On the Advice Complexity of Online Problems[*]
(Extended Abstract)

Hans-Joachim Böckenhauer[1], Dennis Komm[1], Rastislav Královič[2], Richard Královič[1], and Tobias Mömke[1]

[1] Department of Computer Science, ETH Zurich, Switzerland
{hjb, dennis.komm, richard.kralovic, tobias.moemke}@inf.ethz.ch
[2] Department of Computer Science, Comenius University, Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk

Abstract. In this paper, we investigate to what extent the solution quality of online algorithms can be improved by allowing the algorithm to extract a given amount of information about the input. We consider the recently introduced notion of *advice complexity* where the algorithm, in addition to being fed the requests one by one, has access to a tape of advice bits that were computed by some *oracle* function from the complete input. The advice complexity is the number of advice bits read. We introduce an improved model of advice complexity and investigate the connections of advice complexity to the competitive ratio of both deterministic and randomized online algorithms using the paging problem, job shop scheduling, and the routing problem on a line as sample problems. We provide both upper and lower bounds on the advice complexity of all three problems.
Our results for all of these problems show that very small advice (only three bits in the case of paging) already suffices to significantly improve over the best deterministic algorithm. Moreover, to achieve the same competitive ratio as any randomized online algorithm, a logarithmic number of advice bits is sufficient. On the other hand, to obtain optimality, much larger advice is necessary.

1 Introduction

Many problems such as routing, scheduling, or the paging problem work in so-called online environments and their algorithmic formulation and analysis demand a model in which an algorithm that deals with such a problem knows only a part of its input at any specific point during runtime. These problems are called *online problems* and the respective algorithms are called *online algorithms*. In such an online setting, an online algorithm A has a huge disadvantage compared to offline algorithms (i. e., algorithms knowing the whole input already at the beginning of their computation) since A has to make decisions at any time step i without knowing what the next chunk of input at time step $i+1$ will be.

[*] This work was partially supported by ETH grant TH 18 07-3, and APVV grant 0433-06.

As `A` has to produce a part of the final output in every step, it cannot revoke decisions it has already made. These decisions can only be made by merely taking input chunks from time steps 1 to i into account and maybe applying some randomization.

The output quality of such an online algorithm is often analyzed by the well-established *competitive ratio* introduced by Sleator and Tarjan in [9]. Informally speaking, the output quality of an online algorithm `A` is measured by comparing it to an optimal offline algorithm. For an online problem P, let $\texttt{Opt}(I)$ denote an optimal offline solution for a certain input I of P. By $\mathcal{A} = \texttt{A}(I)$ we denote the solution (i. e., the sequence of answers) computed by `A` on I, and we denote its *cost* (or, for maximization problems, *gain*) as $C(\texttt{A}(I))$. Then, for some $c \geq 0$, `A` is called *c-competitive* if there exists some constant $\alpha \geq 0$ such that, for any such input sequence I, $C(\texttt{A}(I)) \leq c \cdot C(\texttt{Opt}(I)) + \alpha$, and it is called *strictly c-competitive*, if $\alpha = 0$. An online algorithm is *optimal* if it is 1-competitive with $\alpha = 0$. As a rather powerful tool, randomness is often employed in the design of online algorithms. The computations (sometimes also called *decisions*) of a randomized online algorithm `R` hereby heavily depend on a sequence of random bits often viewed as the content of a random tape accessible by `R`. For a fixed content of the random tape, `R` then behaves deterministically and achieves a certain competitive ratio. The expected value of the competitive ratio over all possible contents of the random tape is then used to measure the quality of a randomized online algorithm.

On the downside, it seems rather unfair to compare online and offline algorithms since there is simply no reasonable application of an offline algorithm in an online environment. Therefore, offline algorithms are in general more powerful than online algorithms. Hence, we are interested not only in comparing the output quality of `A` to that of an optimal offline algorithm `B`, but we want to investigate what amount of information `A` really lacks. Surprisingly (and as already discussed in [6]), there are problems where only one straightforward piece of information (i. e., one bit) is needed for allowing `A` to be as good as `B`, e. g., the problem SkiRental, see [2, 6]. Clearly, this does not hold in general and thus we are interested in a formal framework allowing us to classify online problems according to how much information about the future input is needed for solving them optimally or with a specific competitive ratio. One way of measuring the amount of information needed for an online algorithm to be optimal is called *advice complexity* and was proposed in [6]. This model of advice complexity can be viewed as a cooperation of a deterministic online algorithm `A` and an *oracle* `O`, which may passively communicate with `A`. The oracle `O`, which has unlimited computational power, sees the whole input of `A` in advance and writes bitwise information needed by `A` onto an *advice tape* before `A` reads any input. Then, `A` can access the bits from the advice tape in a sequential manner, just as a randomized algorithm would use its random tape. The *advice complexity* of `A` on an input I is now defined as the number of advice bits `A` reads while processing this input. As usual, we consider the advice complexity as a function of the input size n by taking the maximum value over all inputs of length at most n.

Note that, by our definition, the oracle has no possibility to explicitly indicate the end of the advice. This eliminates the ability of the oracle to encode some information into the length of the advice string, as it was the case in [6]. As a result, our model is cleaner and more consistent with other complexity measures like, e. g., the number of random bits required for randomized algorithms. Besides asking for the amount of advice that is necessary to compute an optimal solution, we also deal with the question whether some small advice might help to significantly reduce the competitive ratio.

A similar model to that of [6] has been very recently used in [7]. There, the oracle can send an advice message of fixed size to the online algorithm together with every request. This model, however, is suitable only for online problems that cannot be solved efficiently with small advice per request, as it is the case for the problems considered in [7] (metrical task systems, k-server problem). Since all three problems we consider here can be easily solved with 1 advice bit per request, this model is not applicable in our case.

After providing the formal definition of our model and some general observations in Section 2, we analyze the advice complexity of three different online problems: *paging* (in Section 3), *disjoint path allocation* (in Section 4), and *job shop scheduling* (in Section 5). For all three problems, we provide both upper and lower bounds on the trade-off between the advice complexity and the competitive ratio. Our results exhibit a somewhat similar behavior for all three problems: While large advice is necessary to achieve an optimal solution, significantly smaller advice suffices to be on par with the best randomized algorithm. E. g., an optimal solution for paging requires 1 bit per request, but with $\mathcal{O}(\log k)$ bits (where k is the size of the buffer) it is possible to achieve ratio asymptotically equal to the best known randomized algorithm. Furthermore, it is usually the case that very small advice suffices to significantly improve the competitive ratio over the best deterministic algorithm. E. g., two bits of advice for whole input are sufficient to improve the ratio from k to $k/2 + \mathcal{O}(1)$ for paging.

Due to space limitations, this extended abstract does not contain all proofs. All details missing in this paper can be found in the technical report [3].

2 Preliminaries

Often, randomization is used in the design of online algorithms. Formally, randomized online algorithms can compute the answers from the previous requests, as well as from the content of a *random tape* ϕ, i. e., an infinite binary sequence where every bit is chosen uniformly and independently at random. By $C(\mathtt{R}(I))$, we denote the random variable expressing the cost of the solution computed by $\mathtt{R}$ on I. $\mathtt{R}$ is c-competitive (against an oblivious adversary) if there exists a constant α such that, for each input sequence I, the expected value $E[C(\mathtt{R}(I))] \leq c \cdot C(\mathtt{Opt}(I)) + \alpha$. Our work focusses on a model where the algorithm can use some information, called *advice*, about the future input.

Definition 1. *An* online algorithm $\mathtt{A}$ with advice *computes the output sequence* $\mathcal{A}^{\phi} = \mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where*

ϕ is the content of the advice tape, i. e., an infinite binary sequence, and $I = (x_1, \ldots, x_n)$. Algorithm `A` *is* c-competitive with advice complexity *$s(n)$ if there exists a constant α such that, for every n and for each input sequence I of length at most n, there exists some ϕ such that $C(\mathtt{A}^{\phi}(I)) \leq c \cdot C(\mathtt{Opt}(I)) + \alpha$ and at most $s(n)$ bits of ϕ have been accessed during the computation of $\mathtt{A}^{\phi}(I)$. If $\alpha = 0$, then* `A` *is* strictly c-competitive *with advice complexity $s(n)$.*

For the ease of notation, for both randomized online algorithms and online algorithms with advice, if ϕ is clear from the context, we write $\mathtt{A}(I)$ instead of $\mathtt{A}^{\phi}(I)$. If `A` accesses b bits of the advice tape during some computation, we say that b *advice bits are communicated* to `A`, or that `A` *uses b bits of advice*. The advice complexity of `A` gives an upper bound on the number of communicated advice bits, depending on the size n of the input. Note that, if some randomized online algorithm achieves a competitive ratio of r using b random bits, the same competitive ratio r can be achieved by an online algorithm with advice using b advice bits. We use $\log(x)$ to denote the logarithm of x with base two.

For proving upper bounds, we sometimes use the following idea. The oracle needs to communicate some n-bit string to the algorithm, but this string always contains only few ones or few zeros, allowing for the following efficient encoding.

Lemma 1. *Consider an online algorithm* `A` *with advice, achieving competitive ratio r while using some n-bit advice string that contains at most n/t zeros or at least $n - n/t$ zeros, where $t \geq 2$ is a fixed constant. Then, it is possible to design an improved online algorithm* `B` *that knows the parameter t and achieves an advice complexity of*

$$s(n) = \min\left\{ n \log\left(t/(t-1)^{\frac{t-1}{t}} \right), \frac{n \log n}{t} \right\} + 3 \log n + \mathcal{O}(1).$$

3 Paging

To use computer memory as efficiently as possible, the well-known technique of *paging* is widely used. Formally, we define paging problem as follows.

Definition 2 (Paging Problem). *The input is a sequence of integers representing requests to logical pages $I = (x_1, \ldots, x_n)$, $x_i > 0$, $x_i \neq x_{i+1}$. An online algorithm* `A` *maintains a buffer (content of the physical memory) $B = \{b_1, \ldots, b_K\}$ of K integers, where K is a fixed constant known to* `A`*. Before processing the first request, the buffer gets initialized as $B = \{1, \ldots, K\}$. Upon receiving a request x_i, if $x_i \in B$, then $y_i = 0$. If $x_i \notin B$, then a* page fault *occurs, and the algorithm has to find some* victim *b_j, i. e., $B := B \setminus \{b_j\} \cup \{x_i\}$, and $y_i = b_j$. The cost of the solution $\mathcal{A} = \mathtt{A}(I)$ is the number of page faults, i. e., $C(\mathcal{A}) = |\{y_i : y_i > 0\}|$.*

The paging problem (or Paging for short) has been extensively studied (see [2]). It is known that every deterministic algorithm is at least K-competitive, and there exist K-competitive deterministic algorithms. A simple upper bound on the advice complexity of an optimal algorithm follows from [6, Lemma 3.1].

Fact 1 *For* Paging*, there is an optimal online algorithm with advice complexity $n + K$ and a 1-competitive online algorithm with advice complexity n.*

A lower bound for the competitive ratio of any randomized paging algorithm was proven in [2]: every randomized algorithm is at least H_K-competitive, where H_K is the K-th harmonic number, i. e., $H_K = \sum_{i=1}^{K} \frac{1}{i}$. The lower bound is almost tightly matched by randomized *marking algorithm*[3]: the randomized marking algorithm is H_K-competitive, if the logical pages are chosen from a set of cardinality $K + 1$, and they are $2H_K$-competitive in general.

Constant Competitive Ratio. From Fact 1, we can see that using one bit of advice per request is sufficient to obtain an optimal paging algorithm. In the next theorem, we show that it is possible to obtain paging algorithms with good competitive ratios using smaller advice. More precisely, we show an upper bound on the advice complexity depending on the competitive ratio, indicating that using an amortized constant amount of bits per request $s(n)/n < 1$ is enough to achieve a constant competitive ratio.

Theorem 1. *For each constant $r \geq 1$, there exists an r-competitive algorithm with advice complexity $s(n) = n \log\left((r+1)/\left(r^{r/(r+1)}\right)\right) + 3 \log n + \mathcal{O}(1)$.*

The proof of Theorem 1 is presented in [3]. As a corollary, note that it is possible to obtain a constant competitive ratio with an amortized constant number of bits of advice per request. This number of bits can be arbitrarily close to 0 at the expense of a very high competitive ratio; on the other hand, a competitive ratio arbitrarily close to 1 is reachable with the number of bits per request approaching 1. More precisely, for each constant $r \geq 1$, there exists an r-competitive algorithm with advice complexity $s(n)$ such that $\lim_{n\to\infty}(s(n)/n) = \log\left((r+1)/\left(r^{r/(r+1)}\right)\right)$.

Next, we show a lower bound on the advice complexity required to obtain a constant competitive ratio, more precisely, we express the minimal number of advice bits per request required to obtain an r-competitive algorithm.

Theorem 2. *Let r be any constant such that $1 \leq r \leq 1.25$. For any paging algorithm* `A` *with advice complexity $s(n)$ and competitive ratio r it holds that $\frac{s(n)}{n} \geq \frac{1}{2K-2}\left[1 + \log(3-2r) - (2r-2)\log\left(\frac{1}{2r-2} - 1\right)\right] - \mathcal{O}\left(\frac{1}{n}\right)$. The constant of the $\mathcal{O}(1/n)$ depends on K and the parameters r, α of* `A`.

The core idea of the proof of Theorem 2 is to consider a certain (sufficiently large) class of inputs arranged into a complete tree. If the advice is small enough, there must be many inputs processed with the same advice. This makes it possible to prove that `A` must behave inefficiently on at least one of them, by exploiting the tree structure of the inputs. For a detailed proof, see [3].

[3] In general, a marking algorithm (see [2]) starts with all pages unmarked, and upon a hit marks the requested page. If there is no unmarked page upon a page fault, all pages in the memory are unmarked and a new phase begins.

To obtain an optimal algorithm, large advice is necessary. Similar arguments as in Theorem 2 yield a lower bound on the advice complexity of $\frac{s(n)}{n} \geq 1 - \frac{\log(K-1)+C}{4(K-1)} - \mathcal{O}\left(\frac{1}{n}\right)$, for some constant C, converging to 1 bit per request with growing cache size K. Thus, the upper bound from Fact 1 is tight for large K.

Small advice. We now analyze the case where the algorithm is allowed to use only a constant number of advice bits for the whole input. We show that, even in this very restricted case, the competitive ratio can be significantly improved with respect to the best deterministic algorithm. We start with upper bounds on the advice complexity, and complement them later with a lower bound.

Theorem 3. *Consider the class of inputs with a buffer of size K, and let $c < K$ be a power of 2. There is an algorithm with oracle size $\log c$ for* Paging *on this class of inputs with competitive ratio $r \leq 3\log c + (2(K+1)/c) + 1$.*

For the proof, we construct c deterministic marking algorithms. Their executions are naturally divided into phases, and the construction makes sure that, in each phase, the algorithms induce many different behaviors. A careful choice of page replacement strategies enables to use an accounting scheme to bound the overall number of faults occurring in all algorithms during a phase. The proof is finished by arguing that there must be an algorithm with at most $1/c$-th fraction of them, and comparing this amount with a trivial general lower bound. A detailed proof can be found in [3].

The presented results show that even very small advice can be used efficiently. For example, providing just two bits of advice per whole input instance reduces the competitive ratio from K (i. e., the best deterministically achievable ratio) to $K/2 + \mathcal{O}(1)$. Additionally, $\log K$ bits of advice (i. e., making c equal to the largest power of 2 smaller than K) can be used to achieve the competitive ratio $3\log K + \mathcal{O}(1)$ which is asymptotically equal (albeit the constant is worse) to the best possible ratio of $H_K = \sum_{i=1}^{K} \frac{1}{i}$ for a randomized algorithm without advice.

Next, we complement the result of Theorem 3 by presenting a lower bound on the competitive ratio for paging algorithms with constant advice complexity.

Theorem 4. *Consider the class of inputs with $K + 1$ possible pages, where K is the size of the buffer, and let c be a power of 2. Any deterministic algorithm* `A` *with advice complexity $s(n) = \log c$ has competitive ratio at least K/c.*

The main proof idea is to consider all inputs of certain length arranged into a complete K-ary tree. Then, it is either possible to find an input with many faults in the beginning of the computation, or to select an input prefix such that some advice is not used for any extension of this prefix, and to continue by induction.

4 Disjoint Path Allocation

We consider a special type of network topology where the entities of a network are connected by one line (i. e., a bus network). The network is a path P and all

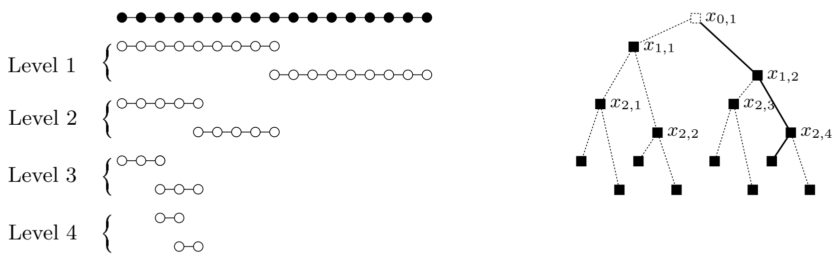

Fig. 1: A sample input from the class $\mathcal{I}_3$ and its representation as tree $\mathcal{T}$ of height h

connections in P have a capacity of one. For the *disjoint path allocation problem* (DisPathAlloc, described in [2]), additionally a set of subpaths of P is given. Each subpath $(v_i, \ldots, v_j)$ models a *call request*, i. e., a request to establish a permanent connection between the two endpoints v_i and v_j. If such a request is satisfied, no inner entity of the path is able to be part of any other call. Therefore, a *disjoint path allocation* is simply a set of edge-disjoint subpaths of P.

Definition 3 (Disjoint Path Allocation Problem). *Given a path $P = (V, E)$, where $V = \{v_0, \ldots, v_L\}$ is a set of entities, and a set $\mathcal{P}$ of subpaths of P where $|\mathcal{P}| = n$,* DisPathAlloc *is the problem of finding a maximum set $\mathcal{P}' \subseteq \mathcal{P}$ of edge-disjoint subpaths of P.*

Here, we deal with an environment in which the subpaths $P_1, \ldots, P_n \in \mathcal{P}$ arrive in an online fashion. We assume that $L = |V| - 1$ is known to the online algorithm in advance, but that this is not the case for n. For the ease of presentation, let (v_i, v_j) denote the path $(v_i, v_{i+1}, \ldots, v_j)$. The cost function $C(\mathtt{A}(I))$ denotes the number of requests satisfied by $\mathtt{A}$ on input I. Since DisPathAlloc is a maximization problem, an algorithm $\mathtt{A}$ solving it is c-competitive if $C(\mathtt{Opt}(I)) \leq c \cdot C(\mathtt{A}(I)) + \alpha$ for some constant α and every input I and strictly c-competitive if $\alpha = 0$. The competitive ratio can be measured in terms of n as well as in terms of L. We focus on the former case, the latter is discussed in [3]. It is not difficult to check that any deterministic online algorithm $\mathtt{A}$ is no better than strictly $(n-1)$-competitive and strictly L-competitive. These bounds are tight, since they are reached by a simple greedy algorithm.

The Class $\mathcal{I}$ of Inputs. For some of the following proofs, we consider input instances as depicted in Fig. 1 for any even n. For every h, we define the class $\mathcal{I}_h$ (and $\mathcal{I} = \bigcup_{h \in \mathbb{N}} \mathcal{I}_h$) as follows. Every element of $\mathcal{I}_h$ consists of $h + 1$ *levels*. Level 1 consists of two disjoint consecutive requests, splitting the line network into two parts of the same size. After that, two disjoint consecutive requests on level $i+1$ do the same with one of the intervals from level i. This is iterated until two requests of size 1 appear on level $h + 1$. Fig. 1, for instance, shows an input from $\mathcal{I}_3$. It is obvious that any optimal algorithm satisfies exactly one request on each of the first h levels, allowing it to satisfy both requests on level $h + 1$.

Clearly, we may represent an optimal solution, for any input sequence as described above, by a path from the root to a leaf in a complete binary tree $\mathcal{T}$

of height h with its root on a notional level 0 (see Fig. 1). The 2^h leaves of $\mathcal{T}$ represent the 2^h different inputs of the class $\mathcal{I}_h$. Let $\mathcal{OPT}$ denote a path corresponding to some input instance $\in \mathcal{I}_h$. We may say that an optimal algorithm $\texttt{Opt}$ makes moves *according to this path.* For an arbitrary online algorithm $\texttt{A}$ that satisfies one request on level i, we say that $\texttt{A}$ makes the *correct* decision on this level if it also acts according to $\mathcal{OPT}$. However, if $\texttt{A}$ satisfies one interval not according to this path $\mathcal{OPT}$ or satisfies both requests, we say that $\texttt{A}$ makes a *wrong* decision on level i. In this case, we say that $\texttt{A}$ is *out* (after level i). Moreover, for every i and an input instance $I_h \in \mathcal{I}_h$, let $C_i(\texttt{A}(I_h))$ denote the overall number of requests satisfied by $\texttt{A}$ up to level i. If $\texttt{A}$ is out after level i, this means that, for every $j \geq i$, $C_j(\texttt{A}(I_h)) = C_i(\texttt{A}(I_h))$ (and obviously, therefore, $C(\texttt{A}(I_h)) = C_i(\texttt{A}(I_h))$). Let $correct(\texttt{A})$ [$wrong(\texttt{A})$] denote the set of time steps in which $\texttt{A}$ makes the correct [wrong] decision. Since making the wrong decision can only happen once (because the algorithm is out afterwards), the overall gain of $\texttt{A}$ is $C(\texttt{A}(I_h)) \leq |correct(\texttt{A})| + 2 \cdot |wrong(\texttt{A})| \leq |correct(\texttt{A})| + 2$, which directly implies that, for an optimal algorithm $\texttt{Opt}$, $C(\texttt{Opt}(I_h)) = h + 2$.

Advice Complexity Bounds. DisPathAlloc is a hard online problem even for randomized algorithms. A lower bound of $\Omega(n)$ on the competitive ratio has been proven in [5, Theorem 9] and [3] independently. We now investigate how many advice bits are needed for an online algorithm $\texttt{A}$ to yield optimality.

Theorem 5. *For any online algorithm with advice for* DisPathAlloc*, at least* $\frac{n+2}{2r} - 2$ *bits of advice are required to achieve a strict competitive ratio of* r.

Note that, by setting $r = 1$, we immediately get a lower bound of $b_{opt} = \frac{n-2}{2}$ advice bits for achieving an optimal solution. On the other hand, it is not difficult to construct an online algorithm with advice for DisPathAlloc whose advice complexity is only a factor of $\log n$ away from this lower bound for large r, and is asymptotically tight for constant r: Consider an algorithm $\texttt{A}$ that reads one bit of advice per request, and satisfies the request if and only if this bit is one. Communicating an advice string with at most n/r ones is sufficient to achieve a competitive ratio r. Hence, Lemma 1 for $t := r$ implies the following theorem.

Theorem 6. *For every r, there exists an r-competitive online algorithm for* DisPathAlloc *with advice complexity*

$$s(n) = \min\left\{n \log\left(r/(r-1)^{(r-1)/r}\right), (n \log n)/r\right\} + 3 \log n + \mathcal{O}(1).$$

The previous theorem also shows that advice complexity $\mathcal{O}(\log^2 n)$ is sufficient to obtain a competitive ratio asymptotically better than any randomized algorithm is able to achieve.

5 Job Shop Scheduling

We consider a special case of job shop scheduling with two jobs consisting of n unit-length tasks each, which have to be processed on the same n machines, each task on a different one, but possibly in a different order within the two jobs. Two tasks requiring the same machine in one time step causes a delay in the schedule and the goal in this problem JSSchedule is to minimize the number of these delays. This problem can be modelled as finding a shortest path in an $(n \times n)$-grid from the upper left to the lower right corner, where also diagonal steps are allowed, except for some grid cells which are blocked by obstacles modelling tasks competing for the same machine. For the details of the problem description, we refer to [3, 8]. Full proofs of the results from this section can be found in [3].

Lemma 2. *For every instance of* JSSchedule*, there exists an optimal solution $\mathcal{A}$ which, as long as it does not arrive at the right or bottom border of the grid, always moves diagonally if no obstacle is in its way.*

Consider the optimal solution for an arbitrary instance, and assume that it makes d diagonal moves, h moves to the right, and $v = h$ moves downwards. The cost of this solution is $d + 2h = n + h$, since $d + h = n$. According to [8], for every instance of JSSchedule, there exists an optimal solution with cost at most $n + \lceil \sqrt{n} \rceil$ and therefore $h \leq \lceil \sqrt{n} \rceil$. Furthermore, using Lemma 2, it is easy to see that there exists an online algorithm $\mathtt{A}$ with advice needing at most $2h \leq 2\lceil \sqrt{n} \rceil$ advice bits: Algorithm $\mathtt{A}$ simply acts according to Lemma 2. Only at time steps where a diagonal move is not possible due to an obstacle, it has to read one bit from the advice tape indicating whether to move downwards or to the right. As a corollary, for every instance of JSSchedule, there exists an optimal online algorithm $\mathtt{A}$ with advice complexity $s(n) = 2\lceil \sqrt{n} \rceil$. On the other hand, this upper bound is asymptotically tight, as claimed by the following lower bound on the number of advice bits necessary to compute an optimal solution.

Theorem 7. *Any online algorithm $\mathtt{A}$ with advice for* JSSchedule *needs advice complexity $s(n) = \Omega(\sqrt{n})$ to achieve optimality.*

While an optimal solution of JSSchedule requires large advice, much shorter advice is sufficient to achieve a solution with competitive ratio close to 1. In [8], a randomized algorithm for JSSchedule with a competitive ratio of $\mathcal{O}(1 + 1/\sqrt{n})$ is presented which easily implies that there is an online algorithm for JSSchedule with advice complexity $s(n) \leq 1 + \log(n)$ that achieves competitive ratio $\mathcal{O}(1 + 1/\sqrt{n})$.

6 Conclusion

Our results suggest the hypothesis that logarithmic advice complexity is sufficient to achieve the competitive ratio of the best randomized algorithm for *any* online problem. This claim, however, cannot be proven in full generality. In fact, we can construct an online problem where the number of required advice bits is as high as the number of random bits in order to keep up with randomized algorithms.

Consider the following problem, where the online algorithm has to guess a sequence of n bits. If all bits are guessed correctly, the algorithm is greatly rewarded, otherwise, the cost of the solution is zero. For this problem, we can construct a very simple randomized algorithm that has a positive number as expected gain whereas it is easy to show that any deterministic algorithm needs a linear number of advice bits for having a gain greater than zero. Unfortunately, this situation is quite artificial: if an algorithm with advice `A` outperforms a randomized algorithm `R`, then `A` is exponentially better than `R`, too. Nevertheless, this situation shows that, in general, the advice complexity is somehow orthogonal to randomization. It remains as an open problem to find more connections between these complexity measures, e. g., to somehow characterize classes of online problems where small advice is sufficient to keep up with randomized algorithms. Furthermore, it might be interesting to consider randomized online algorithms with advice.

Acknowledgments

The authors would like to thank Juraj Hromkovič for many helpful discussions on the topic of this paper and elaborating the model of advice complexity of online problems to the point we could start investigating the problems introduced.

References

1. S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.
2. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, New York, 1998.
3. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, T. Mömke. Online Algorithms with Advice. *Technical Report 614.* ETH Zurich, Department of Computer Science, 2009.
4. P. Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40(4):353–359, 1988.
5. I. Caragiannis, A. V. Fishkin, C. Kaklamanis, and E. Papaioannou. Randomized on-line algorithms and lower bounds for computing large independent sets in disk graphs. *Discrete Applied Mathematics*, 155(2):1190–136, 2007.
6. S. Dobrev, R. Královič, and D. Pardubská. Measuring the Problem-Relevant Information in Input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009.
7. Y. Emek, P. Fraigniaud, A. Korman, A. Rosén. Online Computation with Advice In *Automata, Languages and Programming, 36th International Colloquium (ICALP 2009)*, LNCS 5555, pp. 427–438.
8. J. Hromkovič, T. Mömke, K. Steinhöfel, P. Widmayer. Job shop scheduling with unit length tasks: bounds and algorithms. *Algorithmic Operations Research*, 2(1):1–14, 2007.
9. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.