Online Algorithms with Advice: A Survey

JOAN BOYAR, LENE M. FAVRHOLDT, CHRISTIAN KUDAHL, KIM S. LARSEN, and JESPER W. MIKKELSEN, University of Southern Denmark

19

In online scenarios requests arrive over time, and each request must be serviced in an irrevocable manner before the next request arrives. Online algorithms with advice is an area of research where one attempts to measure how much knowledge of future requests is necessary to achieve a given performance level, as defined by the competitive ratio. When this knowledge, the advice, is obtainable, this leads to practical algorithms, called semi-online algorithms. On the other hand, each negative result gives robust results about the limitations of a broad range of semi-online algorithms. This survey explains the models for online algorithms with advice, motivates the study in general, presents some examples of the work that has been carried out, and includes an extensive set of references, organized by problem studied.

CCS Concepts: • **Theory of computation → Online algorithms;**

Additional Key Words and Phrases: Online algorithms, advice complexity, competitive analysis, approximation

ACM Reference Format:
Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. 2017. Online algorithms with advice: A survey. ACM Comput. Surv. 50, 2, Article 19 (April 2017), 34 pages.
DOI: http://dx.doi.org/10.1145/3056461

1. INTRODUCTION

In online scenarios, an algorithm solves some optimization problem. In traditional algorithmics, all input is available from the beginning, but for online problems, the input is presented with one request arriving at a time. On receiving a request, the online algorithm must make some irrevocable decision, generally without any knowledge of future requests, attempting to minimize or maximize some global objective function. This models many real-life problems; for instance, problems where agents request some resource from a central system that is responsible for allocating them. Examples include paging, car rental, and cell phone frequency allocation.

In many practical scenarios, there actually is some knowledge about future requests that is available. In paging, for example, for a database organized as a search tree, the root node will certainly be accessed again soon. If the algorithm is given this information as advice, then it can keep the root in cache, improving performance. When considering the advice complexity of an online problem, one considers how much advice is necessary to improve the worst-case performance of algorithms.

This work is supported in part by the Danish Council for Independent Research, Natural Sciences, grant DFF-1323-00247, and the Villum Foundation, grant VKR023219.
This work has previously appeared in ACM SIGACT News, 47(3): 93–129, 2016.
Authors' addresses: Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark; emails: {joan, lenem, kudahl, kslarsen, jesperwm}@imada.sdu.dk.

DOI: http://dx.doi.org/10.1145/3056461

There are various measures for the quality of online algorithms [Dorrigiv and López-Ortiz 2005; Boyar et al. 2015b], but the most standard is the competitive ratio [Sleator and Tarjan 1985; Karlin et al. 1988], which is the online analog of the approximation ratio. The performance of an online algorithm ALG is compared to the performance of an optimal offline algorithm OPT. Let ALG(I) denote the value of the objective function applied to the output computed by ALG when given the request sequence I as input. Define OPT(I) similarly. For minimization problems, ALG is *c-competitive* if there exists a constant[1] α, such that for all finite request sequences, I,

$$\text{ALG}(I) \leq c \cdot \text{OPT}(I) + \alpha.$$

Similarly, for maximization problems,

$$\text{OPT}(I) \leq c \cdot \text{ALG}(I) + \alpha.$$

In both cases, if the inequality holds with $\alpha = 0$, the algorithm is *strictly c-competitive*. The (*strict*) *competitive ratio* of an algorithm is the infimum over all values of c for which the algorithm is (strictly) c-competitive.

Note that competitive analysis is a worst-case measure. Thus, it can be useful to think of the input as being generated by a malicious adversary who knows ALG. When studying an online problem, it is customary to consider both deterministic and randomized online algorithms. A randomized online algorithm ALG is c-competitive if it is c-competitive in expectation, that is, if there exists a constant α such that $\mathbb{E}[\text{ALG}(I)] \leq c \cdot \text{OPT}(I) + \alpha$ for all inputs I. This corresponds to the adversary being *oblivious* to the random choices made by the algorithm. See Borodin and El-Yaniv [1998] and Ben-David et al. [1994] for further details.

We use n to denote the length of an input sequence. In some cases, the competitiveness, c, is a function of n. When referring to *optimal* algorithms or solutions, we always refer to a solution that could have been produced by an optimal offline algorithm.

Advice Complexity. Note that there are three basic assumptions underlying competitive analysis: The input is adversarial, decisions are irrevocable, and an online algorithm knows nothing about the requests before they arrive. Many possible ways of relaxing one or more of these assumptions have been studied. In the advice complexity model, the "no knowledge" assumption is relaxed in a problem-independent and quantitative way (while the two first assumptions remain unaltered). In this model, an *online algorithm with advice* is provided with some bits of advice about the request sequence I. These bits are provided by a trusted *oracle* that knows the entire request sequence and has no computational limitations (the formal definition of the advice complexity model(s) can be found in Section 2). Obviously, an online algorithm with advice may perform better than a traditional online algorithm, but if the amount of advice it receives from the oracle is bounded, then it may perform less well than an optimal offline algorithm. The *advice complexity* of an algorithm is the maximum number of bits read by that algorithm on any request sequence of a given length.

Motivation. Given an online problem considered in the advice model, the major question asked is as follows:

How many bits of advice are necessary and sufficient to obtain a competitive ratio c?

This includes determining the number of bits to become optimal (strictly 1-competitive) or to beat the best deterministic or randomized algorithms. It also includes

[1]To avoid any possible confusion, when online problems are described in a parameterized form (using k for the size of the cache in the paging problem, for example), we quantify over these problem-specific parameters first. Thus, α may depend on such parameters.

considerations in the other direction, such as determining what can be obtained using a constant number of bits, for instance.

In what follows, we have attempted to list the most important reasons the advice complexity model is interesting and relevant.

—Lower bounds on advice complexity give *robust bounds on what is possible using semi-online algorithms*. The value of certain upper bounds, on the other hand, may be questioned for the following reason. Since the advice is not restricted except by its size, it can happen that algorithms with advice are not of practical interest; they sometimes rely on advice that we do not expect to possess. However, lower bounds in the advice complexity model are very strong exactly because we do *not* impose any restrictions on the type of advice: They apply to *any* possible information about the request sequence that can be encoded using a sufficiently small number of bits. Thus, they can be very relevant for the study of semi-online algorithms (see Section 3).

—There are *strong connections between online algorithms with advice and randomized online algorithms*. For example, important open problems regarding randomized online algorithms (such as the best possible competitive ratio of a randomized k-Server or List Update algorithm) can be stated equivalently as problems about online algorithms with advice. Some results on online algorithms with advice lead to new lower and/or upper bounds on randomized online algorithms (see Section 4).

—It may be possible to use online algorithms with advice in settings where it is feasible to run multiple algorithms and output the best solution. For example, Boyar et al. [2014] gave an algorithm using two bits of advice to choose among three algorithms for List Update, obtaining a competitive ratio better than any deterministic online algorithm. Using a List Update algorithm as a post-processing step of the Burrows-Wheeler Transform, the algorithm performing the compression can compare the results obtained from more than one algorithm. It can then choose the best, and include information on which algorithm was actually used as a prefix of the compressed data. In this way, it is possible at a later stage to read this information and uncompress; see Kamali and López-Ortiz [2014b].

—Suppose that an online algorithm with b bits of advice runs in time $O(T(n))$. Then one may *convert the algorithm into an offline approximation algorithm* with time complexity $O(2^b \cdot T(n))$ by running the algorithm on all possible 2^b advice strings. For Reordering Buffer Management, the currently fastest $(1+\varepsilon)$-approximation algorithm is obtained in exactly this way by using an online algorithm with advice due to Adamaszek et al. [2016].

—Online algorithms with advice may be viewed as *non-deterministic* online algorithms, since one may think of the online algorithm as non-deterministically guessing the advice which it then uses to compute its output. Thus, the advice complexity of a problem measures the amount of non-determinism an online algorithm needs to achieve a given solution quality. Understanding the power of non-determinism (as compared to determinism and randomization) is one of the main challenges and most well-studied problems in theoretical computer science (P vs. NP, DFA vs. NFA, etc.). It seems natural to try to improve our understanding of how non-determinism may help when solving problems in an online environment.

—The first *complexity classes* for online algorithms have been based on advice complexity (see Section 7.3). The first class, Asymmetric Online Covering (AOC), contains many problems where the algorithm's irrevocable decisions are whether to accept or reject each request. All AOC-complete problems, such as Vertex Cover, Independent Set, Dominating Set, Cycle Finding, and Disjoint Path Allocation, have essentially the same advice complexity (linear in $\frac{n}{c}$, where c is the desired competitive ratio). Weighted versions of AOC-complete minimization problems are even harder. These complexity classes are not only interesting with respect to advice; in the online setting without advice, the complete problems are also exceptionally hard.

We now give two examples of simple advice complexity results. Note that the minimum number of bits required to encode the decisions OPT makes is an obvious upper bound for the amount of advice needed to be optimal. Sometimes, though, encoding OPT's decisions requires fewer bits than one first expects. PAGING is an example of this.

Example 1. In PAGING, there is a set of N pages. A request sequence arrives online; each request is a page. The algorithm has a cache that starts out empty and can contain up to $k < N$ pages. When a page not in cache is requested, the page must be brought into cache, at a cost of 1. This is referred to as a page fault. If the cache is already full, then the algorithm must select another page from its cache to evict to make room for the new page; this is the irrevocable online decision.

The optimal offline PAGING algorithm is Longest Forward Distance (LFD) [Belady 1966] that always evicts the page, which will not be requested for the longest time. For deterministic online PAGING algorithms without advice, the best attainable competitive ratio is k [Sleator and Tarjan 1985], and for randomized algorithms it is H_k [Achlioptas et al. 2000; McGeoch and Sleator 1991], where $H_k \approx \ln k$ is the kth harmonic number.

How many advice bits does an algorithm need to be optimal? Clearly, $\lceil \log k \rceil n$ bits of advice are enough to simulate LFD by specifying the index in cache of the page to evict (if any) at each request. However, using a more clever encoding, one can obtain the following result:

THEOREM 1.1 (DOBREV ET AL. [2009]). *There is an optimal* PAGING *algorithm,* ALG, *which reads n bits of advice.*

PROOF. Using a fixed optimal solution for the given input, the oracle provides one bit of advice per request. That bit indicates whether, in the optimal solution, the page requested is kept in cache until the next time it is requested. ALG only evicts pages that will cause faults on their next request in the optimal solution as well. Thus, ALG is optimal. □

Figure 1 shows that for the question of the number of advice bits necessary and sufficient to achieve a certain competitive ratio, the "phase transitions" are essentially completely understood for PAGING. In Mikkelsen [2016], the following thresholds were proven: For any fixed cache size k, a large but constant total number of advice bits is sufficient to achieve a competitive ratio of $H_k + \varepsilon$ (for any $\varepsilon > 0$) and a linear number of advice bits is necessary to be better than H_k-competitive.

The connection between advice complexity and randomization is key to proving both the upper and lower bounds of H_k. For more details on PAGING, see Section 8.

Another problem with a sharp phase transition is UNIFORM KNAPSACK.

Example 2. In UNIFORM KNAPSACK, a sequence of requests arrives online. Each request is a value in the range $(0, 1]$. When a request arrives, the online algorithm decides irrevocably whether to pack it in the knapsack or reject it. The total size of accepted requests is not allowed to exceed 1, the size of the knapsack. The goal is to maximize the sum of the sizes of the accepted requests. UNIFORM KNAPSACK is the special case of the standard knapsack problem where the size and the value of requests are always equal to each other. The problem is analyzed using strict competitive analysis, since setting the additive constant in the definition of competitiveness equal to 1 would make any algorithm 1-competitive.

A deterministic algorithm without advice for UNIFORM KNAPSACK has unbounded competitive ratio [Marchetti-Spaccamela and Vercellis 1995]. However, with just one bit of

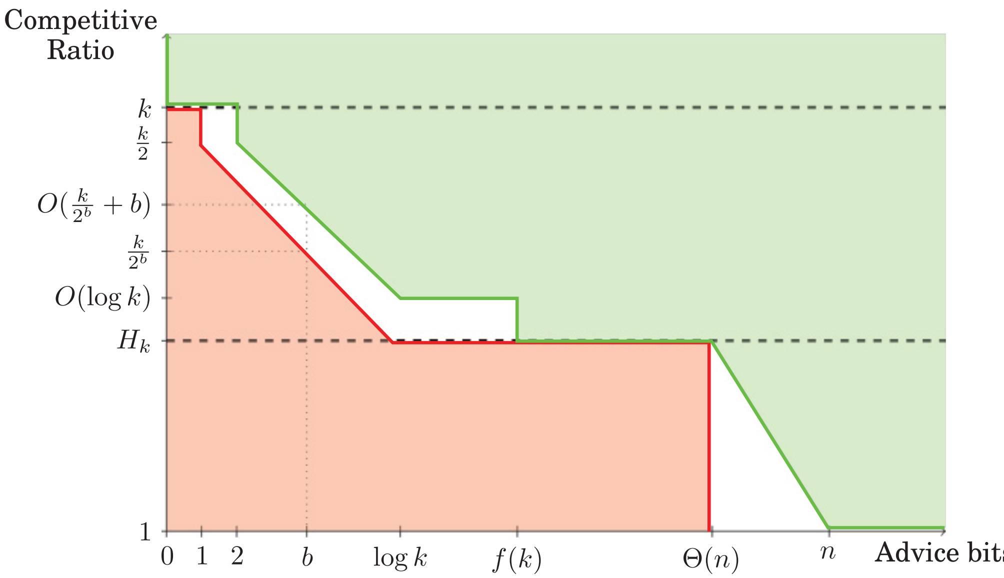

Fig. 1. The (asymptotic) tradeoff between competitive ratio and advice for PAGING. The function $f(k)$ is a rapidly growing function of k (but does not depend on n). Consider a tradeoff point (b, c) where b is a number of advice bits and c is a competitive ratio. The red area shows those tradeoffs that provably cannot be achieved. The green area shows those tradeoffs that we currently have algorithms achieving. It is an open problem whether tradeoffs in the white area are achievable. The horizontal dashed lines are the best possible competitive ratios of deterministic and randomized algorithms without advice.

advice it is possible to be 2-competitive. The one bit of advice is used to indicate whether there is an item in the input sequence of size at least $\frac{1}{2}$. That information might actually be available in some applications, so it can also be viewed as a *semi-online algorithm*, an online algorithm that knows something about the request sequence in advance.

THEOREM 1.2 (BÖCKENHAUER ET AL. [2014C]). *There exists a 2-competitive* UNIFORM KNAPSACK *algorithm that reads one bit of advice.*

PROOF. The oracle writes a 0 on the advice tape if no request of size at least $\frac{1}{2}$ will arrive and a 1 otherwise. The algorithm reads this one bit, b, of advice. If $b = 0$, then it packs each request if it has enough space left for it. If $b = 1$, then it rejects everything until it encounters an item of size at least $\frac{1}{2}$, which it packs (it may pack additional items that fit after this point).

For $b = 0$, if the total size of all requests arriving is less than 1, then the algorithm will be optimal; otherwise, its knapsack will be at least half full the first time it rejects a request. If $b = 1$, then the knapsack will again be at least half full. Thus, the algorithm is 2-competitive. □

In Böckenhauer et al. [2014c], it is also proven that to obtain a competitive ratio better than 2, $\Omega(\log n)$ advice bits are required. See Section 9 for more about the knapsack problem.

Organization of Survey. First, we introduce the advice models in Section 2. Then, we discuss the relationship between advice and semi-online algorithms in Section 3.

The strong connections between advice complexity and randomization, showing that results in either area can often be carried over to the other, is discussed in Section 4.

Some of the techniques that can be used in designing online algorithms with advice are discussed in Section 5, and lower bound techniques are discussed in Section 6.

A specific frequently used lower bound technique is based on String Guessing and its variants, which can also sometimes be used for proving upper bounds. These problems are discussed in Section 7, along with the first complexity classes for online algorithms, developed based on String Guessing results.

Note that all problems discussed in this survey are online unless explicitly stated otherwise. Metrical Task System problems, including k-Server and Paging, are discussed in Section 8. Bin Packing, Scheduling, and further results on Uniform Knapsack are discussed in Section 9. Graph Coloring is discussed in Section 10 and Graph Exploration in Section 11. Problems studied using advice complexity, along with references, are listed in the appendix.

2. ADVICE MODELS

In this section, we define advice models and describe the historical development. We compare the models and also discuss alternative views on what an advice model represents.

All models make use of a trusted oracle that knows the entire request sequence and has unlimited computational power. Bits that we refer to as *advice bits* are supplied to the algorithm by the oracle in some manner. These bits can be assumed to give a correct answer to any question the online algorithm poses. For example, an online algorithm could pose the question of how many future requests there are of a certain type and interpret the bits that are made available as the number of interest. Note that the oracle knows the online algorithm, so the questions are not explicitly asked; the oracle simply writes the answers and the algorithm reads them.

The term *advice complexity* for online algorithms was coined by Dobrev et al. [2009]. They suggest two models, referred to as the *helper mode* and the *answerer mode*. In the helper mode, the online algorithm receives a number of advice bits, which could be zero, prior to processing each request. The advice complexity is defined to be the total number of bits received from a perfectly designed oracle for the online algorithm to be optimal. The answerer mode is similar, except that advice bits are only given when requested by the online algorithm, in which case at least one bit is given. Note that the length itself of the bit sequence given as response to a request for advice may transfer information in both the helper and the answerer mode.

Allowing the online algorithm to gain knowledge from *not* receiving any bits (or, in general, receiving a varying number of bits) may be reasonable in some applications (see Dobrev et al. [2009] for a discussion), but it also introduces an additional complication that is not always desirable. Following the introduction of online algorithms with advice in Dobrev et al. [2009], two other models were suggested, both avoiding this complication in different ways.

One model was introduced in Hromkovič et al. [2010], with technical development in Böckenhauer et al. [2009]. They suggest using an infinite advice tape, written by the oracle; we refer to this model as the *Tape Model*. The online algorithm may consult this advice tape at its discretion, and the advice complexity is simply the number of bits read. The term "tape" is likely suggested by tradition; the important properties are that the algorithm has an unbounded supply of bits that it can receive one at a time on request and that there is no indication of an "end," that is, it is the algorithm that stops asking for bits, not the supply that runs out. In other words, the Tape Model is similar to the answerer mode of Dobrev et al. [2009], except that the algorithm must specify how many bits it wants to receive when asking for advice.

Another model was introduced by Emek et al. [2011]. They define a universe, $\mathcal{U}$, of all possible answers, assume that $\lceil \log |\mathcal{U}| \rceil$ advice bits are given to the online algorithm with every request, and define the advice complexity to be $\lceil \log |\mathcal{U}| \rceil$. Phrased in terms of the first models from Dobrev et al. [2009], this corresponds to an advice complexity of $\lceil \log |\mathcal{U}| \rceil n$, where n is the length of the request sequence. Thus, to obtain a good advice complexity, the size of the universe must be minimized, which is equivalent to using as few advice bits per request as possible. We refer to this model as the *Per Request Model*. The optimal solution for Paging discussed in the Introduction falls naturally into this model.

In the Per Request Model, any algorithm employing advice uses at least a linear number of bits, making it impossible to explore a lack of information that can be overcome using a sublinear number of advice bits (which is possible in the previously discussed models). Algorithms with sublinear advice are of significant interest for Bin Packing (see Section 9) and several other online problems.

Earlier than Dobrev et al. [2009], a similar notion of advice complexity was introduced by Fraigniaud et al. [2008] in the setting of graph exploration (see Section 11) rather than for traditional online algorithms. Here, all the advice is given in the beginning, and the algorithm learns the length of the advice.

Unless explicitly stated otherwise, the results in this survey are in the Tape Model.

Further Technical Details. Now we discuss some technical details that, although they can be useful to know, are not essential to get an overview of the models.

If an online algorithm wants to read a number of bits encoding an integer X without a (good) known upper bound, then the number of bits to be read must also be provided as information. The standard technique for this is to use a so-called *self-delimiting* encoding (also known as a *prefix code*), as in Böckenhauer et al. [2011]. For example, one may write $\lceil \log(X+1) \rceil$ in unary (using ones), then a zero as a delimiter, followed by X in binary, using $2\lceil \log(X+1) \rceil + 1$ bits in total (this is similar to Elias gamma coding [Elias 1975]). Slightly more efficient encodings may be obtained by iterating this construction. The next iteration (similar to Elias delta coding) uses $\log X + 2 \log\log X + O(1)$ bits to encode an integer X. However, by Kraft's inequality [Cover and Thomas 2006], there does not exist a self-delimiting encoding of the integers using, for example, $\log X + \log\log X + O(1)$ bits, and so we cannot obtain significantly better encodings.

In the graph exploration model of Fraigniaud et al. [2008], all of the advice is given at the beginning and the algorithm learns its length. An algorithm with advice complexity b in this model can be converted to an algorithm with advice complexity $b + O(\log b)$ in the Tape Model by including a self-delimiting encoding of b. Most lower bounds stated in the Tape Model in the literature are in reality shown in a model similar to that of Fraigniaud et al. [2008], not using that the algorithm does not know the length of the advice. This means that upper bounds often contain a logarithmic lower-order term that is not present in the lower bounds.

A technical detail about the graph exploration model of Fraigniaud et al. [2008] is that the oracle is not required to send exactly b bits of advice but may send fewer. Because there are $b+1$ different lengths for strings of length at most b, their oracle is able to send $\sum_{i=0}^{b} 2^i = 2^{b+1} - 1$ different advice strings using at most b bits of advice. This is in contrast to the case where bits are read at the algorithm's discretion from an unbounded supply, where reading up to b bits cannot give more than 2^b different advice strings.

Some bounds transfer between the Per Request Model and the Tape Model. An upper bound from the Per Request Model of b bits for each of n requests gives an upper bound of bn bits in the Tape Model (assuming that b is known to the algorithm, otherwise $bn + O(\log b)$ bits may be required). Similarly, a lower bound stating that b bits are necessary in the Tape Model implies that at least $\lceil \frac{b}{n} \rceil$ bits per request are required in the Per Request Model.

Allowing the algorithm random access to the tape in the Tape Model (as opposed to sequential access) does not make a difference: Since the oracle knows both the algorithm and the input when preparing the advice tape, it can predict which bits the algorithm would access in a random access model and simply place them first sequentially on the advice tape.

Comparison to other Computational Models. The traditional approach of providing an online algorithm with a specific type of knowledge is discussed in detail in Section 3 on semi-online algorithms.

Hromkovič et al. [2010] motivated advice complexity in the context of understanding the information content of a problem, as a function of the number of requests in the instances. For example, in considering the paging problem, or any online problem where the algorithm only makes decisions as to which requests to accept or reject, one bit of advice per request is sufficient to obtain 1-competitiveness. A 1-competitive algorithm does not need more information than this. One bit per request is presumably a smaller number of bits than an upper bound on the Kolmogorov complexity of instances with the same number of requests would be, although an upper bound on the Kolmogorov complexity of the instances of a fixed length always gives an upper bound on the advice complexity for achieving 1-competitiveness. In Hromkovič et al. [2010], they also proposed parameterizing the Tape Model with an upper bound on the running time of the algorithm to obtain an analog of resource-bounded Kolmogorov complexity. These ideas for parameterizing do not appear to have been investigated much yet.

As mentioned in Dobrev et al. [2009] and Emek et al. [2011], the advice complexity model for online problems is similar to an earlier advice complexity model for distributed computing [Fraigniaud et al. 2010]. There, the question was how much advice the nodes in a network need in order to complete some task using as little communication as possible (such as broadcasting, leader election, or coloring the nodes of the network).

Note that what is traditionally called a Turing machine with advice (see Arora and Barak [2009], for example) does not correspond well to an online algorithm with advice. A Turing machine with advice receives advice that may only depend on the *length* of the input, not the input itself.

3. RELATIONSHIP TO SEMI-ONLINE ALGORITHMS

A major motivation for considering advice complexity is the relationship it has to semi-online algorithms. In the literature, the term "semi-online" is used for many different types of problems. For example, a semi-online algorithm may have a look-ahead, that is, the ability to see some of the future requests; the algorithm may be allowed to postpone some decisions or modify some of them after arrival of more input; or the algorithm may be allowed to make assumptions about the request sequence, such as non-increasing sizes. These types of semi-online problems have little known relation to advice complexity. Those that do are the type that either assume some advance knowledge about the input or maintain more than one solution and choose the best solution at the end.

3.1. Assuming Advance Knowledge

Having advance knowledge available to a semi-online algorithm corresponds to advice from an oracle in the advice complexity setting. Thus, depending on the type of advice an oracle provides, an online algorithm with advice can be seen as a semi-online algorithm. Uniform Knapsack, mentioned in the Introduction, is a good example of where the advice model can lead to potentially practical semi-online algorithms; it is only necessary to know if there exists an item of size at least $\frac{1}{2}$. Similarly, there is an online algorithm with advice for BIN PACKING, where only knowledge of the number of items with sizes in $(\frac{1}{2}, \frac{2}{3}]$ is necessary (see Section 9).

Lower bounds on advice complexity, on the other hand, can give proofs that no good semi-online algorithm (of a certain type) exists. For example, a linear (or even super-logarithmic) lower bound on the advice necessary to obtain a competitive ratio of c shows that knowing the number of requests, which would only require a logarithmic number of bits of advice, cannot be sufficient to obtain a competitive ratio of c. At the same time, it would also rule out many other semi-online algorithmic possibilities.

We present some examples to show the interest in semi-online algorithms assuming advance knowledge and show some of the types of advance knowledge that have been considered. Most of the work of this type focuses on scheduling problems (see Section 9 for definitions of scheduling and the makespan objective), and much of it has been for cases where the number of machines is a small constant. In the examples we give, the number of machines, m, is unbounded.

For SCHEDULING on identical machines for makespan, Fleischer and Wahl [2000] present an upper bound of 1.9201 on the competitive ratio of deterministic algorithms, and Rudin [2001] reports a lower bound of 1.88. However, if a semi-online algorithm knows the total sum of processing times, algorithms can do better. A lower bound of 1.585 is proven in Albers and Hellwig [2012], and this lower bound is met by the algorithm in Kellerer et al. [2015] when the number of machines tends to infinity. On the other hand, knowing the value of the optimal makespan, the problem becomes identical to BIN STRETCHING. This problem was introduced in Azar and Regev [2001], and the currently best lower bound, $1.\overline{3}$, was proven there. A 1.5-competitive algorithm for BIN STRETCHING was presented in Böhm [2016].

For SCHEDULING preemptively on uniformly related machines for makespan, if the value of the optimal makespan is given in advance, then an optimal schedule is possible [Ebenlendr and Sgall 2009]. If only an approximation to the optimal value is known, then, even for identical machines, the competitive ratio is increasing with respect to both m and the uncertainty [Jiang and He 2007]. Note that in terms of advice complexity, more uncertainty would generally imply less advice.

Seiden et al. [2000] present a best possible online algorithm for SCHEDULING preemptively on identical machines for makespan, assuming decreasing job sizes (a competitive ratio of about 1.36603), and remark that the assumption of decreasing job sizes can be replaced with knowledge of the size of the largest job.

As an example where quite a bit of advance knowledge (or advice) is used, fitting well into the Per Request Model, SCHEDULING parallel batches with known arrival time of the first job among those remaining with the longest processing times for makespan is considered in Yuan et al. [2011].

For MACHINE COVERING (maximizing the minimum load), it was shown in Woeginger [1997] that the List Scheduling algorithm [Graham 1966] is m-competitive; it is well known that this is best possible (see Azar and Epstein [1997]). The ratio goes down to $m-1$ ($m \geq 3$) if either the total sum of processing times or the longest processing time is known [Tan and Wu 2007]. Even if not all machines become available at the same time, the ratio goes down to $m-2$ ($m > 3$) if both of these are known [Huang and Wu 2010]. If the optimal value is known, then the ratio is only $2 - \frac{1}{m}$ [Azar and Epstein 1997].

3.2. Parallel Solutions

This model was considered in Albers and Hellwig [2014] for SCHEDULING on identical machines for makespan. The model is that the online algorithm is allowed to maintain multiple (partial) solutions. This is similar to running several online algorithms at the same time, but in this model, the algorithm gets to choose the best solution when the input sequence ends. Since this is applied to a scheduling problem, the term *parallel schedules* is used. The number of parallel schedules an online algorithm is allowed to maintain during the processing is a parameter of the problem. Example choices include limiting the number to a constant or to a polynomial.

For example, one of their results is a $(\frac{4}{3} + \varepsilon)$-competitive algorithm using $(\frac{1}{\varepsilon})^{O(\log \frac{1}{\varepsilon})}$ parallel schedules. A corresponding $(\frac{4}{3} + \varepsilon)$-competitive algorithm with advice would receive the index of the best of the $(\frac{1}{\varepsilon})^{O(\log \frac{1}{\varepsilon})}$ parallel schedules from an oracle using $O(\log^2 \frac{1}{\varepsilon})$ bits of advice and perform the same computations as the algorithm with parallel schedules, but only using the schedule indexed by the advice. Similarly, any algorithm with $b(n)$ bits of advice to achieve competitive ratio c can be converted into $2^{b(n)}$ algorithms, each giving a schedule, and choosing the best schedule will give a c-competitive result. Thus, advice complexity can conveniently be used to give lower bounds for parallel solutions approaches.

Maintaining parallel solutions was also considered for the independent set problem in Halldórsson et al. [2002] in a slightly different model. Their upper and lower bound results were asymptotically tight for this model. However, using advice complexity techniques, asymmetric string guessing, and the AOC-completeness (see Section 7) of the problem, both the upper and lower bounds were improved in Boyar et al. [2015a], determining the exact constant for the high-order term in the number of parallel solutions.

4. ADVICE VS. RANDOMIZATION

Before covering algorithmic techniques for advice more broadly, we discuss the strong connection to randomization as further motivation for studying advice complexity.

Derandomization using Advice. It is trivial to see that if an online algorithm uses b random bits, then we can exhibit a deterministic algorithm, which uses b advice bits and is at least as good: The oracle chooses the random bits giving the best performance. However, it seems reasonable to ask for derandomization results not depending on the number of random bits used by the algorithm. Using derandomization techniques, Böckenhauer et al. [2011] obtained the following result: Let $I(n)$ denote the number of inputs of length n to some minimization online problem (later extended to maximization problems [Böckenhauer et al. 2014a; Mikkelsen 2016; Dürr et al. 2016]). If there exists a randomized c-competitive algorithm without advice, then for every constant $\varepsilon > 0$, there exists a deterministic $(c + \varepsilon)$-competitive algorithm with advice complexity $O(\log n + \log\log I(n))$. For a large number of online problems, the number of possible inputs of length n is at most $2^{n^{O(1)}}$. Thus, for these problems, it is possible to convert any randomized algorithm into an (almost) equally good deterministic algorithm with advice complexity $O(\log n)$.

We remark that this result is essentially tight. It is shown in Mikkelsen [2016] that, for any increasing function $I(n)$, there exists (pathological) online problems where $\Omega(\log\log I(n))$ bits of advice are indeed needed for such a conversion. Thus, for online problems with large input spaces, it is possible that a lot of advice is required to simulate randomization. However, so far no one has stumbled on a “natural” online problem (that is, a problem not specifically constructed for this purpose) where more than $O(\log n)$ bits of advice are needed to simulate randomization.

Finally, we note that this derandomization result can, of course, also be used to convert an algorithm that uses both advice and randomization into a deterministic algorithm with advice (see Mikkelsen [2016]). Therefore, randomized algorithms with advice are rarely studied explicitly.

Replacing Advice Bits with Random Bits. Intuitively, it might appear that having access to even a rather small number of advice bits provided by an omniscient oracle knowing the entire input should often be more powerful than simply having access to (any number of) random bits. Perhaps surprisingly, it turns out that for many important online problems, this is not the case.

Let us first consider the naive idea of simply running an algorithm with advice, ALG, with a tape full of random bits (instead of bits provided by an oracle). Call the resulting randomized algorithm RAND. It is easy to construct a pathological minimization problem where a single bit of advice yields an optimal algorithm while no randomized algorithm can achieve any meaningful competitive ratio (consider a problem where one of the first two requests should be chosen over the other, and either can have arbitrarily larger cost than the other). On the other hand, for a maximization problem with non-negative profits, the naive conversion will turn a c-competitive algorithm reading b bits of advice into a $(c \cdot 2^b)$-competitive randomized algorithm. Indeed, for every input I, we have $\text{RAND}(I) = \text{ALG}(I)$ with probability at least $\frac{1}{2^b}$. Since scores cannot be negative, this implies that $\mathbb{E}[\text{RAND}(I)] \geq \frac{\text{ALG}(I)}{2^b}$.

It is possible to do significantly better than the naive conversion for a large class of important online minimization problems. In particular, it is possible to do better for any problem that can be modeled as a METRICAL TASK SYSTEM (see Section 8). Before the introduction of advice models, this was studied as the problem of "combining online algorithms online." Blum and Burch [2000] showed how to use the celebrated machine learning algorithm Randomized Weighted Majority to obtain the following result: For every $\varepsilon > 0$, it is possible to combine m algorithms for a METRICAL TASK SYSTEM, $\text{ALG}_1, \ldots, \text{ALG}_m$, into a single randomized algorithm, RAND, such that for every input I,

$$\mathbb{E}[\text{RAND}(I)] = (1+\varepsilon) \cdot \min_{1 \leq i \leq m} A_i(I) + O(\Delta \log m).$$

Here Δ is the normalized diameter of the underlying metric space. Note that if $m \in O(1)$, then $O(\Delta \log m)$ is just an additive constant. Thus, using our terminology, Blum and Burch show the almost surprisingly strong connection that for any METRICAL TASK SYSTEM, a c-competitive algorithm with advice complexity $O(1)$ can be converted into a $(c+\varepsilon)$-competitive randomized algorithm without advice. The result of Blum and Burch was later extended in Mikkelsen [2016] by showing that such a conversion is also possible if the algorithm uses $o(n)$ bits of advice instead of constant advice. Together with the derandomization result, this gives a striking equivalence between advice and randomization for many online problems, including those mentioned in the following theorem:

THEOREM 4.1 (MIKKELSEN [2016]). *Let* P *be* METRICAL TASK SYSTEM*,* k-SERVER*,* LIST UPDATE*,* PAGING*, or* DYNAMIC BINARY SEARCH TREE *and assume that the underlying metric space/node set is finite. Let c be a constant independent of the input length n. The following are equivalent:*

—*For every $\varepsilon > 0$, there exists a $(c+\varepsilon)$-competitive* P *algorithm with advice complexity $o(n)$.*
—*For every $\varepsilon > 0$, there exists a $(c+\varepsilon)$-competitive randomized* P *algorithm without advice.*

Note that for k-SERVER, for example, determining the best possible competitive ratio of a randomized algorithm is a long-standing open problem. In particular, the randomized k-SERVER conjecture states that for every metric space, there exists an $O(\log k)$-competitive randomized algorithm [Koutsoupias 2009]. It was noted in Böckenhauer et al. [2011] that, due to the derandomization result, a sufficiently large advice complexity lower bound would disprove this conjecture. Theorem 4.1 shows that the randomized k-Server conjecture is in fact equivalent to the conjecture that there exists an $O(\log k)$-competitive deterministic algorithm with advice complexity $o(n)$ (assuming the underlying metric space is finite). See Section 8 for more information on k-Server.

5. ALGORITHMIC TECHNIQUES

We discuss general techniques for designing algorithms with advice.

Derandomization using Advice. It is often possible to convert a randomized online algorithm into a deterministic online algorithm reading $O(\log n)$ bits of advice. Section 4 was devoted to the treatment of the relationship between advice and randomization.

Adapting Offline Algorithms. It is sometimes possible to convert an existing (exact or approximation) offline algorithm into an online algorithm using a relatively small number of advice bits. This has been done for Bin Packing and Scheduling [Renault et al. 2015] and Multi-Coloring [Christ et al. 2015] (see Sections 9 and 10.2). It can also be possible to convert streaming algorithms, for example, into online algorithms with advice, as has been done for bipartite matching [Dürr et al. 2016].

The Now-or-Later Technique. The now-or-later technique is based on giving one bit of advice per request. The technique has been used for Paging as described in Example 1 in the Introduction: Each time a page is requested, one bit of advice is given, indicating whether the requested page can safely be evicted the next time a page fault occurs or if the algorithm should keep the page in cache until it has been requested at least once more.

Reordering Buffer Management is similar to paging: A buffer of a certain size is given, and the input is a sequence of items. For each request, if the buffer is full, then an item must be removed from the buffer. Each item has a color, and if the evicted item has a color that differs from the previously evicted item, then a cost of 1 is incurred. A slightly more complicated version of the now-or-later technique (using two advice bits per request to also include a "soon, but not now"-option) was applied to Reordering Buffer Management in Adamaszek et al. [2016] (see also [Renault 2014]), resulting in a $\frac{3}{2}$-competitive algorithm, which was extended to a $(1+\varepsilon)$-competitive algorithm using $O(\log \frac{1}{\varepsilon})$ bits per request.

The Follow-OPT Technique. This technique was introduced in Emek et al. [2011] and has been used for Metrical Task System and k-Server [Emek et al. 2011; Böckenhauer et al. 2011; Renault and Rosén 2015]. In these problems, there is a bounded number of possible states. With a lot of advice, it is possible to specify exactly which state the algorithm should be in after each request. With fewer bits, the idea is to specify the exact state as often as possible, ensuring that the state of the algorithm often coincides with the state of Opt. When serving those requests for which the precise state of Opt is not specified, the algorithm tries to be conservative and not make risky decisions.

Combinatorial Designs. In many cases, the amount of advice needed to achieve a given competitive ratio is closely related to the minimum size of certain combinatorial structures. The idea is to "compress" the optimal set S of advice strings into a smaller set S'. The strings in S' have the same length as those in S, and each string in S is "close to" some string S', that is, each string in S' can be thought of as representing a subset of S. The advice given is an index to a string in the smaller set S'. If the aim is simply to minimize the Hamming distance between each string in S and its representative in S', then covering codes can be used. However, in many cases, it must be ensured that all ones (or all zeros) in the string in S be present in its representative in S'. In this case, covering designs can be used. For example, upper bounds on the size of covering designs have been used to obtain algorithms with advice for PAGING (see Section 8) and MINASG (see Section 7.2). Similarly, upper bounds on the size of covering codes have been used to construct algorithms with advice for problems that include STRING GUESSING (see Section 7.1) and MATCHING on paths and trees [Keller 2014].

Note that since we generally do not restrict the running time of our online algorithms, the upper bounds on the size of the given combinatorial structure need not be constructive. This is important for the applications involving covering designs, for example, where good upper bounds proven via the probabilistic method exist, but where it is not known how to construct such covering designs efficiently (see Boyar et al. [2015a] for details).

The Warning Signal Technique. An obvious technique for designing algorithms with advice is to consider an online algorithm ALG without advice and try to use advice to pinpoint exactly when ALG makes mistakes. The idea is that simply warning the algorithm of mistakes that it is about to make might be much cheaper than telling the algorithm exactly what to do. This has been done for edge coloring of trees (see Section 10.2).

Exponential Sparsification. For weighted problems where a good advice algorithm exists for the case where there are only few different weights, exponential sparsification can sometimes be used. The requests are grouped based on their weights into intervals $((1+\varepsilon)^k, (1+\varepsilon)^{k+1}]$ for $k = -\infty \ldots \infty$.

The first idea is to treat requests with weights in the same interval $((1+\varepsilon)^k, (1+\varepsilon)^{k+1}]$ as having weight $(1+\varepsilon)^{k+1}$. For some problems, this gives only a small loss in competitive ratio for the algorithm. This idea was used in Renault et al. [2015] for online scheduling. It has also been used for different variants of approximation problems (no advice involved), such as developing a polynomial-time approximation scheme for minimizing makespan in scheduling; see pages 80–84 of Vazirani [2003], for example.

The second idea is that requests with weights in an interval $((1+\varepsilon)^k, (1+\varepsilon)^{k+1}]$, for sufficiently small (or large) k (compared to that of the other requests), may be served in some simple way without using any advice with only a small loss in competitive ratio. For example, for WEIGHTED INDEPENDENT SET, a policy could be to always reject vertices with a weight below some threshold. Depending on this threshold, this might only give a small loss in competitive ratio. Note that, in the beginning, some scheme should be used to identify which requests have (relatively) small weights. This could, for example, involve using $O(\log n)$ bits to give the index of the first request that does not have a small weight.

Combining the two ideas, we now just need an algorithm (for the remaining requests) that solves the problem well when only few different weights are allowed. This approach was used in Boyar et al. [2016b] for WEIGHTED VERTEX COVER, WEIGHTED INDEPENDENT SET, and several other problems.

6. LOWER BOUND TECHNIQUES

We discuss general techniques for establishing lower bounds against algorithms with advice.

The Pigeonhole Technique. Construct a set of inputs, $\mathcal{I}$, where $|\mathcal{I}| = m$. Suppose that an algorithm reads at most b bits of advice on any input from $\mathcal{I}$. By the pigeonhole principle, this algorithm must read the same advice for at least $\lceil \frac{m}{2^b} \rceil$ of the inputs in $\mathcal{I}$. Thus, it suffices to show that for any subset, $\mathcal{I}' \subset \mathcal{I}$, of size at least $\lceil \frac{m}{2^b} \rceil$ and any fixed deterministic algorithm (without advice), there is an input from $\mathcal{I}'$ on which the algorithm performs poorly. In many cases, this is achieved by designing $\mathcal{I}$ such that all inputs have some common prefix. On this common prefix, a deterministic algorithm selected for $\mathcal{I}'$, based on the advice, will always produce the same output. So, if different inputs in $\mathcal{I}'$ require different outputs for the common prefix, this yields a lower bound on the advice required. More generally, one may use a partition tree [Barhum et al. 2014], where nodes in the tree represent sets of inputs with a common prefix. The pigeonhole technique is applied in Böckenhauer et al. [2014c], Böckenhauer et al. [2011], Komm et al. [2015], Boyar et al. [2016c], Bianchi et al. [2014b], and Renault et al. [2015], for example.

The Multiple Algorithms Technique. Any algorithm ALG reading b bits of advice can be converted into 2^b algorithms, $\text{ALG}_1, \ldots, \text{ALG}_{2^b}$, without advice such that for every input I, $\text{ALG}(I) = \min_{1 \le i \le 2^b} \text{ALG}_i(I)$ for minimization problems (for maximization problems, min is replaced by max). Thus, we can get a lower bound by showing how an adversary can construct an input such that all of the 2^b algorithms perform poorly on that input. One can, for example, create an input in rounds, where each round ensures that some fraction of the algorithms perform poorly. This technique is applied in Mikkelsen [2015], Boyar et al. [2015a], Komm and Královič [2011], Komm et al. [2012], and Clemente et al. [2016], for example.

The Probabilistic Method. Suppose that we are able to construct a probability distribution over a set of inputs $\mathcal{I}$ and show that for any deterministic algorithm without advice, the probability that the algorithm performs "well" is very small. Then this gives an advice complexity lower bound. For example, let ALG be an algorithm reading b bits of advice. Then ALG can be converted into 2^b deterministic algorithms, $\text{ALG}_1, \ldots, \text{ALG}_{2^b}$, without advice (as done in the multiple algorithms technique). Assume that for every deterministic algorithm, DET, without advice, it holds that $\Pr[\text{DET}(I) \le c \cdot \text{OPT}(I)] < \delta$, where I is drawn from $\mathcal{I}$ according to our input distribution. Then, by the union bound, this implies that $\Pr[\text{ALG}(I) \le c \cdot \text{OPT}(I)] = \Pr[\min_{1 \le i \le 2^b}\{\text{ALG}_i(I)\} \le c \cdot \text{OPT}(I)] \le 2^b\delta$. If $2^b\delta < 1$, then this implies that there exists an input $I \in \mathcal{I}$ such that $\text{ALG}(I) > c \cdot \text{OPT}(I)$, and hence ALG is not strictly c-competitive. The probabilistic method is applied in Barhum [2014], Gebauer et al. [2015], and Mikkelsen [2016], for example. See also Section 7.1 for a simple but useful lower bound obtained via this technique.

Advice-Preserving Reduction. Suppose that we already have a lower bound on the advice complexity for a problem P. An easy way to obtain a lower bound on the advice complexity for a related problem P′ is to reduce P to P′ in a suitable way. A number of abstract guessing games have been introduced specifically with the purpose of serving as the starting point of such reductions (see Section 7).

***Σ**-Repeatable Online Problems*. It was shown in Mikkelsen [2016] that for online problems that are "repeatable," it is often possible to translate lower bounds for randomized algorithms without advice into lower bounds for algorithms with sublinear advice. Informally, an online problem is Σ-*repeatable* if it is always possible to combine r (sufficiently costly) inputs $I_1, \ldots, I_r$ into a single input $I = f(I_1, \ldots, I_r)$ such that serving I essentially amounts to serving each I_i independently and adding the costs incurred. In particular, the way an algorithm serves $I_1, \ldots, I_{i-1}$ should not significantly affect how efficiently the algorithm can serve I_i. Paging is Σ-repeatable, since one may simply concatenate the inputs $I_1, \ldots, I_r$. The only dependency between the number of page faults of two different rounds is that our initial cache when serving the requests of I_i corresponds to our final cache when serving the requests of I_{i-1}. However, if we make sure that $\text{OPT}(I_i)$ is much larger than the cache size, then this small dependency can essentially be ignored when proving lower bounds. A problem that is *not* Σ-repeatable is BIN PACKING. Consider inputs $I_1 = (\frac{1}{2} - \varepsilon, \ldots, \frac{1}{2} - \varepsilon)$ and $I_2 = (\frac{1}{2} + \varepsilon, \ldots, \frac{1}{2} + \varepsilon)$, both of length n. While concatenating I_1 and I_2 does give a valid BIN PACKING input I, if we pack the items of I_1 two per bin, then we have to open n new bins for serving the items of I_2. On the other hand, if we pack each item of I_1 in a separate bin, then we may pack the items of I_2 without opening any new bins at all. Thus, the choice of how to serve the items of I_1 has a significant influence on the number of bins needed to serve the items of I_2. Of course, one might try to construct $I = f(I_1, I_2)$ in a more clever way than just concatenating I_1 and I_2, but it can be shown that no choice of f will work for Bin Packing.

For a Σ-repeatable problem, we have the following result [Mikkelsen 2016] (omitting some minor technical conditions): Let P be a Σ-repeatable online problem, where, for each n, the number of inputs of length n is finite. Suppose that a randomized algorithm without advice cannot be better than c-competitive, where c does not depend on the input length n. Furthermore, suppose that this lower bound holds even if the adversary has to reveal an upper bound on the length of the input in advance. Then a (possibly randomized) algorithm reading $o(n)$ bits of advice must have competitive ratio at least c. An informal, intuitive explanation is the following: By assumption, there is a hard input distribution against which randomized algorithms without advice are no better than c-competitive. Since the problem is Σ-repeatable, one can repeatedly draw instances from this distribution and combine these to obtain an input consisting of r independent rounds. If an algorithm reads only $o(n) = o(r)$ bits of advice, then it will have close to no advice for most of the r rounds and, hence, will perform as poorly as an algorithm without advice in most rounds.

The currently best known lower bounds (for algorithms with sublinear advice) for Paging, k-Server, List Update, Max-SAT, Unit Clustering, Bipartite Matching, and several other problems have been achieved by combining the result above with the currently best known lower bounds for randomized algorithms without advice [Mikkelsen 2016].

$\vee$-repeatable Online Problems. For a Σ-repeatable problem, the total cost has to be essentially the sum of costs incurred in each individual round. It is also possible to consider another collection of repeatable problems, where the total cost is the maximum cost incurred in a single round. We call such problems *$\vee$-repeatable*. For those problems, we have the following lower bound result [Mikkelsen 2016]: Let P be an $\vee$-repeatable online problem. Suppose that a deterministic algorithm without advice cannot be better than c-competitive, where c does not depend on n. Furthermore, assume that the lower bound holds even if the algorithm knows Opt(I) in advance and knows an upper bound on the number of requests. Then no (possibly randomized) algorithm reading $o(n)$ bits of advice can be better than c-competitive. This result is similar to the result for Σ-repeatable problems, but note that, for $\vee$-repeatable problems, we only need a lower bound for *deterministic* algorithms without advice in order to apply the technique. On the other hand, for $\vee$-repeatable problems, we have an additional assumption regarding the cost of an optimal solution.

The assumption regarding Opt(I) turns out to be crucial. Intuitively, if the lower bound for algorithms without advice can only be proven by a construction where Opt(I) varies depending on how the deterministic algorithm works, then repeating the construction using r rounds will not necessarily work. For example, it may happen that Opt incurs a large cost, T, in a single round, and a much lower cost, t, in the other $r-1$ rounds. In that case, the algorithm may only need to perform well in the one round where Opt incurred a cost of T. Indeed, even if the algorithm incurs a cost of ct in the other $r-1$ rounds, then if ct is smaller than T, this will not affect the total cost of the algorithm, since the cost is the maximum (and not the sum) over all of the rounds. See the Multi-Coloring results in Section 10 for a concrete example of this phenomenon.

The main examples of $\vee$-repeatable problems are graph coloring problems.

Direct Product Theorems. Direct product theorems were introduced as a way to prove lower bounds in Mikkelsen [2016]. Intuitively, a direct product theorem says that if b bits of advice are needed for an online algorithm to ensure a cost of at most t when faced with requests drawn from a probability distribution p, then $r \cdot b$ bits of advice are needed to ensure a total cost of at most $r \cdot t$ when r independent rounds of requests are drawn from p.

The result for Σ-repeatable online problems discussed earlier is proven by having the requests of each round be an entire input itself (drawn from some hard input distribution) and then applying a direct product theorem. However, it is also sometimes possible to have each round be only a single request of the input. Obviously, this approach will usually require more effort, since one no longer treats the hard input distribution just as a black box (as was the case with the result for Σ-repeatable problems). On the other hand, this approach can lead to significantly stronger lower bounds than what can be achieved by only using the general result for Σ-repeatable problems. For example, a super-linear lower bound for Vertex Coloring has been proven using this approach (see Section 10.1).

7. STRING GUESSING AND COMPLEXITY CLASSES

String Guessing is a rather artificial problem that is used primarily to show linear lower bounds on the advice complexity of certain problems, so most of the problems considered here are hard from an advice complexity point of view, that is, much advice is needed to obtain a good competitive ratio. Some of the problems are even hard offline.

There are several types of string guessing problems. We start with the simplest version.

7.1. String Guessing

String Guessing was introduced in Böckenhauer et al. [2014b], and it is essentially the same as Generalized Matching Pennies, defined and studied earlier by Emek et al. [2011]. Both of these problems consider strings of length n over an alphabet of size q. The goal is to guess as many of the characters of the input string as possible correctly. There are two versions of the problem: String Guessing with *known history*, where the correct answer to the previous request is revealed with each new request, and String Guessing with *unknown history*, where the correct answers are revealed only at the end of the input.

Note that an algorithm that answers uniformly at random each time will guess $\frac{n}{q}$ characters correctly in expectation. Clearly, one can achieve the same guarantee with a deterministic algorithm, reading $\lceil \log q \rceil$ bits of advice (identifying the most frequent character in the input string). The following theorem gives a lower bound on the advice needed to guess more than a fraction of $\frac{1}{q}$ of the input characters correctly.

Theorem 7.1 (Böckenhauer et al. [2014b]). *Any online algorithm with advice for* String Guessing *with known history (over an alphabet of size q), guaranteeing guessing γn characters of the input correctly, for some constant $\frac{1}{q} < \gamma < 1$, must read at least*

$$\left(1 + (1-\gamma)\log_q\left(\frac{1-\gamma}{q-1}\right) + \gamma \log_q \gamma\right) n \log q \in \Omega(n \log q)$$

advice bits.

The lower bound (Theorem 7.1) can equivalently be written as $(1 - H_q(1-\gamma))(\log q)n$, where H_q is the q-ary entropy function [Böckenhauer et al. 2014b]. Also, it may be useful to know that Theorem 7.1 is closely related to the Chernoff bound [Hoeffding 1963]. Indeed, it can be proven using the probabilistic method (see Section 6) as follows:

Choose the input string uniformly at random. Let DET be a fixed deterministic algorithm without advice. Each time, the probability that DET guesses the correct character is exactly $\frac{1}{q}$, and this probability is independent of all other guesses. Thus, the number of characters guessed correctly by DET is a sum of independent identically distributed Bernoulli random variables with expected value $\frac{1}{q}$. By the Chernoff bound, the probability that DET guesses γn (or more) characters correctly is at most $2^{-(1-H_q(1-\gamma))(\log q)n}$. It follows that an algorithm with advice needs at least $b \geq (1 - H_q(1 - \gamma))(\log q)n$ bits of advice to ensure that it always correctly guesses at least γn characters [Mikkelsen 2016]. This is exactly the lower bound of Theorem 7.1.

Via reductions, STRING GUESSING with known history has been used to prove many advice complexity lower bounds, including some in Böckenhauer et al. [2014b], Emek et al. [2011], Gupta et al. [2013], Angelopoulos et al. [2015], Böckenhauer et al. [2015], Boyar et al. [2016c], Boyar et al. [2014], Adamaszek et al. [2016], Komm et al. [2016], Dürr et al. [2016], and Bianchi et al. [2016].

For STRING GUESSING with unknown history, Böckenhauer et al. [2014b] give (using known bounds on the size of covering codes) an upper bound that matches the lower bound for STRING GUESSING with known history up to an additive $O(\log n)$ term. Note that the lower bound is for the easier of the two problems, and the upper bound is for the harder version. Thus, both bounds are as general as possible.

Other String Guessing Problems. Other string guessing variants were analyzed in Mikkelsen [2016]: ANTI-STRING GUESSING yields better lower bounds for PAGING with advice and for INDUCED SUBGRAPH [Komm et al. 2016], and WEIGHTED BINARY STRING GUESSING yields a better lower bound for BIN PACKING. A special case of STRING GUESSING with known history, where the algorithm knows the number of zeros in the string in advance, was used for advice complexity lower bounds for bin packing [Boyar et al. 2016c; Angelopoulos et al. 2015].

7.2. Asymmetric String Guessing

Consider accept/reject minimization problems, that is, minimization problems where the irrevocable decision for each request is either to accept or reject it. Assume that the problem is such that a superset of a feasible solution is always feasible. An example one could keep in mind is VERTEX COVER. This is the standard vertex cover problem in the *vertex arrival model*, so the vertices arrive online, and each vertex arrives with a list of all previous vertices to which that vertex is adjacent. The accepted vertices must form a vertex cover, so at least one endpoint of each edge must be chosen. The fact that edges to vertices that have not been seen yet are unknown when a vertex arrives means that the well-known 2-approximation algorithm, accepting both endpoints of some edges, cannot be used.

The obvious advice to give is a string of bits, one for each request, with ones indicating acceptance and zeros indicating rejection for an optimal solution. One can also use this idea in a c-competitive algorithm with advice. Suppose that for each request sequence length n and each $t \leq \lceil \frac{n}{c} \rceil$, the algorithm can compute a set of binary strings, $S_{n,t,c}$, such that for every request sequence of length n with a minimum solution of size t, there is a string in $S_{n,t,c}$ that indicates a superset of a minimum solution, where the superset must have size at most ct. Then the oracle can give the algorithm n, t, and an appropriate index into $S_{n,t,c}$. The algorithm can be c-competitive by using the indexed string and answering "accept" or "reject" based on that string, ignoring the actual request sequence. If $t > \lceil \frac{n}{c} \rceil$, then it is safe to answer "accept" for every request. Note that the value n must be given in a self-delimiting encoding, and the total length of the advice is $\lceil \log |S_{n,t,c}| \rceil + O(\log n)$. One can think of the above algorithm as trying to guess a string corresponding to a minimum solution but being allowed to make a few errors in the direction of guessing ones for some zeros in that optimal solution.

Realizing that many problems exhibit the same characteristics as VERTEX COVER, Boyar et al. [2015a] study an abstraction of this problem in the form of Minimum Asymmetric String Guessing, MINASG. As with other string guessing problems, MINASG does not appear interesting in its own right, but the above example shows its relation to other problems. In MINASG, the request sequence is a sequence of bits that the algorithm must try to guess (for example, indicating a minimum solution to an instance of VERTEX COVER). The cost is the number of ones guessed, unless the algorithm at some point guesses zero, when the correct bit was a one. In the latter case, the cost is infinite (this corresponds to a (possibly) infeasible answer in the VERTEX COVER case). The goal is, of course, to minimize cost.

As with STRING GUESSING, there are two variants of MINASG, known history and unknown history. There is also a maximization version of Asymmetric String Guessing, MAXASG. For that version, INDEPENDENT SET could be the problem to keep in mind. The objective is to guess as many zeros correctly as possible, and guessing a zero where the correct answer is a one gives a profit of $-\infty$. Again, there is a version with known history and one with unknown history.

Using results on covering designs, tight bounds are proven in Boyar et al. [2015a] on all four versions of Asymmetric String Guessing, showing that the number of advice bits necessary and sufficient to achieve competitive ratio c is

$$b = \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n \pm \Theta(\log n), \tag{1}$$

where

$$\frac{1}{e \ln 2}\frac{n}{c} \leq \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n \leq \frac{n}{c}\,;$$

here $\pm\Theta(\log n)$ means that there exist constants c_l and c_u, where $c_l \leq 0 \leq c_u$, so, for sufficiently large n, $c_l \log n$ and $c_u \log n$ are the lower and upper bounds on the additive term.

Returning to the motivating VERTEX COVER example, the closed formula (1) bounds the term $\lceil \log |S_{n,t,c}| \rceil + O(\log n)$ from that example. VERTEX COVER is not exactly the same problem as either MINASG with known or unknown history, since it may be possible to deduce some but not all information about past mistakes during the processing of vertices. However, MINASG with known history can be used to provide lower bounds, whereas MINASG with unknown history can be used for upper bounds.

7.3. Complexity Classes

Problems such as MINASG and VERTEX COVER led to the definition of the first complexity class for online algorithms, Asymmetric Online Covering (AOC) [Boyar et al. 2015a], which contains many accept/reject problems, both minimization and maximization (the maximization problems are generally packing problems, despite the name AOC). The minimization problems have the property that any superset of a feasible solution is a feasible solution, and the maximization problems have the property that any subset of a feasible solution is a feasible solution. In both cases, the cost/profit of a feasible solution is the size of the accepted set, and the cost (profit) of an infeasible solution is ∞ ($-\infty$). Maximization versions of MINASG have the same advice complexity as MINASG. This is used to show an upper bound on the advice complexity of all problems in AOC.

The hardest problems in AOC, those that require

$$\log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n \pm \Theta(\log n)$$

bits of advice to be c-competitive, are called AOC-complete [Boyar et al. 2015a]. Using reductions from the asymmetric string guessing problems, Vertex Cover, Independent Set, Dominating Set, Disjoint Path Allocation, Set Cover, and Cycle Finding are shown to be AOC-complete. All but the last of these correspond to offline problems that are NP-hard. Note that although these problems are proven to be complete via reductions, there are no unproven assumptions, such as $P \neq NP$. Tight bounds on the advice complexity of these problems are known. The AOC-complete problems are all hard online problems: Without advice, these problems have competitive ratios that are $\Omega(\frac{n}{\log n})$, and, in fact, all the known AOC-complete problems [Boyar et al. 2015a] have $\Omega(n)$ competitive ratios (actually, n or $n-1$ for all but one problem). Examples of problems that are in AOC, but not AOC-complete, are Uniform Knapsack and Matching.

Corresponding to each AOC problem is a weighted version of the problem, which is still an accept/reject problem, but the cost/profit of each request may vary due to a weight associated with the request. For example, for Weighted Independent Set, the vertex arrival model is used, but each vertex arrives with a weight, in addition to a list of all previous vertices adjacent to it. The goal is to accept a maximum-weight independent set. In contrast to the unweighted case, when weights are added to AOC-complete problems, the maximization and minimization problems have different advice complexities. In Boyar et al. [2016b], it was shown that the weighted versions of the complete maximization problems have advice complexity at most an additive term $O(\log^2 n)$ worse than the unweighted versions, but the weighted versions of the known complete minimization problems all have unbounded competitive ratios with fewer than $n - O(\log n)$ bits of advice. This latter result is proven using length-preserving advice reductions; all known AOC-complete minimization problems were proven complete for AOC using this type of reduction. Thus, the class containing the weighted versions of these complete minimization problems is harder with respect to advice complexity than the class containing the weighted versions of the complete maximization problems.

The maximization (and not the minimization) problems in AOC are examples of problems where the greedy algorithm is best possible according to online bounded analysis, which is defined in Boyar et al. [2016a].

In Komm et al. [2016], graphs with certain properties are studied. A graph property, Π, is a set of graphs. It is said to be

—hereditary if, for every graph G in Π, all induced subgraphs of G are also in Π.
—cohereditary if, for every graph G in Π, all graphs containing G as an induced subgraph are also in Π.
—non-trivial if there are an infinite number of graphs in Π and an infinite number of graphs not in Π.

Examples of non-trivial hereditary graph properties include independent sets, forests, and planar graphs. Examples of non-trivial cohereditary graph properties include graphs containing a cycle and non-planar graphs. Let a non-trivial hereditary graph property Π be given. A graph is presented in the vertex arrival model and the goal is for the algorithm to accept as many vertices as possible, such that the induced subgraph defined by the accepted vertices is in Π. They show that at least

$$\log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n - \Theta(\log^2 n)$$

bits of advice are required to be c-competitive for these problems (independent of the choice of Π). These problems are, in some sense, shown to be almost AOC-complete. For a cohereditary graph property Π, the problem considered is the same, except that the goal is to accept as few vertices as possible, such that the induced subgraph defined by the accepted vertices at the end is in Π (it is guaranteed that the graph presented is in Π). They show that for this problem, the advice complexity depends crucially on the choice of graph property. For some properties, the problem is AOC-complete; for others, it is possible for an algorithm to be optimal using only $O(\log n)$ advice bits.

8. *K*-SERVER, PAGING, AND FRIENDS

A Metrical Task System [Borodin et al. 1992] is defined by a tuple $(\mathcal{S}, \mathcal{T}, d)$ where $\mathcal{S}$ is a set of N *states*, $\mathcal{T}$ is a set of *tasks*, and $d : S \times S \to [0, \infty)$ is a metric distance function. A *task* is a mapping $t : \mathcal{S} \to [0, \infty]$ satisfying that there exists at least one state $s \in \mathcal{S}$ such that $t(s) \neq \infty$. An input consists of an initial state $s_0 \in \mathcal{S}$ and n tasks $t_1, \ldots, t_n$. Immediately after a task t_i arrives, the online algorithm must choose a state s_i for serving t_i: The online algorithm moves from its current state s_{i-1} to the state s_i at a cost of $d(s_{i-1}, s_i)$ and serves the task t_i at a cost of $t_i(s_i)$. The goal is to minimize the total cost incurred. Each of the classic online problems of Paging, k-Server, and List Update can be modeled as a Metrical Task System (see Borodin and El-Yaniv [1998], for example).

For the classic online scenario, a matching upper and lower bound of $2N-1$ is known for the competitive ratio of deterministic Metrical Task System algorithms [Borodin et al. 1992]. For the randomized case, there exists a randomized $O(\log^2 N \log\log N)$-competitive algorithm [Fakcharoenphol et al. 2004], whereas the best known lower bound on the competitive ratio is the $\Omega(\log N)$ lower bound arising from Paging.

The advice complexity of Metrical Task System is well understood. We know that sublinear advice is equivalent to randomization (Theorem 4.1). Furthermore, it was shown in Emek et al. [2011] that b bits of advice per request are both necessary and sufficient to be $\Theta(\frac{\log N}{b})$-competitive. The upper bound is achieved using the follow-OPT technique. The matching lower bound is proven via a reduction from Generalized Matching Pennies (see Section 7.1).

The advice complexity of k-Server (see Figure 2) is not as well understood as for Metrical Task System. Again, we know that randomization is equivalent to sublinear advice. Depending on the size N of the metric space, the currently best known randomized algorithm without advice for k-Server is either the $O(\log^2 k \log^3 N \log\log N)$-competitive algorithm due to Bansal et al. [2015] or simply the deterministic $(2k-1)$-competitive Work Function Algorithm [Koutsoupias and Papadimitriou 1995]. From Mikkelsen [2016], randomized k-Server algorithms (on finite metric spaces) can be simulated using a number of advice bits depending only on k and the metric space. The current best upper bound for algorithms using $b \geq 3$ bits of advice per request is $O(\frac{\log k}{b})$, using the follow-OPT technique [Renault and Rosén 2015; Böckenhauer et al. 2011]. However, no matching lower bound is known. The lower bound used for Metrical Task System does not seem to be applicable to k-Server. For k-Server, we only know that $\Omega(n \log k)$ bits of advice are needed to be optimal [Böckenhauer et al. 2011] and that $\Omega(n)$ bits of advice are needed to be better than H_k-competitive (the last lower bound follows since Paging is a special case of k-Server; see the next paragraph for details). In particular, it is an intriguing open problem whether it is possible to be $(1+\varepsilon)$-competitive using $O(n)$ bits of advice for arbitrarily small ε. It was shown in Böckenhauer et al. [2011] that this is in fact the case if the underlying metric space is the Euclidean plane: Along with every

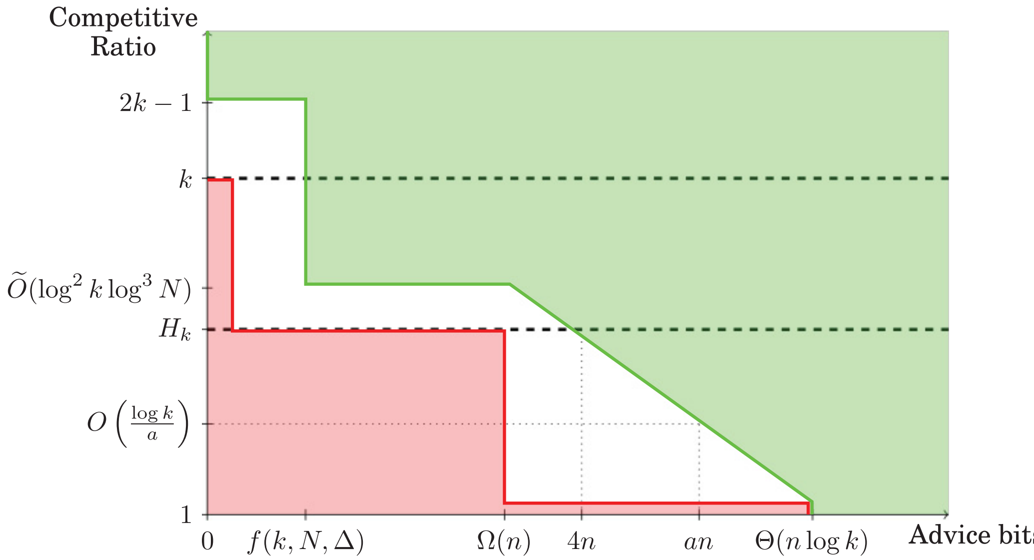

Fig. 2. The asymptotic tradeoff between competitive ratio and advice for k-Server. N is the number of points in the metric space and Δ the (normalized) diameter. The function $f(k, N, \Delta)$ is a rapidly growing function of k, N, and Δ (but does not depend on n). For the randomized algorithm with a competitive ratio depending on N, we assume that N is relatively small; polynomial in k, for example.

request, one may use $O(1)$ bits of advice to indicate as precisely as possible in which "direction" the server used by Opt for serving this request is currently located. Also, for various sparse metric spaces (such as paths, trees, and planar graphs), algorithms that are better than the algorithm for the general case are known [Renault and Rosén 2015; Gupta et al. 2013].

The asymptotic advice complexity of Paging is essentially completely understood (see Figure 1). Recall that the best possible competitive ratio for a randomized Paging algorithm without advice is $H_k \in \Theta(\log k)$. Using b bits of advice, it is possible to be $(\frac{2k+2}{2^b} + 3b)$-competitive, while any algorithm using only b bits of advice must have a competitive ratio of at least $\frac{k}{2^b}$ [Böckenhauer et al. 2009]. In particular, $O(\log k)$ bits of advice suffice to be $O(\log k)$-competitive while $o(\log k)$ bits of advice is not enough to be, for example, $k^{0.99}$-competitive. Furthermore, it is possible to be $(H_k + \varepsilon)$-competitive using a number of advice bits depending only on k and ε (and not the input length n) [Mikkelsen 2016]. In order to achieve a competitive ratio better than H_k, we need $\Omega(n)$ bits of advice (since reading $o(n)$ bits of advice is equivalent to randomization, according to Theorem 4.1). On the other hand, n bits of advice suffice to be optimal using the algorithm described in the Introduction.

The exact tradeoff between advice and the competitive ratio for Paging is still open. For constant competitive ratios, the current best upper bound is (perhaps a bit surprisingly) achieved by using the upper bound for the AOC-complete problem minASG (see Section 7.2 and in particular Equation (1)): Let $x = x_1 \dots x_n$ be a binary string such that x_i is 0 if and only if the page requested in round i will be requested once more before it is removed from the cache of Opt. As already mentioned in the Introduction, a Paging algorithm that is given x as advice can be optimal. It was observed in Böckenhauer et al. [2009] that if an algorithm is given an n-bit string x' such that $x_i = 1 \Rightarrow x'_i = 1$ and such that $|x'| \le c|x|$ (where $|x|$ is the Hamming weight of x), then a Paging algorithm that knows x' can be c-competitive. This means that (for all cache sizes) there exists a c-competitive PAGING algorithm reading $\log(1+\frac{(c-1)^{c-1}}{c^c})n + O(\log n)$ bits of advice on inputs of length n. In particular, $(\log\frac{5}{4})n + O(\log n) > 0.3219n + O(\log n)$ bits of advice suffice to be 2-competitive. The best known lower bound on the exact advice complexity of PAGING was proven in Mikkelsen [2016] by a reduction from ANTI-STRING GUESSING. This lower bound is quite far from the AOC-based upper bound. For example, it only shows that at least $0.00877n - O(\log n)$ bits are needed to be 2-competitive.

LIST UPDATE has been studied with advice in Boyar et al. [2014]. The main result is a $\frac{5}{3}$-competitive algorithm using just two bits of advice (in total). The advice tells which of the three classic algorithms TIMESTAMP, MOVETOFRONT-EVEN, and MOVETOFRONT-ODD is the best algorithm for the current input. An interesting application of this in the context of data compression has been described in Kamali and López-Ortiz [2014b], where they choose the better of two classic algorithms for LIST UPDATE based on a simple scan of the data.

9. BIN PACKING, MACHINE SCHEDULING, AND KNAPSACK

In this section, we consider three related problems.

Bin Packing. In BIN PACKING, requests are sizes in the range (0, 1]. Bins of size 1 are available, and items must be placed in a bin such that the total volume of items placed in that bin does not exceed 1. The objective is to minimize the number of bins used.

The ultimate advice for any online problem is to be informed of exactly how OPT behaves on the request sequence. For BIN PACKING, OPT uses OPT(I) bins on a request sequence I, so with $n\lceil\log \text{OPT}(I)\rceil$ bits of advice, it is possible to mimic the behavior of OPT. This was observed in Boyar et al. [2016c], where it was also established that this is essentially tight, in that a lower bound of $(n - 2\,\text{OPT}(I))\log \text{OPT}(I)$ was given. They employed the pigeonhole technique, giving a long prefix that has to be packed exactly right, depending on the unknown suffix, in order to pack all the items in the optimal number of bins.

To beat the best known lower bound for BIN PACKING of 1.54037 [Balogh et al. 2012] (the best known upper bound is 1.5815 [Heydrich and van Stee 2016]), a ratio of $\frac{3}{2}$ was obtained using $\log n + o(\log n)$ bits of advice [Boyar et al. 2016c]. The observation underlying this result is that large items fill bins sufficiently and small items are easy to pack effectively, so we need to know about medium-sized items (concretely in the range $(\frac{1}{2}, \frac{2}{3}]$), and $\lceil\log(n+1)\rceil$ bits are sufficient to specify the number of such items in the input. Different categorization schemes by Angelopoulos et al. [2015] led to a competitive ratio of $1.47012+\varepsilon$, for any fixed ε, using a constant number of advice bits, dependent on ε. They also show that 16 bits of advice are sufficient to beat the best algorithm without advice, obtaining a competitive ratio of 1.530.

Using a linear number of bits, $2n + o(n)$, to get limited information regarding OPT's packing, a ratio of $\frac{4}{3}+\varepsilon$, for any ε, was obtained in Boyar et al. [2016c]. Asymptotically, for quite large input, Renault et al. [2015] proved that one can get arbitrarily close to optimal, establishing $(1+\varepsilon)$-competitiveness using $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ bits of advice per request.

A further improvement of the $\frac{4}{3}$ result is claimed in Zhao and Shen [2014], but we have not been able to verify the result. An example problematic sequence for their algorithm is $\frac{n}{2}$ items of size $\frac{1}{3}-\varepsilon$ followed by $\frac{n}{2}$ items of size $\frac{2}{3}+\varepsilon$, where their algorithm appears to only achieve a performance ratio of $\frac{4}{3}$.

For negative results, Boyar et al. [2016c] showed that $\frac{9}{8}$ is a lower bound for algorithms with sublinear advice. Refining those methods, Angelopoulos et al. [2015] raised this lower bound to $\frac{7}{6} = 1.1\overline{6}$ and Mikkelsen [2016] to $4 - 2\sqrt{2} > 1.1715$.

An overview of these results is given in Figure 3.

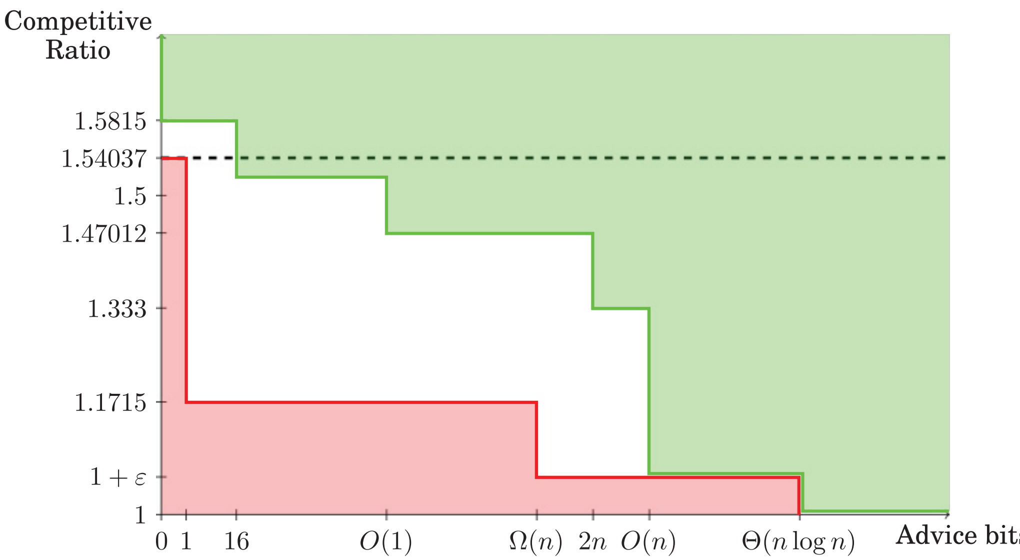

Fig. 3. The best known bounds on the advice complexity of Bin Packing. The horizontal dashed line is the currently best lower bound on (possibly randomized) Bin Packing algorithms without advice.

Finally, we mention some special cases. For a limited number m of different items, by using $m\lceil\log(n+1)\rceil + o(\log n)$ bits of advice to inform the algorithm in advance of how many items to expect of the different types, one can be essentially optimal, achieving a packing of $(1+\varepsilon)\,\textsc{Opt}(I)+1$ bins. This is essentially tight, since $(m-1)\log n - 2m\log m$ bits of advice are required to be optimal [Boyar et al. 2016c]. If all items are known to be larger than $\frac{1}{3}$, then one bit of advice is sufficient to be 1.3904-competitive [Angelopoulos et al. 2015].

Machine Scheduling. Consider Scheduling on m machines, where requests are real numbers, referred to as job sizes, and the irrevocable decision is to assign a request to a particular machine. The load of a machine is the sum of the job sizes assigned to that machine and the makespan is the maximum load of any machine. A commonly investigated objective is that of minimizing the makespan.

In Section 3, we discuss a parallel solutions algorithm from Albers and Hellwig [2014], where the objective is to minimize the makespan, that is, the maximum sum of job sizes assigned to any one machine. The parallel solutions algorithm can be viewed as a $(\frac{4}{3}+\varepsilon)$-competitive algorithm using $O(\log^2 \frac{1}{\varepsilon})$ bits of advice in the Tape Model. The same article gives a $(1+\varepsilon)$-competitive algorithm that can be viewed as using $O(\frac{1}{\varepsilon}\log\frac{m}{\varepsilon}\log\frac{1}{\varepsilon})$ advice bits.

In Boyar et al. [2016b], $(1+\varepsilon)$-competitive algorithms are given for weighted scheduling problems with various objective functions: For minimizing a norm (the makespan, for example) on related machines (known speed ratios between the machines), an algorithm reading $O(\frac{1}{\varepsilon}\log^2 n)$ advice bits is given. For minimizing a norm on a constant number of unrelated machines, an algorithm reading $O(\frac{1}{\varepsilon^m}\log^{m+1} n)$ bits of advice is given. The same advice complexity is obtained for maximizing a semi-norm (the minimum load as in Machine Covering, for example) on a constant number of unrelated machines. For a non-constant number of unrelated machines, the expressions for the advice complexity are more complicated; see the article for details.

For the Per Request Model, Renault et al. [2015] obtain a competitive ratio of $1+\varepsilon$ for minimizing makespan using $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ bits of advice per request. Similar results are obtained for MACHINE COVERING and minimization of the L_p norm, $p \geq 2$. Complementing these results, using the pigeonhole technique, they establish a $(1-\frac{2m}{n})\log m$ lower bound on advice per request in order to obtain optimality, that is, almost as much advice is required as is used by the trivial optimal algorithm with advice that receives $\lceil \log m \rceil$ bits of advice per request, indicating which machine to place a job on.

Knapsack. In the Introduction, UNIFORM KNAPSACK was used as an example (Example 2). An algorithm from Böckenhauer et al. [2014c] using one advice bit was described: The advice bit indicates whether the input sequence contains an item of size at least $\frac{1}{2}$. If it does, then the first item accepted by the algorithm is the first item of size at least $\frac{1}{2}$. Otherwise, the algorithm accepts items greedily. In this section, we describe other results from this article.

The competitive ratio of the above algorithm cannot be improved using a few additional advice bits; no algorithm reading fewer than $\lfloor \log(n-1) \rfloor$ bits of advice has a competitive ratio better than 2. On the other hand, for any constant $\varepsilon > 0$, there is a $(1+\varepsilon)$-competitive algorithm reading $O(\frac{1}{\varepsilon}\log n)$ bits of advice. For optimality, $n-1$ bits of advice are necessary and sufficient.

If one considers the obvious randomized algorithm based on the above 2-competitive algorithm, then its competitive ratio is 4: Simply flip a coin instead of reading an advice bit. There is a related 2-competitive randomized algorithm using only one bit of randomness: One of the deterministic algorithms to choose between is again just accepting everything possible; the other is rejecting until the first item, which would have been rejected had everything before it been accepted, and then accepting from there when possible. This is best possible, since no randomized algorithm can have competitive ratio better than 2.

For the weighted version, where each item has both a size and a value, any (possibly randomized) algorithm reading fewer than $\log n$ bits of advice has unbounded competitive ratio. On the other hand, if all values and weights can be represented within polynomial space, then $O(\frac{\sqrt{1+\varepsilon}}{\sqrt{1+\varepsilon}-1}\log n)$ advice bits suffice to be $(1+\varepsilon)$-competitive.

10. GRAPH COLORING

Being of both practical and theoretical interest, graph coloring problems have been extensively studied from an online perspective. In fact, some of the earliest results on online graph coloring predate the formal introduction of competitive analysis. We refer to Kierstead and Trotter [1991] for a good (although slightly dated) survey on VERTEX COLORING. In this section, we survey some of the results obtained on various graph coloring problems in the advice complexity model. With a single notable exception, it generally turns out that a lot of advice is needed in order to obtain good online graph coloring algorithms.

10.1. Vertex Coloring

The most classic graph coloring problem is VERTEX COLORING, where the vertices of a graph must be colored such that no two neighbors receive the same color. The aim is to use as few colors as possible. In the most studied online model, the *vertex-arrival* model, vertices arrive one by one, each with information about its edges to vertices that have already arrived. Usually, the colors are enumerated starting from 1.

Without advice, VERTEX COLORING is an extremely difficult problem; Halldórsson and Szegedy [1994] showed that any (possibly randomized) online algorithm has a competitive ratio of $\Omega(\frac{n}{\log^2 n})$. The hardness of the problem carries over to the advice setting;

applying a direct product theorem (see Section 6) to the lower bound of Halldórsson and Szegedy [1994], Mikkelsen [2016] showed that any $O(n^{1-\varepsilon})$-competitive VERTEX COLORING algorithm must read $\Omega(n \log n)$ bits of advice. This is an unusually strong advice complexity lower bound. VERTEX COLORING is so far the only known example of a natural online problem where linear advice is not enough to obtain a truly sublinear competitive ratio. Also, note that $O(n \log n)$ bits of advice trivially suffice to achieve optimality. In fact, $n \log n - n \log \log n + O(n)$ advice bits are necessary and sufficient for an optimal coloring, even necessary if the vertices arrive in a breadth-first order [Forišek et al. 2012]. Thus, VERTEX COLORING has a sharp phase transition in its advice complexity.

On trees, First-Fit (the greedy algorithm using the lowest available color) uses at most $\lfloor \log n \rfloor + 1$ colors [Gyárfás and Lehel 1990], thus obtaining a competitive ratio of $\frac{1}{2} \log n$. This is a best possible result, since, even on trees, any deterministic online algorithm can be forced to use $\lfloor \log n \rfloor + 1$ colors [Gyárfas and Lehel 1988], while OPT, of course, only needs two. Since VERTEX COLORING is $\vee$-repeatable, and since the lower bound of $\frac{1}{2} \log n \in \omega(1)$ does not depend on the algorithm not knowing n or $\text{OPT}(I)$, it follows that no VERTEX COLORING algorithm with $o(n)$ bits of advice can achieve a constant competitive ratio, even on trees [Mikkelsen 2016] (see Section 6).

For bipartite graphs, any deterministic online algorithm without advice can be forced to use $2 \log n - 10$ colors [Gutowski et al. 2014]. Thus, coloring bipartite graphs is harder than coloring trees. On the other hand, the online algorithm (without advice) Bipartite First-Fit (BFF) uses at most $2 \log n$ colors [Kierstead 1998] for $n \geq 2$. For each vertex v, BFF simply uses the smallest color not used in the opposite partition of the connected component containing v.

Building on BFF, a family of algorithms, A_k, with advice for coloring bipartite graphs was given in Bianchi et al. [2014a], obtaining a tradeoff between competitive ratio and advice. For $k \geq 2$, the algorithm A_k uses advice to ensure that the color $k-1$ is only used in one partition of the final graph and that the color k is only used in the other partition. For each vertex v, if BFF would use a color no larger than $k-2$, then A_k uses this color. Otherwise, if at least one of the colors $k-1$ and k is already used in the connected component containing v, then the algorithm can deduce which color to use. If this is not the case, then the algorithm reads one bit of advice to decide which of the colors $k-1$ and k to use. Since BFF uses color $k-1$ only if the requested vertex is contained in a connected component of at least $2^{\frac{k-1}{2}}$ vertices, and since the algorithm does not use advice for more than one vertex within a connected component, the number of advice bits used is at most $\frac{n-1}{2^{\frac{k-1}{2}}} = \frac{n-1}{\sqrt{2^{k-1}}}$. This shows that $O(\sqrt{n})$ advice bits suffice to use fewer than $\log n$ colors, beating the lower bound for deterministic online algorithms without advice. For two and three colors, the upper bound is complemented with essentially tight lower bounds of $n-3$ and $\frac{n}{2}-4$ bits, respectively.

Note that the approach taken by the algorithms A_k resembles the warning signal technique described in Section 5. However, a sublinear number of advice bits is obtained, because the algorithms can detect themselves when they need advice.

In Steffen [2014], the algorithm A_k from Bianchi et al. [2014a] was specialized to trees, using First-Fit instead of Bipartite First-Fit. Since, on trees, First-Fit uses the color $k-1$ only if the requested vertex belongs to a component of at least 2^{k-2} vertices [Gyárfas and Lehel 1988], a k-coloring is obtained using at most $\frac{n-1}{2^{k-2}}$ bits of advice. Thus, for each additional color, the number of advice bits is halved.

For 3-coloring of trees, a linear lower bound of approximately $0.0328n$ advice bits is given in Steffen [2014]. The dissertation also contains linear lower (and upper) bounds for coloring combs and caterpillars with two or three colors. Note that caterpillars can be 4-colored without advice, since no vertex has more than three neighbors.

For 3-colorable graphs, the trivial upper bound of $(\log 3)n$ is essentially tight, even if the graphs are chordal [Seibert et al. 2013].

10.2. Edge Coloring and Variants of Vertex Coloring

Many variants of VERTEX COLORING have been studied. Here we mention two of them.

For $L(i, j)$-COLORING, each pair of neighboring vertices must receive colors that are at least i apart, and each pair of vertices at distance two must receive colors that are at least j apart. The aim is to minimize the span of the coloring, that is, the difference, λ, between the largest and smallest color used (thus, potentially, $\lambda + 1$ colors are used).

For MULTI-COLORING, a graph is given from the beginning and the requests are vertices, with possible repetitions. For each request, an (additional) color must be assigned to the requested vertex. The colors assigned to a vertex and its neighbors must all be distinct.

Though not as famous as VERTEX COLORING, many articles have been devoted to EDGE COLORING. Analogous to VERTEX COLORING, the edges of a graph must be colored such that no two adjacent edges receive the same color, and the aim is to use as few colors as possible. In the online version, one typically uses the *edge arrival model*, where the edges arrive one by one, each with information about adjacent edges among those that have already arrived. Adhering to standard notation in graph theory, where n denotes the number of vertices and m the number of edges, we let m denote the sequence length for this particular problem.

In Bianchi et al. [2014b], $L(2, 1)$-COLORING of paths was studied. In the offline setting, the color range $0, 1, \ldots, \lambda = 4$ is sufficient. For the best possible online algorithm without advice, the color span is $\lambda = 6$ in the worst case, resulting in a competitive ratio of $\frac{3}{2}$. To obtain a better competitive ratio, a linear number of advice bits are necessary (a lower bound of approximately $3.9402 \cdot 10^{-10}n$ bits for obtaining $\lambda = 5$ is given in Bianchi et al. [2014b]). This was the first example of a natural online problem with the property that beating the best deterministic online algorithm without advice requires a linear number of advice bits. Note that linear advice trivially suffices to be optimal (in fact, approximately $0.6955n$ bits of advice are sufficient [Bianchi et al. 2014b]). Since $\lambda = 4$ is obtainable in the offline setting, the linear lower bound for $\lambda = 5$, together with the derandomization technique of Böckenhauer et al. [2011] mentioned in Section 4, implies a lower bound of $\frac{5}{4}$ on the competitive ratio of any randomized online algorithm for the problem.

For edge coloring of a graph with m edges and maximum degree Δ, $m\lceil\log(\Delta + 1)\rceil$ bits of advice trivially suffice for an optimal solution (by Vizing's Theorem [Vizing 1965]). In Mikkelsen [2015], it was shown that, in the Per Request Model, this bound is asymptotically tight. On the other hand, the article also shows that, for graphs of bounded degeneracy (including planar graphs), $O(m)$ advice bits are sufficient to be optimal. For trees, the warning signal technique applied to First-Fit yields an optimal algorithm for trees using exactly one bit of advice per edge: If the advice bit is a 0, then First-Fit colors the current edge as usual. If the advice bit is a 1, then First-Fit will skip the lowest numbered color available and instead use the second lowest numbered color available.

For edge coloring without advice, the competitive ratio is 2 on trees as well as in general [Bar-Noy et al. 1992]. In Mikkelsen [2015], it was shown that, even for trees, linear advice is necessary to beat the best deterministic online algorithm without advice. Comparing the proof and the proof of the corresponding result for $L(2, 1)$-COLORING, it turns out that they are in fact quite similar. Based on this observation, Mikkelsen [2016] showed that these problems are indeed hard for essentially the same reason; they are both $\vee$-repeatable (see Section 6).

Recall that a problem being $\vee$-repeatable is not enough for a lower bound to carry over from deterministic online algorithms to online algorithms with sublinear advice; it is required that the lower bound does not depend on the online algorithm not knowing $\text{OPT}(I)$ (or n). It turns out that this requirement is vital for the lower bound technique to work. In fact, Christ et al. [2015] showed that sublinear advice suffices to be optimal for MULTI-COLORING on a path, whereas it is known that an algorithm without advice cannot be better than $\frac{4}{3}$-competitive [Chan et al. 2006]. This may seem at odds with the previously mentioned result, but the reason is that the $\frac{4}{3}$ lower bound relies heavily on the algorithm not knowing $\text{OPT}(I)$. In fact, it is shown in Christ et al. [2015] that if $\text{OPT}(I)$ is known (note that $\text{OPT}(I)$ can be encoded using $O(\log n)$ bits of advice), then it is easy for an online algorithm to be optimal. The case where the exact value of $\text{OPT}(I)$ is not known (or not communicated to the algorithm) is also considered, resulting in a tradeoff, where the competitive ratio ranges from 1 to $\frac{9}{8}$ and the number of advice bits ranges from $\log n + O(\log \log n)$ to $O(\log \log n)$.

A standard topology for modeling cellular networks is hexagonal graphs, which are graphs that can be obtained by placing (at most) one node in each cell of a hexagonal grid (a beehive pattern) and adding an edge between any pair of nodes placed in neighboring cells. On hexagonal graphs, no MULTI-COLORING algorithm without advice can be better than $\frac{3}{2}$-competitive [Chan et al. 2010]. In Christ et al. [2015], it is shown that $\Omega(n)$ bits are necessary for obtaining a ratio better than $\frac{5}{4}$, $n + 2|V|$ bits are sufficient to obtain a ratio of $\frac{4}{3}$, and $\log n + O(\log \log n)$ bits suffice to obtain a ratio of $\frac{3}{2}$.

11. GRAPH EXPLORATION

GRAPH EXPLORATION is a family of problems where an agent (sometimes called a robot) with a fixed starting point explores an unknown graph. The goal is usually to visit each vertex of the graph, minimizing the total cost of following edges. Sometimes assumptions are made on the structure of the graph.

These problems are unusual online problems in the following sense: For most other online problems, it is possible to fix an input sequence, $I = x_1, \ldots, x_n$, such that x_i is revealed in round i no matter how the algorithm behaves (of course, if it is deterministic, we know what it will do). In GRAPH EXPLORATION, even when an input is fixed, the new information the algorithm gains in each step still depends on what it has done in previous steps. Thus, an input sequence cannot be defined independently of an algorithm.

Kalyanasundaram and Pruhs [1994] present an algorithm for GRAPH EXPLORATION that is 16-competitive on planar graphs. In Megow et al. [2012], it is shown that this algorithm does not have a constant competitive ratio on general graphs but is $16(1+2g)$-competitive for graphs with genus at most g. Furthermore, Megow et al. [2012] give an algorithm with constant competitive ratio for general graphs with a bounded number of distinct weights. The main open question is whether there exists an algorithm that has a constant competitive ratio for arbitrary graphs with arbitrary weights.

In Fraigniaud et al. [2008], TREE EXPLORATION with advice is considered. A robot explores an unknown undirected tree, and its goal is to visit every vertex at least once. Each move incurs a cost of 1. When the robot is at a given vertex, it can see the labels of the neighboring vertices, but the advice is only allowed to depend on the structure of the tree and not the labels (which are assigned adversarially after the advice is given). Without advice, the best possible competitive ratio for deterministic online algorithms is 2. This is achieved by depth-first-search (DFS). It is shown that roughly $\log \log D$ bits of advice are necessary and sufficient to achieve a better competitive ratio (D is the diameter of the graph). For the upper bound, one bit is used to choose between two algorithms; one is DFS and the other is a more sophisticated algorithm using an approximation of D. The model used is the Tape Model, except that the length of the advice is known to the algorithm (see Section 2). The lower bound is shown on paths.

The more general case, GRAPH EXPLORATION, is studied in Dobrev et al. [2012]. Here the unknown undirected graph is arbitrary and edges have non-negative weights. When the robot is at a vertex, it can see the weight of each adjacent edge and the label of its other endpoint. The goal is to visit each vertex and return to the starting point. Each time an edge is traversed, it costs the weight of that edge. Here the advice is allowed to depend on the labels. It is shown that $\Theta(n \log n)$ bits are necessary and sufficient to be optimal. A $(6+\varepsilon)$-competitive algorithm with $O(n)$ advice is also given. The algorithm works by traversing edges of a minimum spanning tree and some additional light edges.

A related problem, TREASURE HUNT, is studied in Komm et al. [2015]. The model is the same as in Dobrev et al. [2012] with the following difference: The robot is given the label of a target vertex and the goal is to visit that vertex. It is observed that a simple greedy algorithm has competitive ratio $\Theta(n)$, and this is best possible for online algorithms without advice (even on unweighted graphs and if randomization is allowed). It is shown that there is an optimal algorithm reading n bits of advice. For each vertex, one bit of advice indicates if that vertex is on a fixed shortest path. For the unweighted case, it is shown that $\Theta(\frac{n}{c})$ bits are necessary and sufficient to achieve a competitive ratio of c (where c has to be of a certain form but may depend on n).

12. OPEN PROBLEMS

We end the survey with a few open problems:

—Can advice complexity be used to build a complexity theory for online computation? The study of online algorithms with advice has led to the first complexity classes in online algorithms and to new possibilities for proving results on randomized online algorithms and semi-online algorithms. Further study may lead to additional meaningful complexity classes and new fundamental insights into the properties of online problems.
—Is it possible for a k-SERVER algorithm to be $(1+\varepsilon)$-competitive with $O(1)$ bits of advice per request?
Currently, it is known that the answer to this problem is "yes" if the underlying metric space is the Euclidean plane. It is also known that $\Omega(\log k)$ bits per request are required to be 1-competitive.
—How small a competitive ratio can be achieved for BIN PACKING using constant advice?
—Are there further connections between advice and randomization in online computation that have not yet been discovered?

APPENDIX

A. PROBLEMS STUDIED IN ADVICE COMPLEXITY MODELS

We list problems explicitly studied in advice complexity models.

—Inherently online problems
  —K-server [Böckenhauer et al. 2011; Emek et al. 2011; Renault and Rosén 2015; Mikkelsen 2016]
  —K-server on sparse graphs [Gupta et al. 2013]
  —K-server on a path [Smula 2015]
  —List update [Boyar et al. 2014; Mikkelsen 2016]; application in Kamali and López-Ortiz [2014b]

  - —Paging [Böckenhauer et al. 2009; Mikkelsen 2016]
  - —Metrical task systems [Emek et al. 2011]
  - —Sleep state management [Böckenhauer et al. 2015; Mikkelsen 2016]
  - —Online search [Clemente et al. 2016]
- —Scheduling and packing problems
  - —Scheduling on identical machines with constant advice [Albers and Hellwig 2014; Dohrau 2015]
  - —Scheduling with sublinear advice [Boyar et al. 2016b]
  - —Job shop scheduling [Wehner 2014, 2015; Komm and Královič 2011]
  - —Job shop with randomized adversary [Wehner 2014]
  - —Linear advice approximation schemes for bin packing and scheduling [Renault et al. 2015]
  - —Bin packing with sublinear advice [Boyar et al. 2016c; Angelopoulos et al. 2015; Mikkelsen 2016]
  - —Dual bin packing [Renault 2016]
  - —Bin packing [Zhao and Shen 2014][2]
  - —Square packing [Kamali and López-Ortiz 2014a]
  - —Reordering buffer management [Adamaszek et al. 2016; Mikkelsen 2016]
  - —Buffer management [Dorrigiv et al. 2012]
  - —Knapsack [Böckenhauer et al. 2014c]
  - —Set cover [Komm et al. 2012]
- —Coloring problems
  - —2-vertex coloring [Bianchi et al. 2014a; Mikkelsen [2016]
  - —3-vertex coloring [Seibert et al. 2013]
  - —Graph coloring, general graphs [Forišek et al. 2012; Mikkelsen 2016]
  - —Graph coloring on paths [Forišek et al. 2012]
  - —Multi-coloring paths and grids [Christ et al. 2015]
  - —Edge coloring [Mikkelsen 2015]
  - —$L(2,1)$-coloring on paths [Bianchi et al. 2014b]
- —Other graph problems
  - —Tree exploration with advice [Fraigniaud et al. 2008]
  - —Graph exploration [Dobrev et al. 2012; Barun Gorain 2016]
  - —Treasure hunt [Komm et al. 2015]
  - —Bipartite matching [Dürr et al. 2016; Miyazaki 2014; Mikkelsen 2016]
  - —Independent set [Halldórsson et al. 2002; Boyar et al. 2015a]
  - —Independent set with known supergraph [Dobrev et al. 2015]
  - —Vertex cover on restricted graph classes [Steffen 2014]
  - —Steiner trees [Barhum 2014]
  - —Disjoint path allocation [Barhum et al. 2014; Gebauer et al. 2015]
  - —Minimum spanning tree [Bianchi et al. 2016]
  - —Matching on restricted graph classes [Keller 2014]
- —Asymmetric online covering
  - —AOC [Boyar et al. 2015a] (complexity class comprising, among other problems, independent set, vertex cover, dominating set, disjoint path allocation)
  - —Induced subgraph [Komm et al. 2016]
  - —Weighted AOC [Boyar et al. 2016b]
- —Miscellaneous
  - —String guessing/generalized matching pennies [Böckenhauer et al. 2014b; Emek et al. 2011; Krug 2015]
  - —Repeated matrix games [Mikkelsen 2016]

[2]We discuss some issues with the result in Section 9.

—Graph coloring with randomized adversary [Burjons et al. 2016]
—Brief survey [Královič 2014]

REFERENCES

Dimitris Achlioptas, Marek Chrobak, and John Noga. 2000. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.* 234, 1–2 (2000), 203–218. DOI:http://dx.doi.org/10.1016/S0304-3975(98)00116-9 Preliminary version in ESA'96.

Anna Adamaszek, Marc P. Renault, Adi Rosén, and Rob van Stee. 2016. Reordering buffer management with advice. *J. Sched.* (2016). To appear. DOI:http://dx.doi.org/10.1007/s10951-016-0487-8 Preliminary version in WAOA'13.

Susanne Albers and Matthias Hellwig. 2012. Semi-online scheduling revisited. *Theor. Comput. Sci.* 443 (2012), 1–9. DOI:http://dx.doi.org/10.1016/j.tcs.2012.03.031

Susanne Albers and Matthias Hellwig. 2014. Online makespan minimization with parallel schedules. In *SWAT (LNCS)*, Vol. 8503. Springer, 13–25. DOI:http://dx.doi.org/10.1007/978-3-319-08404-6_2

Spyros Angelopoulos, Christoph Dürr, Shahin Kamali, Marc P. Renault, and Adi Rosén. 2015. Online bin packing with advice of small size. In *WADS (LNCS)*, Vol. 9214. Springer, 40–53. DOI:http://dx.doi.org/10.1007/978-3-319-21840-3_4

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

Yossi Azar and Leah Epstein. 1997. On-line machine covering. *J. Sched.* 1, 2 (1997), 67–77. DOI:http://dx.doi.org/10.1002/(SICI)1099-1425(199808)1:2⟨67::AID-JOS6⟩3.0.CO;2-Y Preliminary version in ESA'97.

Yossi Azar and Oded Regev. 2001. On-line bin-stretching. *Theor. Comput. Sci.* 268 (2001), 17–41. DOI:http://dx.doi.org/10.1016/S0304-3975(00)00258-9 Preliminary version in RANDOM'98.

János Balogh, József Békési, and Gábor Galambos. 2012. New lower bounds for certain classes of bin packing algorithms. *Theor. Comput. Sci.* 440–441 (2012), 1–13. DOI:http://dx.doi.org/10.1016/j.tcs.2012.04.017 Preliminary version in WAOA'10.

Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. 2015. A polylogarithmic-competitive algorithm for the *k*-server problem. *J. ACM* 62, 5 (2015), 40:1–40:49. DOI:http://dx.doi.org/10.1145/2783434 Preliminary version in FOCS'11.

Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. 1992. The greedy algorithm is optimal for on-line edge coloring. *Inform. Process. Lett.* 44 (1992), 251–253. DOI:http://dx.doi.org/10.1016/0020-0190(92)90209-E

Kfir Barhum. 2014. Tight bounds for the advice complexity of the online minimum steiner tree problem. In *SOFSEM (LNCS)*, Vol. 8327. Springer, 77–88. DOI:http://dx.doi.org/10.1007/978-3-319-04298-5_8

Kfir Barhum, Hans-Joachim Böckenhauer, Michal Forišek, Heidi Gebauer, Juraj Hromkovič, Sacha Krug, Jasmin Smula, and Björn Steffen. 2014. On the power of advice and randomization for the disjoint path allocation problem. In *SOFSEM (LNCS)*, Vol. 8327. Springer, 89–101. DOI:http://dx.doi.org/10.1007/978-3-319-04298-5_9

Andrzej Pelc Barun Gorain. 2016. Deterministic graph exploration with advice. *ArXiv* (2016). http://arxiv.org/abs/1607.01657 arXiv:1607.01657 [cs.DS].

Laszlo A. Belady. 1966. A study of replacement algorithms for virtual-storage computer. *IBM Syst. J.* 5, 2 (1966), 78–101. DOI:http://dx.doi.org/10.1147/sj.52.0078

Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. 1994. On the power of randomization in on-line algorithms. *Algorithmica* 11, 1 (1994), 2–14. DOI:http://dx.doi.org/10.1007/BF01294260 Preliminary version in STOC'90.

Maria Paola Bianchi, Hans-Joachim Böckenhauer, Tatjana Brülisauer, Dennis Komm, and Beatrice Palano. 2016. Online minimum spanning tree with advice. In *SOFSEM (LNCS)*, Vol. 9587. Springer, 195–207. DOI:http://dx.doi.org/10.1007/978-3-662-49192-8_16

Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. 2014a. Online coloring of bipartite graphs with and without advice. *Algorithmica* 70, 1 (2014), 92–111. DOI:http://dx.doi.org/10.1007/s00453-013-9819-7 Preliminary version in COCOON'12.

Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, Sacha Krug, and Björn Steffen. 2014b. On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. *Theor. Comput. Sci.* 554 (2014), 22–39. DOI:http://dx.doi.org/10.1016/j.tcs.2014.06.027 Preliminary version in COCOON'13.

Avrim Blum and Carl Burch. 2000. On-line learning and the metrical task system problem. *Machine Learning* 39, 1 (2000), 35–58. DOI:http://dx.doi.org/10.1023/A:1007621832648 Preliminary version in COLT'97.

Hans-Joachim Böckenhauer, Richard Dobson, Sacha Krug, and Kathleen Steinhöfel. 2015. On energy-efficient computations with advice. In *COCOON (LNCS)*, Vol. 9198. Springer, 747–758. DOI:http://dx.doi.org/10.1007/978-3-319-21398-9_58

Hans-Joachim Böckenhauer, Juraj Hromkovič, and Dennis Komm. 2014a. A technique to obtain hardness results for randomized online algorithms—A survey. In *Computing with New Resources (LNCS)*, Vol. 8808. Springer, 264–276. DOI:http://dx.doi.org/10.1007/978-3-319-13350-8_20

Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. 2014b. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.* 554 (2014), 95–108. DOI:http://dx.doi.org/10.1016/j.tcs.2014.06.006 Preliminary version in COCOON'13.

Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. 2011. On the advice complexity of the k-server problem. In *ICALP (1) (LNCS)*, Vol. 6755. Springer, 207–218. DOI:http://dx.doi.org/10.1007/978-3-642-22006-7_18

Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. 2009. On the advice complexity of online problems. In *ISAAC (LNCS)*, Vol. 5878. Springer, 331–340. DOI:http://dx.doi.org/10.1007/978-3-642-10631-6_35

Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. 2014c. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.* 527 (2014), 61–72. DOI:http://dx.doi.org/10.1016/j.tcs.2014.01.027 Preliminary version in LATIN'12.

Martin Böhm. 2016. Lower bounds for online bin stretching with several bins. In *Student Research Forum Papers and Posters at SOFSEM (CEUR Workshop Proceedings)*, Vol. 1548. RWTH Aachen University, 1–12. http://ceur-ws.org/Vol-1548/001-Bohm.pdf.

Allan Borodin and Ran El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press.

Allan Borodin, Nathan Linial, and Michael E. Saks. 1992. An optimal on-line algorithm for metrical task system. *J. ACM* 39, 4 (1992), 745–763. DOI:http://dx.doi.org/10.1145/146585.146588 Preliminary version in STOC'87.

Joan Boyar, Leah Epstein, Lene M. Favrholdt, Kim S. Larsen, and Asaf Levin. 2016a. Online bounded analysis. In *CSR (LNCS)*, Vol. 9691. Springer, 131–145. DOI:http://dx.doi.org/10.1007/978-3-319-34171-2_10

Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. 2015a. Advice complexity for a class of online problems. In *STACS (LIPIcs)*, Vol. 30. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 116–129. DOI:http://dx.doi.org/10.4230/LIPIcs.STACS.2015.116 Full paper to appear in *Theor. Comput. Syst*.

Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. 2016b. Weighted online problems with advice. In *IWOCA (LNCS)*, Vol. 9843. Springer, 179–190. DOI:http://dx.doi.org/10.1007/978-3-319-44543-4_14

Joan Boyar, Sandy Irani, and Kim S. Larsen. 2015b. A comparison of performance measures for online algorithms. *Algorithmica* 72, 4 (2015), 969–994. DOI:http://dx.doi.org/10.1007/s00453-014-9884-6 Preliminary version in WADS'09.

Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. 2014. On the list update problem with advice. In *LATA (LNCS)*, Vol. 8370. Springer, 210–221. DOI:http://dx.doi.org/10.1007/978-3-319-04921-2_17 Full paper to appear in *Inform. Comput.*

Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. 2016c. Online bin packing with advice. *Algorithmica* 74, 1 (2016), 507–527. DOI:http://dx.doi.org/10.1007/s00453-014-9955-8 Preliminary version in STACS'14.

Elisabet Burjons, Juraj Hromkovič, Xavier Muñoz, and Walter Unger. 2016. Online graph coloring with advice and randomized adversary. In *SOFSEM (LNCS)*, Vol. 9587. Springer, 229–240. DOI:http://dx.doi.org/10.1007/978-3-662-49192-8_19

Joseph Wun-Tat Chan, Francis Y. L. Chin, Deshi Ye, and Yong Zhang. 2010. Absolute and asymptotic bounds for online frequency allocation in cellular networks. *Algorithmica* 58, 2 (2010), 498–515. DOI:http://dx.doi.org/10.1007/s00453-009-9279-2

Joseph Wun-Tat Chan, Francis Y. L. Chin, Deshi Ye, Yong Zhang, and Hong Zhu. 2006. Frequency allocation problems for linear cellular networks. In *ISAAC (LNCS)*, Vol. 4288. Springer, 61–70. DOI:http://dx.doi.org/10.1007/11940128_8

Marie G. Christ, Lene M. Favrholdt, and Kim S. Larsen. 2015. Online multi-coloring with advice. *Theor. Comput. Sci.* 596 (2015), 79–91. DOI:http://dx.doi.org/10.1016/j.tcs.2015.06.044 Preliminary version in WAOA'14.

Jhoirene Clemente, Juraj Hromkovič, Dennis Komm, and Christian Kudahl. 2016. Advice complexity of the online search problem. In *IWOCA (LNCS)*, Vol. 9843. Springer, 203–212. DOI:http://dx.doi.org/10.1007/978-3-319-44543-4_16

Thomas M. Cover and Joy A. Thomas. 2006. *Elements of Information Theory* (2nd ed.). Wiley. DOI:http://dx.doi.org/10.1002/047174882X

Stefan Dobrev, Rastislav Královič, and Richard Královič. 2015. Advice complexity of maximum independent set in sparse and bipartite graphs. *Theor. Comput. Syst.* 56, 1 (2015), 197–219. DOI:http://dx.doi.org/10.1007/s00224-014-9592-2 Preliminary version in WAOA'12.

Stefan Dobrev, Rastislav Královič, and Euripides Markou. 2012. Online graph exploration with advice. In *SIROCCO (LNCS)*, Vol. 7355. Springer, 267–278. DOI:http://dx.doi.org/10.1007/978-3-642-31104-8_23

Stefan Dobrev, Rastislav Královič, and Dana Pardubská. 2009. Measuring the problem-relevant information in input. *RAIRO - Theor. Inf. Appl.* 43, 3 (2009), 585–613. DOI:http://dx.doi.org/10.1051/ita/2009012 Preliminary version entitled "How Much Information about the Future Is Needed?" in SOFSEM'08.

Jérôme Dohrau. 2015. Online makespan scheduling with sublinear advice. In *SOFSEM (LNCS)*, Vol. 8939. Springer, 177–188. DOI:http://dx.doi.org/10.1007/978-3-662-46078-8_15

Reza Dorrigiv, Meng He, and Norbert Zeh. 2012. On the advice complexity of buffer management. In *ISAAC (LNCS)*, Vol. 7676. Springer, 136–145. DOI:http://dx.doi.org/10.1007/978-3-642-35261-4_17

Reza Dorrigiv and Alejandro López-Ortiz. 2005. A survey of performance measures for on-line algorithms. *SIGACT News* 36 (2005), 67–81.

Christoph Dürr, Christian Konrad, and Marc P. Renault. 2016. On the power of advice and randomization for online bipartite matching. In *ESA (LIPIcs)*, Vol. 57. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 37:1–37:16. DOI:http://dx.doi.org/10.4230/LIPIcs.ESA.2016.37

Tomás Ebenlendr and Jirí Sgall. 2009. Optimal and online preemptive scheduling on uniformly related machines. *J. Sched.* 12, 5 (2009), 517–527. DOI:http://dx.doi.org/10.1007/s10951-009-0119-7 Preliminary version in STACS'04.

Peter Elias. 1975. Universal codeword sets and representations of the integers. *IEEE T. Inform. Theory* 21, 2 (1975), 194–203. DOI:http://dx.doi.org/10.1109/TIT.1975.1055349

Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. 2011. Online computation with advice. *Theor. Comput. Sci.* 412, 24 (2011), 2642–2656. DOI:http://dx.doi.org/10.1016/j.tcs.2010.08.007 Preliminary version in ICALP'09.

Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.* 69, 3 (2004), 485–497. DOI:http://dx.doi.org/10.1016/j.jcss.2004.04.011 Preliminary version in STOC'03.

Rudolf Fleischer and Michaela Wahl. 2000. Online scheduling revisited. *J. Sched.* 3 (2000), 343–353. DOI:http://dx.doi.org/10.1007/3-540-45253-2_19 Preliminary version in ESA'00. The doi is for the conference version.

Michal Forišek, Lucia Keller, and Monika Steinová. 2012. Advice complexity of online coloring for paths. In *LATA (LNCS)*, Vol. 7183. Springer, 228–239. DOI:http://dx.doi.org/10.1007/978-3-642-28332-1_20

Michal Forišek, Lucia Keller, and Monika Steinová. 2012. Advice complexity of online graph coloring. (2012). http://people.ksp.sk/ misof/junk/chwd.pdf Unpublished manuscript.

Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. 2008. Tree exploration with advice. *Inf. Comput.* 206, 11 (2008), 1276–1287. DOI:http://dx.doi.org/10.1016/j.ic.2008.07.005 Preliminary version in MFCS'06.

Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. 2010. Communication algorithms with advice. *J. Comput. Syst. Sci.* 76, 3-4 (2010), 222–232. DOI:http://dx.doi.org/10.1016/j.jcss.2009.07.002 Preliminary verison in PODC'06.

Heidi Gebauer, Dennis Komm, Rastislav Královič, Richard Královič, and Jasmin Smula. 2015. Disjoint path allocation with sublinear advice. In *COCOON (LNCS)*, Vol. 9198. Springer, 417–429. DOI:http://dx.doi.org/10.1007/978-3-319-21398-9_33

R. L. Graham. 1966. Bounds for certain multiprocessing anomalies. *Bell Syst. Tech. J.* 45, 9 (1966), 1563–1581. DOI:http://dx.doi.org/10.1002/j.1538-7305.1966.tb01709.x

Sushmita Gupta, Shahin Kamali, and Alejandro López-Ortiz. 2013. On advice complexity of the k-server problem under sparse metrics. In *SIROCCO (LNCS)*, Vol. 8179. Springer, 55–67. DOI:http://dx.doi.org/10.1007/978-3-319-03578-9_5

Grzegorz Gutowski, Jakub Kozik, Piotr Micek, and Xuding Zhu. 2014. Lower bounds for on-line graph colorings. In *ISAAC (LNCS)*, Vol. 8889. Springer, 507–515. DOI:http://dx.doi.org/10.1007/978-3-319-13075-0_40

András Gyárfas and Jenö Lehel. 1988. Online and first-fit colorings of graphs. *J. Graph Theor.* 12, 2 (1988), 217–227. DOI:http://dx.doi.org/10.1002/jgt.3190120212

András Gyárfás and Jenő Lehel. 1990. First fit and on-line chromatic number of families of graphs. *Ars Combinatoria* 29, C (1990), 168–176.

Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. 2002. Online independent sets. *Theor. Comput. Sci.* 289, 2 (2002), 953–962. DOI:http://dx.doi.org/10.1016/S0304-3975(01)00411-X Preliminary version in COCOON'00.

Magnús M. Halldórsson and Mario Szegedy. 1994. Lower bounds for on-line graph coloring. *Theor. Comput. Sci.* 130, 1 (1994), 163–174. DOI:http://dx.doi.org/10.1016/0304-3975(94)90157-0 Preliminary version in SODA'92.

Sandy Heydrich and Rob van Stee. 2016. Beating the harmonic lower bound for online bin packing. In *ICALP (LIPIcs)*, Vol. 55. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 41:1–41:14. DOI:http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.41

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.* 58, 301 (1963), 13–30. DOI:http://dx.doi.org/10.2307/2282952

Juraj Hromkovič, Rastislav Královič, and Richard Královič. 2010. Information complexity of online problems. In *MFCS (LNCS)*, Vol. 6281. Springer, 24–36. DOI:http://dx.doi.org/10.1007/978-3-642-15155-2_3

Yikun Huang and Yong Wu. 2010. Optimal semi-online algorithm for machine covering with nonsimultaneous machine available times. *Int. Math. Forum* 5, 4 (2010), 185–190. http://www.m-hikari.com/imf-2010/1-4-2010/wuyongIMF1-4-2010.pdf

Yiwei Jiang and Yong He. 2007. Optimal semi-online algorithms for preemptive scheduling problems with inexact partial information. *Acta Inform.* 44, 7–8 (2007), 571–590. DOI:http://dx.doi.org/10.1007/s00236-007-0058-8 Preliminary version in ISAAC'05.

Bala Kalyanasundaram and Kirk R. Pruhs. 1994. Constructing competitive tours from local information. *Theor. Comput. Sci.* 130, 1 (1994), 125–138. DOI:http://dx.doi.org/10.1016/0304-3975(94)90155-4 Preliminary version in ICALP'93.

Shahin Kamali and Alejandro López-Ortiz. 2014a. Almost online square packing. In *The Canadian Conference on Computational Geometry*. www.cccg.ca, 162–168. Retrieved from http://www.cccg.ca/proceedings/2014/papers/paper24.pdf.

Shahin Kamali and Alejandro López-Ortiz. 2014b. Better compression through better list update algorithms. In *Data Compression Conference*. IEEE Computer Society Press, 372–381. DOI:http://dx.doi.org/10.1109/DCC.2014.86

Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. 1988. Competitive snoopy caching. *Algorithmica* 3 (1988), 77–119. DOI:http://dx.doi.org/10.1007/BF01762111 Preliminary version in FOCS'86.

Lucia Keller. 2014. *Complexity of Optimization Problems, Advice and Approximation*. Ph.D. Dissertation. ETH Zürich. DOI:http://dx.doi.org/10.3929/ethz-a-010143463

Hans Kellerer, Vladimir Kotov, and Michaël Gabay. 2015. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *J. Sched.* 18, 6 (2015), 623–630. DOI:http://dx.doi.org/10.1007/s10951-015-0430-4

Hal A. Kierstead. 1998. Coloring graphs online. In *Online Algorithms—The State of the Art*. Springer, 281–305. DOI:http://dx.doi.org/10.1007/BFb0029574

Hal A. Kierstead and William T. Trotter. 1991. On-line graph coloring. In *On-Line Algorithms (DIMACS Series in Discrete Mathematics and Theoretical Computer Science)*, Vol. 7. DIMACS/AMS, 85–92.

Dennis Komm and Richard Královič. 2011. Advice complexity and barely random algorithms. *RAIRO Theor. Inf. Appl.* 45, 2 (2011), 249–267. DOI:http://dx.doi.org/10.1051/ita/2011105 Preliminary version in SOFSEM'11.

Dennis Komm, Rastislav Královič, Richard Královič, and Christian Kudahl. 2016. Advice complexity of the online induced subgraph problem. In *MFCS (LIPIcs)*, Vol. 58. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 59:1–59:13. DOI:http://dx.doi.org/10.4230/LIPIcs.MFCS.2016.59

Dennis Komm, Rastislav Královič, Richard Královič, and Jasmin Smula. 2015. Treasure hunt with advice. In *SIROCCO (LNCS)*, Vol. 9439. Springer, 328–341. DOI:http://dx.doi.org/10.1007/978-3-319-25258-2_23

Dennis Komm, Richard Královič, and Tobias Mömke. 2012. On the advice complexity of the set cover problem. In *CSR (LNCS)*, Vol. 7353. Springer, 241–252. DOI:http://dx.doi.org/10.1007/978-3-642-30642-6_23

Elias Koutsoupias. 2009. The *k*-server problem. *Comput. Sci. Rev.* 3, 2 (2009), 105–118. DOI:http://dx.doi.org/10.1016/j.cosrev.2009.04.002

Elias Koutsoupias and Christos H. Papadimitriou. 1995. On the *k*-server conjecture. *J. ACM* 42, 5 (1995), 971–983. DOI:http://dx.doi.org/10.1145/210118.210128 Preliminary version in STOC'94.

Rastislav Královič. 2014. Advice complexity: Quantitative approach to A-priori information. In *SOFSEM (LNCS)*, Vol. 8327. Springer, 21–29. DOI:http://dx.doi.org/10.1007/978-3-319-04298-5_3

Sacha Krug. 2015. Towards using the history in online computation with advice. *RAIRO Theor. Inf. Appl.* 49, 2 (2015), 139–152. DOI:http://dx.doi.org/10.1051/ita/2015003

Alberto Marchetti-Spaccamela and Carlo Vercellis. 1995. Stochastic on-line knapsack problems. *Math. Program.* 68 (1995), 73–104. DOI:http://dx.doi.org/10.1007/BF01585758

Lyle A. McGeoch and Daniel D. Sleator. 1991. A strongly competitive randomized paging algorithm. *Algorithmica* 6 (1991), 816–825. DOI:http://dx.doi.org/10.1007/BF01759073

Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. 2012. Online graph exploration: New results on old and new algorithms. *Theor. Comput. Sci.* 463 (2012), 62–72. DOI:http://dx.doi.org/10.1016/j.tcs.2012.06.034 Preliminary version in ICALP'11.

Jesper W. Mikkelsen. 2015. Optimal online edge coloring of planar graphs with advice. In *CIAC (LNCS)*, Vol. 9079. Springer, 352–364. DOI:http://dx.doi.org/10.1007/978-3-319-18173-8_26

Jesper W. Mikkelsen. 2016. Randomization can be as helpful as a glimpse of the future in online computation. In *ICALP (LIPIcs)*, Vol. 55. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 39:1–39:14. DOI:http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.39

Shuichi Miyazaki. 2014. On the advice complexity of online bipartite matching and online stable marriage. *Inform. Process. Lett.* 114, 12 (2014), 714–717. DOI:http://dx.doi.org/10.1016/j.ipl.2014.06.013

Marc Renault. 2014. *Lower and Upper Bounds for Online Algorithms with Advice*. Ph.D. Dissertation. Université Paris Diderot–Paris 7. Retrieved from https://www.irif.univ-paris-diderot.fr/mrenault/papers/renaultPhD.pdf.

Marc P. Renault. 2016. Online algorithms with advice for the dual bin packing problem. *Cent. Eur. J. Oper. Res.* (2016). To appear. DOI:http://dx.doi.org/10.1007/s10100-016-0450-y

Marc P. Renault and Adi Rosén. 2015. On online algorithms with advice for the *k*-server problem. *Theor. Comput. Syst.* 56, 1 (2015), 3–21. DOI:http://dx.doi.org/10.1007/s00224-012-9434-z Preliminary version in WAOA'11.

Marc P. Renault, Adi Rosén, and Rob van Stee. 2015. Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.* 600 (2015), 155–170. DOI:http://dx.doi.org/10.1016/j.tcs.2015.07.050

John F. Rudin, III. 2001. *Improved Bounds for the On-Line Scheduling Problem*. Ph.D. Dissertation. University of Texas at Dallas.

Sebastian Seibert, Andreas Sprock, and Walter Unger. 2013. Advice complexity of the online coloring problem. In *CIAC (LNCS)*, Vol. 7878. Springer, 345–357. DOI:http://dx.doi.org/10.1007/978-3-642-38233-8_29

Steven S. Seiden, Jirí Sgall, and Gerhard J. Woeginger. 2000. Semi-online scheduling with decreasing job sizes. *Oper. Res. Lett.* 27, 5 (2000), 215–221. DOI:http://dx.doi.org/10.1016/S0167-6377(00)00053-5

Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (1985), 202–208. DOI:http://dx.doi.org/10.1145/2786.2793 Preliminary version in STOC'84.

Jasmin Smula. 2015. *Information Content of Online Problems, Advice versus Determinism and Randomization*. Ph.D. Dissertation. ETH Zürich. DOI:http://dx.doi.org/10.3929/ethz-a-010497710

Björn Christian Steffen. 2014. *Advice Complexity of Online Graph Problems*. Ph.D. Dissertation. ETH Zürich. DOI:http://dx.doi.org/10.3929/ethz-a-010185054

Zhiyi Tan and Yong Wu. 2007. Optimal semi-online algorithms for machine covering. *Theor. Comput. Sci.* 372, 1 (2007), 69–80. DOI:http://dx.doi.org/10.1016/j.tcs.2006.11.015

Vijay V. Vazirani. 2003. *Approximation Algorithms*. Springer. DOI:http://dx.doi.org/10.1007/978-3-662-04565-7

Vadim G. Vizing. 1965. Critical graphs with given chromatic class. *Metody Diskret. Analiz.* 5 (1965), 9–17. In Russian.

David Wehner. 2014. A new concept in advice complexity of job shop scheduling. In *MEMICS (LNCS)*, Vol. 8934. Springer, 147–158. DOI:http://dx.doi.org/10.1007/978-3-319-14896-0_13

David Wehner. 2015. Advice complexity of fine-grained job shop scheduling. In *CIAC (LNCS)*, Vol. 9079. Springer, 416–428. DOI:http://dx.doi.org/10.1007/978-3-319-18173-8_31

Gerhard J. Woeginger. 1997. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.* 20, 4 (1997), 149–154. DOI:http://dx.doi.org/10.1016/S0167-6377(96)00055-7

Jinjiang Yuan, C. T. Ng, and T. C. E. Cheng. 2011. Best semi-online algorithms for unbounded parallel batch scheduling. *Discr. Appl. Math.* 159 (2011), 838–847. DOI:http://dx.doi.org/10.1016/j.dam.2011.01.003

Xiaofan Zhao and Hong Shen. 2014. On the advice complexity of one-dimensional online bin packing. In *Frontiers in Algorithmics (LNCS)*, Vol. 8497. Springer, 320–329. DOI:http://dx.doi.org/10.1007/978-3-319-08016-1_29

Received September 2016; revised November 2016; accepted November 2016