the instance,* $n-1$ *queries without additional information and a string* $d = d_1 d_2 \ldots d_n$*, where* $d_i \in \Sigma, 1 \leq i \leq n$*. Let* A *be an online algorithm that solves* q-SGUH*, then* $A(I) = (y_1, \ldots, y_n, \_)$*, where* $y_i \in \Sigma$*. We define the cost of a solution as the Hamming distance between the sequence* $(y_1, \ldots, y_n)$ *and the sequence* $(d_1, \ldots, d_n)$*.*

Both q-SGKH and q-SGUH consist of $n+1$ queries where for the first n queries the algorithm is expected to guess a single character of the instance; for the last query no meaningful response is expected, its purpose is only to reveal the input string to allow an offline algorithm to guess the whole string correctly. The only difference between the two variants is that in q-SGKH it is revealed whether the algorithm guessed correctly after each guess and in q-SGUH this is revealed in the last turn.

For the sake of simplicity, we may sometimes speak about the input string $d = d_1 d_2 \ldots d_n$ instead of the corresponding input instance $I = (n, d_1, \ldots, d_n)$ in the case of q-SGKH or $I = (n, ?_2, \ldots, ?_n, d)$ in the case of q-SGUH.

We formulate the following simple observation.

Observation 2.3. *It should be noted that any upper bound on the advice complexity of* q-SGUH *is also an upper bound on the advice complexity of* q-SGKH *– any algorithm that solves* q-SGUH *can be used to solve* q-SGKH *as well, simply ignoring the characters provided in each query. Similarly, any lower bound for* q-SGKH *is also a lower bound for* q-SGUH*.*

Theorem 2.4. *The advice complexity of* q-SGUH *is at most* $\lceil n \log_2 q \rceil$*.*

Proof. We prove this theorem by describing an algorithm A using $\lceil n \log_2 q \rceil$ bits of advice which solves both q-SGKH and q-SGUH.

The total number of strings of length n is q^n. These can be sorted in a lexicographic order in which each instance has a position. To encode this

position, $\lceil n \log_2 q \rceil$ bits are required.

Therefore, after receiving the number n in the first query, A reads the position m of the string from the advice string and enumerates the first m strings of length n in lexicographic order until it finds the correct one. Then it just yields one character from the string per query. □

Theorem 2.5. *The advice complexity of* q-SGKH *is at least* $\lceil n \log_2 q \rceil$.

Proof. We prove this by contradiction. Suppose there is an algorithm A which solves q-SGKH using m bits of advice, $m < \lceil n \log_2 q \rceil$. The total number of instances of length n is q^n. However, using m bits of advice it is possible to only encode $2^m \leq 2^{\lceil n \log_2 q \rceil - 1} < 2^{n \log_2 q} = q^n$ different values. Therefore, there are two input strings d, d' where the same m-bit advice string φ leads to the optimal solution.

Consider the first position i at which strings d and d' differ, i.e., $d_i \neq d'_i$. Since A gives the optimal result for the input string d, in the i-th turn it emits d_i. However, since up until the i-th turn, the input is the same for d' as well and since the advice string is also the same, A is in exactly the same state in the i-th turn when processing d' as it is when processing d. Therefore, for the input string d', A outputs d_i in the i-th turn as well. This contradicts the assumption that A provides an optimal solution for d'. □

The following corollary follows from the previous two theorems and observation 2.3.

Corollary 2.6. *The advice complexity of both* q-SGKH *and* q-SGUH *is* $\lceil n \log_2 q \rceil$.

It is easy to see that no deterministic online algorithm without advice can guarantee to quess even a single character right for alphabets of size $q \geq 2$. This can be seen by running the algorithm against an adversary which builds the string d by picking a character different from the one a deterministic algorithm emits in each turn.

However, by providing an online algorithm with a constant amount of advice we can already reach a number of correctly guessed characters linear in the input length.

Theorem 2.7. *Using* $\lceil \log_2 q \rceil$ *bits of advice it is possible to guess at least* $\lceil \frac{n}{q} \rceil$ *characters correctly.*

Proof. Using $\lceil \log_2 q \rceil$ it is possible to encode a single character from Σ. Therefore, by encoding the character with the most occurrences in the input string into the $\lceil \log_2 q \rceil$ advice bits and then emitting this character in each turn we can ensure that the algorithm guesses at least $\lceil \frac{n}{q} \rceil$ correctly. □

The following lower bounds on the number of advice bits required to guarantee that an algorithm guesses at least a certain amount of characters right have been established.

Theorem 2.8. *To guarante that an online algorithm A guesses at least* αn *characters right for an instance of* q-SGKH *of length* n *where* $\frac{1}{q} \leq \alpha < 1$*, A needs to access at least the following number of advice bits:*

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha \log_q \alpha\right) n \log_2 q = (1 - H_q(1-\alpha)) n \log_2 q$$

For a proof of this theorem please refer to [B+̈12].

According to observation 2.3, the same lower bound applies to q-SGUH as well.

The same paper also establishes the following upper bounds on the amount of advice required to guarantee a certain rate of success.

Theorem 2.9. *There is an online algorithm that guesses at least* αn *characters right for an instance of* q-SGUH *of length* n *where* $\frac{1}{q} \leq \alpha < 1$*, which accesses at most the following number of advice bits:*

$$\left\lceil (1 - H_q(1-\alpha)) n \log_2 q + 3 \log_2 \frac{n}{2} + \log_2(\ln q) + \frac{1}{2} \right\rceil$$

Again, thanks to observation 2.3, the upper bound is also valid for q-SGKH. The full proof can be found in [B$\ddot{+}$12].

The paper then uses these results to establish lower bounds on the advice required to attain a certain competitive ratio for the online version of the maximum clique problem and the online set cover problem.

[TODO: **Study more analysis techniques.**]

2.2 Known Results in Online Graph Coloring

[TODO: **Let's do this.**]

Most of the results provided in this section are discussed in more detail in [FKS12].

2.2.1 General graphs

The following asymptotically tight estimates on the advice complexity of the most general case of online graph coloring have been established.

Theorem 2.10. *There is an online algorithm with advice which solves* OnlineColoring(any, any) *using* $n \log_2 n - n \log_2 \log_2 n + O(n)$ *bits of advice.*

The general idea is to encode the position of an optimal coloring in a lexicographically sorted list of all partitions of the set of vertices on the advice tape.

Theorem 2.11. *The advice complexity of* OnlineColoring(any, bfs) *is at least* $n \log_2 n - n \log_2 \log_2 n + O(n)$.

The proof of this theorem uses the idea outlined in section 2.1. It is possible to create a set of instances that an online algorithm cannot distinguish based on their prefixes up to a certain length but that require unique colorings in these prefixes already.

These results are crucial in order to quantify how much a restriction on the class of graphs simplifies the coloring problem by means of advice complexity.

2.2.2 Bipartite graphs

As a reminder, bipartite graphs are those that can be colored using two colors.

[TODO: **Mention the cases when advice is not needed.**]

Theorem 2.12. *There is a deterministic online algorithm for* OnlineColoring(bipartite, connected) *without advice.*

Proof. The algorithm for an optimal coloring is trivial. For the first vertex it picks an arbitrary color and afterwards, for each vertex there is at least one neighbor whose color has already been assigned. Therefore the algorithm just picks the other color. □

This result shows that for bipartite graphs it does not really make any sense to analyze any of the connected presentation orders. However, for presentation orders without any restrictions this class of graphs is still interesting from the point of view of advice complexity.

2.2.3 Paths

Paths are a subclass of bipartite graphs, therefore it is only interesting to analyze the most general presentation order.

Theorem 2.13. *The advice complexity of* OnlineColoring(path, any) *is* $\lceil \frac{n}{2} \rceil$.

[TODO: **Why?**]

Conclusion

Once the rest of the thesis is written, this is the place where a witty conclusion will appear.

[TODO: **Make the bibliography consistent.**]

Bibliography

[ACN96] Dimitris Achlioptas, Marek Chrobak, and John Noga. "Competitive analysis of randomized paging algorithms". In: *Algorithms — ESA '96*. Ed. by Josep Diaz and Maria Serna. Vol. 1136. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1996, pp. 419–430. ISBN: 978-3-540-61680-1. DOI: 10.1007/3-540-61680-2_72. URL: http://dx.doi.org/10.1007/3-540-61680-2_72.

[B+09] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. "On the Advice Complexity of Online Problems". In: *Proceedings of the 20th International Symposium on Algorithms and Computation*. ISAAC '09. Honolulu, Hawaii: Springer-Verlag, 2009, pp. 331–340. ISBN: 978-3-642-10630-9. DOI: 10.1007/978-3-642-10631-6_35. URL: http://dx.doi.org/10.1007/978-3-642-10631-6_35.

[B+12] H.J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. "The String Guessing Problem as a Method to Prove Lower Bounds on the Advice Complexity". In: *Electronic Colloquium on Computational Complexity*. 2012.

[DKP] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. "Measuring the problem-relevant information in input". In: *RAIRO - Theoretical Informatics and Applications* 43 (03), pp. 585–613.

ISSN: 1290-385X. DOI: 10.1051/ita/2009012. eprint: http://www.rairo-ita.org/article_S0988375409000125. URL: http://dx.doi.org/10.1051/ita/2009012.

[DKP08] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. "How much information about the future is needed?" In: *Proceedings of the 34th conference on Current trends in theory and practice of computer science*. SOFSEM'08. Nový Smokovec, Slovakia: Springer-Verlag, 2008, pp. 247–258. ISBN: 3-540-77565-X, 978-3-540-77565-2. URL: http://dl.acm.org/citation.cfm?id=1785934.1785957.

[Eme+09] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. "Online Computation with Advice". In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*. ICALP '09. Rhodes, Greece: Springer-Verlag, 2009, pp. 427–438. ISBN: 978-3-642-02926-4. DOI: 10.1007/978-3-642-02927-1_36. URL: http://dx.doi.org/10.1007/978-3-642-02927-1_36.

[FKS12] Michal Forišek, Lucia Keller, and Monika Steinová. "Advice Complexity of Online Coloring for Paths [TODO: replace with the expanded article]". In: *Language and Automata Theory and Applications (LATA 2012)*. 2012, pp. 228–239. ISBN: 978-3-642-28331-4.

[Kar92] R.M. Karp. "On-line algorithms versus off-line algorithms: How much is it worth to know the future". In: *Proc. IFIP 12th World Computer Congress*. Vol. 1. 992. Madrid, Spain. 1992, p. 1.

[ST85] Daniel D. Sleator and Robert E. Tarjan. "Amortized efficiency of list update and paging rules". In: *Commun. ACM* 28.2 (Feb.

1985), pp. 202–208. ISSN: 0001-0782. DOI: 10.1145/2786.2793.
URL: http://doi.acm.org/10.1145/2786.2793.