RAIRO-Theor. Inf. Appl. 45 (2011) 249–267
DOI: 10.1051/ita/2011105

Available online at:
www.rairo-ita.org

ADVICE COMPLEXITY AND BARELY RANDOM ALGORITHMS *

Dennis Komm[1] and Richard Královič[1]

Abstract. Recently, a new measurement – the *advice complexity* – was introduced for measuring the information content of online problems. The aim is to measure the bitwise information that online algorithms lack, causing them to perform worse than offline algorithms. Among a large number of problems, a well-known scheduling problem, *job shop scheduling with unit length tasks*, and the *paging* problem were analyzed within this model. We observe some connections between advice complexity and randomization. Our special focus goes to barely random algorithms, *i.e.*, randomized algorithms that use only a constant number of random bits, regardless of the input size. We adapt the results on advice complexity to obtain efficient barely random algorithms for both the job shop scheduling and the paging problem. Furthermore, so far, it has not yet been investigated for job shop scheduling how good an online algorithm may perform when only using a very small (*e.g.*, constant) number of advice bits. In this paper, we answer this question by giving both lower and upper bounds, and also improve the best known upper bound for optimal algorithms.

Mathematics Subject Classification. 68Q25, 68Q30, 68Q87.

1. Introduction

In classical algorithmics, one is interested in designing fast algorithms that create high-quality solutions for a large set of instances of specific problems. Moreover, in many practical applications, another challenge arises for the algorithm designer: often, not the whole input is known at once, but it arrives piecewise in

Keywords and phrases. Barely random algorithms, advice complexity, information content, online problems.

* *This work was partially supported by ETH grant TH 18 07-3 and SNF grant 200020-120073.* An extended abstract of this paper appeared in [11].

[1] Department of Computer Science, ETH Zurich, Switzerland. dennis.komm@inf.ethz.ch

consecutive time steps. After every such time step, a piece of output has to be created which must not be changed afterwards, *i.e.*, the algorithm has to compute the output without knowing the whole input. We call such situations *online scenarios* and the according strategies to cope with them *online algorithms*. We do not give a detailed introduction, but point the reader to the standard literature, *e.g.*, [4,8].

Classically, the output quality of an online algorithm is measured by the *competitive ratio* [4,8], *i.e.*, the quotient of the cost of the solution the online algorithm computes for a particular problem instance and the cost of an optimal (offline) solution for this instance.

Here, we are dealing with online algorithms that have access to an additional *advice tape* thought of as being written by an oracle $\mathtt{O}$ that sees the whole input in advance and has unlimited computational power. The motivation behind this setup is that $\mathtt{O}$ can give some information about the future parts of the input to the algorithm. This allows us to measure the amount of such information that is necessary/sufficient to obtain an online algorithm with a certain competitive ratio, *i.e.*, we can measure how much information about the future the online algorithm is really missing. We can use this approach to measure the information content of online problems, as proposed in [9]. In some sense, we can see this setup as a generalization of randomized online algorithms where the algorithm has access to another tape with random bits written on it. The concept of *online algorithms with advice* was introduced in [6] and since then revised and applied to several online problems in [2,7].

In the following, we use the same notation as in [2].

Definition 1.1. A minimization online problem consists of a set $\mathcal{I}$ of inputs and a cost function. Every input $I \in \mathcal{I}$ is a sequence of *requests* $I = (x_1, \ldots, x_n)$. Furthermore, a set of feasible outputs (or solutions) is associated with every I; every output is a sequence of *answers* $O = (y_1, \ldots, y_n)$. The cost function assigns a positive real value $\mathrm{cost}(I, O)$ to every input I and any feasible output O. If the input is clear from the context, we omit I and denote the cost of O as $\mathrm{cost}(O)$. For every input I, we call any output O that is feasible for I and has smallest possible cost an *optimal solution of* I, denoted by $\mathtt{Opt}(I)$.

We are now ready to define online algorithms with advice, which solve minimization online problems, and their competitive ratios.

Definition 1.2. Consider an input I of a minimization online problem. 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. We denote the costs of the computed output by $\mathrm{cost}(\mathtt{A}^\phi(I))$. $\mathtt{A}$ is *c-competitive with advice complexity* $s(n)$ if there exists a constant α such that, for every n and for each I of length at most n, there exists

some ϕ such that $\text{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \text{cost}(\mathtt{Opt}(I)) + \alpha$ and at most the first $s(n)$ bits of ϕ have been accessed during the computation of $\mathtt{A}^{\phi}(I)$.

Although ϕ is infinitely long, A only accesses a finite number of bits of ϕ. Since A has no random access to the tape, it reads these bits sequentially, hence the accessed bits form a finite prefix of ϕ. The length of this prefix is, however, determined by the computation performed by A.

Moreover, in this paper we are also dealing with randomized online algorithms, *i.e.*, online algorithms that are allowed to base some of their calculations on random decisions.

Definition 1.3. As above, consider any input I of a minimization online problem. A randomized online algorithm R computes the output sequence $\mathcal{R}^{\phi} = \mathtt{R}^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, \ldots, x_i$, where ϕ is the content of the random tape, *i.e.*, an infinite binary sequence where every bit is chosen uniformly at random and independently of all the others. By $\text{cost}(\mathtt{R}(I))$ we denote the random variable expressing the cost of the solution computed by R on I. Algorithm R is c-competitive if there exists a constant α such that, for every input sequence I, $\mathrm{E}[\text{cost}(\mathtt{R}(I))] \leq c \cdot \text{cost}(\mathtt{Opt}(I)) + \alpha$.

Generating random numbers might be expensive. Hence, we are interested in designing good randomized algorithms that use as few random bits as possible. It is possible to measure the amount of random bits needed by a randomized algorithm as a function of the input length, in a similar way as the time complexity, space complexity, or advice complexity is measured. Randomized algorithms that use only a constant number of random bits, regardless of the input size, are called *barely random algorithms* [4], introduced in [13]. The number of random bits used by these algorithms is asymptotically minimal, hence they can be considered the best algorithms with respect to the amount of randomness used.

It is very simple to observe that, if there is a c-competitive randomized algorithm R solving some online problem P using $r(n)$ random bits, where n is the length of the input instance, there also exists a c-competitive algorithm with advice A solving P with advice complexity $s(n) = r(n)$. Indeed, it is sufficient to provide, for every input, the best possible choice of random bits as an advice for A, which then simulates R in a straightforward way. This result can be used for propagating the lower bounds on advice complexity to lower bounds on randomized algorithms using a restricted number of random bits:

Observation 1.4. Assume that there is no c-competitive algorithm with advice that solves an online problem P with advice complexity $s(n)$. Then there is no c-competitive randomized algorithm that solves P with $r(n) = s(n)$ random bits.

Hence, problems with high information content cannot be efficiently solved by randomized algorithms that use a small number of random bits. The opposite direction does not hold, *i.e.*, it is not always possible to transform an efficient algorithm with advice into an efficient randomized algorithm. For example, consider a (very artificial) problem of guessing a single bit: the first request is always

a symbol '?', the second request is a symbol '0' or '1', and all remaining requests are '*'. The algorithm must answer the request '?' by a symbol '0' or '1' and all subsequent requests by symbols '*'. The total cost of a solution is the sum of the partial costs induced by individual requests, as described below. The answer to '?' always induces costs of 1. Furthermore, if the reply is equal to the second request, all subsequent requests induce costs 0, otherwise they induce costs 1. Easily, an optimal solution has costs of 1, and it can be reached with a single bit of advice. On the other hand, any randomized algorithm, regardless of the number of random bits used, cannot gain expected costs better than $n/2$ for inputs of n requests, which yields an expected competitive ratio of $n/2$. Thus, even problems with information content of 1 bit only might not be efficiently solvable by randomized algorithms.

Nevertheless, the proofs used to construct efficient algorithms with advice may sometimes be adapted to the randomized settings as well. In this way, we obtain some interesting results about barely random algorithms.

Throughout this paper, by $\log x$ we denote the logarithm of x with base 2.

We are now ready to define the two problems we investigate in the following.

1.1. Job shop scheduling

First, we are dealing with the following problem called *job shop scheduling* or Jss for short (see [2,5,8,10,12] for a more detailed introduction and description). Let there be two so-called *jobs* A and B, each of which consists of m *tasks*. The tasks must be executed in sequential order and each task needs to be processed on a specific *machine*. There are exactly m machines identified by their indices $1, 2, \ldots, m$ and each job has exactly one task for every machine. Processing one task takes exactly 1 time unit and, since both jobs need every machine exactly once, we may represent them as permutations $P_A = (p_1, p_2, \ldots, p_m)$ and $P_B = (q_1, q_2, \ldots, q_m)$, where $p_i, q_j \in \{1, 2, \ldots, m\}$ for every $i, j \in \{1, 2, \ldots, m\}$. The meaning of such a permutation is that the tasks must be performed in the order specified by it and that, for every machine, the kth task must be finished before one may start with task $k + 1$. If, at one time step, both jobs A and B ask for the same machine, one of them has to be delayed. The costs of a solution are measured as the total time needed by both jobs to finish all tasks. The goal is to minimize this time, which we also call the makespan.

In an online scenario, the permutations P_A and P_B arrive successively, *i.e.*, only p_1 and q_1 are known at the beginning and p_{i+1} $[q_{j+1}]$ is revealed after p_i $[q_j]$ has been processed.

We use the following graphical representation, which was introduced in [5]. Consider an $(m \times m)$-grid where we label the x-axis with P_A and the y-axis with P_B. The cell (p_i, q_j) models that, in the corresponding time step, A processes a task on machine p_i while B processes a task on q_j. A feasible schedule for the induced instance of Jss is a path that starts at the upper left vertex of the grid and leads to the bottom right vertex. It may use diagonal edges whenever $p_i \neq q_j$. However, if $p_i = q_j$, both A and B ask for the same machine at the same time and

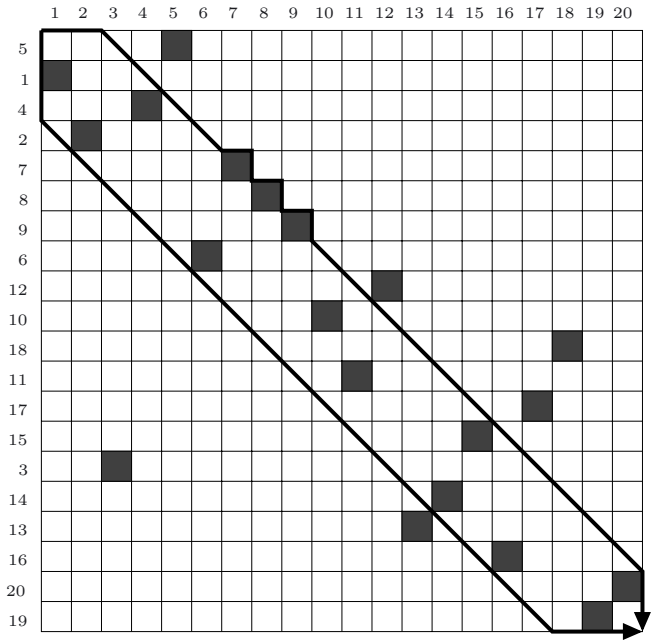

FIGURE 1. An example input with two jobs each of size 20 and the strategies D_{-3} and D_2. Obstacles are marked by filled cells.

therefore, one of them has to be delayed. In this case, we say that A and B collide and call the corresponding cells in the grid *obstacles* (see Fig. 1). If the algorithm has to delay a job, we say that it *hits an obstacle* and may therefore not make a diagonal move, but either a horizontal or a vertical one. In the first case, B gets delayed, in the second case, A gets delayed.

Observation 1.5. The following facts are immediate [2,8].

(i) Since P_A and P_B are permutations, there is exactly one obstacle per row and exactly one obstacle per column for every instance.
(ii) There are exactly m obstacles overall for any instance.
(iii) Every optimal solution has cost of at least m and therefore every online algorithm is 2-competitive or better.
(iv) Every feasible solution makes exactly as many horizontal moves as it makes vertical ones. We call the number of horizontal [vertical] moves the *delay* of the solution.
(v) The cost of a solution is equal to m plus the delay of the solution.
(vi) Hitting an obstacle causes additional costs of at most 1 (in certain situations even none) since one diagonal move can be simulated by exactly one vertical and one horizontal move.

Let diag_0 be the main diagonal (from $(1,1)$ to (m,m)) in the grid. The diagonal that has a distance of i from diag_0 and lies below [above] it, is called diag_{-i} [diag_i]. Similar to [10], for any odd d, we consider a certain set of strategies

$$\mathcal{D}_d = \left\{ D_i \;\middle|\; i \in \left\{ -\frac{d-1}{2}, \ldots, \frac{d-1}{2} \right\} \right\}$$

where D_j is the strategy to move to the starting point of diag_j with j steps, to follow it when possible, and to avoid any obstacle by making a horizontal step directly followed by a vertical one (thus returning to diag_j).

Please note that it is crucial for our analysis that the algorithm *returns* to the diagonal even though there might be situations where it is an advantage not to take the vertical step after the horizontal one.

1.2. Paging

The second problem we focus on is among the most-studied online problems with great practical relevance. The *paging problem*, Paging for short, is motivated by the following circumstance: the performance of today's computers is limited by the fact that the physical memory is a lot slower than the CPU (this fact is known as the von Neumann bottleneck). Hence, the concept of a very fast (and therefore more expensive and consequently smaller) *cache* is used to store as much of the content of the physical memory as possible. We aim at maximizing the communication between the CPU and the cache and thereby minimizing the more costly communication between the CPU and the physical memory. A similar situation occurs between the physical memory and the much slower hard disc. Formally, we deal with the following problem.

Definition 1.6 (paging problem)**.** The input is a sequence of integers representing requests to logical pages $I = (x_1, \ldots, x_n)$, $x_i > 0$. An online algorithm $\mathtt{A}$ maintains a buffer (content of the cache) $B = \{b_1, \ldots, b_K\}$ of K integers, where K is a fixed constant known to $\mathtt{A}$. 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 $\mathtt{A}$ creates the partial output $y_i = 0$. If $x_i \notin B$, then a so-called *page fault* occurs, and the algorithm has to find some *victim* b_j, set $B := (B \setminus \{b_j\}) \cup \{x_i\}$, and output $y_i = b_j$. The cost of the solution $\mathcal{A} = \mathtt{A}(I)$ is the number of page faults, *i.e.*, $\text{cost}(\mathcal{A}) = |\{y_i \mid y_i > 0\}|$.

A more complete description of Paging can be found in [4]. Furthermore, in [2,6], the problem was examined within the scope of advice complexity.

1.3. Organization of this paper

This paper is organized as follows. In Section 2, we deal with the advice complexity of Jss. In Section 2.1, we improve the best so far known upper bound on the advice complexity for achieving optimality. In Section 2.2, we construct an algorithm for any constant d which achieves a competitive ratio tending to

$1 + 1/d$ for large m, using only $\lceil \log d \rceil$ bits of advice. We prove that this bound is almost tight, in particular, that no algorithm with $\lfloor \log d \rfloor$ bits of advice can reach a competitive ratio better than $1 + 1/(3d)$. Finally, Section 3 deals with barely random algorithms. In Section 3.1, we give a barely random algorithm for Jss which achieves an expected competitive ratio that also tends to $1 + 1/d$. Using Observation 1.4 and the results about advice complexity, we obtain lower bounds for barely random algorithms for Jss as well. In Section 3.2, we give a barely random algorithm for Paging which achieves an expected competitive ratio of $\mathcal{O}(\log K)$, what is asymptotically optimal for any randomized algorithm.

2. Advice complexity of job shop scheduling

At first, we consider the advice complexity of Jss, that is, we give lower and upper bounds on the number of advice bits needed to achieve a certain output quality. Doing so, we improve and generalize some of the results obtained in [2].

2.1. Optimality

We quickly discuss the amount of information needed for an online algorithm to produce an optimal output for Jss. In [10], the following lemma was proven.

Lemma 2.1. *For every instance of* Jss*, there is an optimal solution which has costs of at most* $m + \lceil \sqrt{m} \rceil$.

Using this lemma, in [2], an optimal algorithm for Jss has been proposed with advice complexity $s(m) = 2\lceil \sqrt{m} \rceil$. The strategy is to get one bit of advice for every obstacle that is hit indicating whether to move horizontally or vertically to bypass it. Since we know that there always is an algorithm which makes at most $\lceil \sqrt{m} \rceil$ vertical and $\lceil \sqrt{m} \rceil$ horizontal moves by hitting at most $2\lceil \sqrt{m} \rceil$ obstacles, the claim follows easily. We improve this upper bound on the information content of Jss by compressing the advice strings.

Theorem 2.2. *There exists an optimal online algorithm* A (*i.e., an algorithm that always outputs an optimal solution*) *with advice complexity* $s(m) \leq 2\lceil \sqrt{m} \rceil - \frac{1}{4} \log m$ *for any instance of* Jss.

Proof. Indeed, there are $2^{2\lceil \sqrt{m} \rceil}$ possible strings of length $2\lceil \sqrt{m} \rceil$ out of which the oracle provides one to the online algorithm A thus using $2\lceil \sqrt{m} \rceil$ bits in the proof mentioned above. Recall that A knows m and therefore $\lceil \sqrt{m} \rceil$. The crucial part is that all of these strings have a very nice structural property: due to Observation 1.5 (iv), they contain as many ones as they contain zeros. For a fixed m, there exist exactly

$$\binom{2\lceil \sqrt{m} \rceil}{\lceil \sqrt{m} \rceil} = \frac{(2\lceil \sqrt{m} \rceil)!}{(\lceil \sqrt{m} \rceil!)^2}$$

such strings. Applying Stirling's approximation, we get

$$\frac{(2\lceil\sqrt{m}\,\rceil)!}{(\lceil\sqrt{m}\,\rceil!)^2} < \frac{4^{\lceil\sqrt{m}\,\rceil}}{\sqrt{\pi\lceil\sqrt{m}\,\rceil}}.$$

Enumerating all possible strings in canonical order and then merely communicating the index of the specific string gives that it suffices to send

$$\log\left(\frac{4^{\lceil\sqrt{m}\,\rceil}}{\sqrt{\pi\lceil\sqrt{m}\,\rceil}}\right) = \lceil\sqrt{m}\,\rceil\cdot\log 4 - \log\left(\sqrt{\pi\lceil\sqrt{m}\,\rceil}\right) \leq 2\lceil\sqrt{m}\,\rceil - \frac{1}{4}\log m$$

bits. □

2.2. Competitive ratio

Intuitively speaking, we now show that there always exists a solution with low costs close to the main diagonal which implies that only a few bits are needed to achieve a good result. In what follows, d is always a small odd constant which is independent of the input size. Furthermore, let $\delta := d^2/4 - d$ (please note that $\delta > -1$). Before we continue, we need the following lemma.

Recall that we call the number of horizontal [vertical] moves of a solution its delay; the costs (makespan) of the solution is always equal to m plus its delay. The delay of a diagonal strategy D_i is always $|i|$ plus the number of obstacles on diag_i. If $i \geq 0$, the strategy D_i makes i horizontal moves to reach diag_i and then a single horizontal move for every obstacle on diag_i. The argument for $i \leq 0$ is similar, but using vertical moves.

Lemma 2.3. *There exists a diagonal in $\mathcal{D}_d$ such that the corresponding diagonal strategy has a delay of at most $\lceil\frac{\delta+m}{d}\rceil$.*

Proof. As we have seen in Observation 1.5 (ii), there are exactly m obstacles in the whole grid which represents the instance at hand.

Towards contradiction, suppose that the claim is wrong. Therefore, each of the considered strategies has costs of at least $m+\lceil\frac{\delta+m}{d}\rceil+1$. This means that at least $\lceil\frac{\delta+m}{d}\rceil+1$ obstacles are on the main diagonal, at least $\lceil\frac{\delta+m}{d}\rceil$ obstacles are on D_{-1} and D_1, in general at least

$$\left\lceil\frac{\delta+m}{d}\right\rceil + 1 - i$$

obstacles have to be on D_{-i} and D_i, and finally

$$\left\lceil\frac{\delta+m}{d}\right\rceil - \frac{d-3}{2}$$

obstacles are on $D_{-(d-1)/2}$ and $D_{(d-1)/2}$. If we now sum up all obstacles, we immediately get a total of

$$\begin{aligned}
&\left\lceil \frac{\delta+m}{d} \right\rceil + 1 + 2 \sum_{i=1}^{(d-1)/2} \left(\left\lceil \frac{\delta+m}{d} \right\rceil + 1 - i \right) \\
&\geq \frac{\delta+m}{d} + 1 + \left(\frac{\delta+m}{d} + 1 \right)(d-1) - 2 \sum_{i=1}^{(d-1)/2} i \\
&= \left(\frac{\delta+m}{d} + 1 \right) d - \frac{d^2-1}{4} = m + d + \frac{d^2}{4} - d - \frac{d^2-1}{4}
\end{aligned}$$

obstacles which is strictly more than m and therefore directly contradicts our assumption. □

We can now prove the following theorem.

Theorem 2.4. *For every d, there exists an online algorithm $\mathtt{A}_d$ that reads $\lceil \log d \rceil$ bits of advice and achieves a competitive ratio of*

$$1 + \frac{1}{d} + \frac{d}{4(m+1)} - \frac{d+1}{d(m+1)}.$$

Proof. Let $\mathtt{A}_d$ know d and receive $\lceil \log d \rceil$ bits in total that tell the algorithm which out of the diagonals from $\mathcal{D}_d$ to follow using the aforementioned strategy. As we have shown in Lemma 2.3, one out of these strategies has a delay of at most $\lceil \frac{\delta+m}{d} \rceil$. Note that, if the optimal solution has cost m, this solution must take the main diagonal. But in this case, $\mathtt{A}$ is always optimal, because there are no obstacles on diag_0 and the corresponding delay is therefore 0. Hence, without loss of generality, we may assume a lower bound of $m+1$ for the optimal solution. Putting this together, we get a competitive ratio of $\mathtt{A}$ of at most

$$\begin{aligned}
\frac{m + \left\lceil \frac{d^2/4-d+m}{d} \right\rceil}{m+1} &\leq \frac{m + \frac{d^2/4-d+m}{d} + 1}{m+1} = \frac{m + \frac{m}{d} + \frac{d}{4}}{m+1} \\
&= 1 + \frac{1}{d} + \frac{d}{4(m+1)} - \frac{d+1}{d(m+1)}
\end{aligned}$$

as we claimed. □

Figure 2 shows how the competitive ratio of $\mathtt{A}_d$ behaves depending on the number of advice bits. In [10], it was shown that, for any $\varepsilon > 0$, any deterministic online algorithm without advice cannot be better than $(1 + 1/3 - \varepsilon)$-competitive. On the other hand, the competitive ratio of $\mathtt{A}_d$ tends to $(1 + 1/7)$ (recall that d is odd) with only 3 bits of advice for m tending to infinity. Hence, we can beat deterministic strategies with only very little additional information. A similar result was shown for the paging problem in [2].

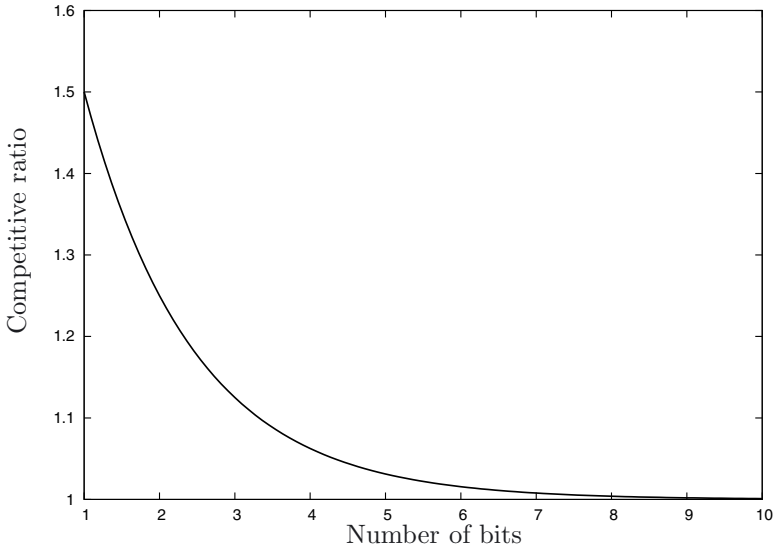

FIGURE 2. The competitive ratio of $\mathtt{A}_d$ depending on $\log d$ for m tending to infinity.

In Theorem 2.4, we did not care about the uniformity of $\mathtt{A}_d$ for different values of d. It is, however, not difficult to avoid the non-uniformity, *i.e.*, to define a single algorithm $\mathtt{A}$ that reaches a competitive ratio tending to $1+1/d$ for any d, depending on the advice received. To do so, the oracle first writes $2\lceil\log\lceil\log d\rceil\rceil$ bits on the advice tape. Bits at odd positions give the number $\lceil\log d\rceil$ in binary, bits at even positions are zero while the next bit still belongs to the first $2\lceil\log\lceil\log d\rceil\rceil$ bits and one otherwise (therefore, this string is self-delimiting). After reading the first bit of value one at an even position, $\mathtt{A}$ knows how many bits to read afterwards telling it what strategy to choose.

Corollary 2.5. *There exists an online algorithm with advice complexity* $\lceil\log d\rceil + 2\lceil\log\lceil\log d\rceil\rceil$ *that achieves a competitive ratio tending to* $1+1/d$ *with growing* m.

It is not difficult to see that the analysis of algorithm $\mathtt{A}_d$ is almost tight for every d. To show this, we give a construction that blocks all diagonals the algorithm chooses from. Following any of the blocked diagonals causes the algorithm to have costs of at least $m+m/d$, whereas an optimal solution has costs of exactly $m+1$.

Lemma 2.6. *For any* d *and any* $\varepsilon > 0$*, the competitive ratio of the algorithm* $\mathtt{A}_d$ *is not better than* $1+1/d-\varepsilon$ *for infinitely many* m.

Proof. Let m be even. We now describe how to sufficiently delay every possible diagonal strategy. Suppose we want to make sure that every strategy has a delay of at least l (where l is divisible by 2). At first, we place l obstacles in the center of the main diagonal, *i.e.*, in the cells $(m/2-l/2+1, m/2-l/2+1)$ to $(m/2+l/2, m/2+l/2)$. For now, let us focus on the cells which are in the bottom-right quadrant of the $(m\times m)$-grid. For each $i\in\{1,2,\ldots,(d-1)/2\}$, we create

one block of obstacles. The block corresponding to i consists of $l-i$ obstacles. All of these obstacles are put on the ith diagonal above the main one, in consecutive rows, just below the rows used by the block $i-1$. In particular, the obstacles of block 1 are located on

$$\left(\frac{m+l}{2}+1, \frac{m+l}{2}+2\right), \ldots, \left(\frac{m+l}{2}+l-1, \frac{m+l}{2}+l\right),$$

the obstacles of block 2 are located on

$$\left(\frac{m+l}{2}+l, \frac{m+l}{2}+l+2\right), \ldots, \left(\frac{m+l}{2}+2l-3, \frac{m+l}{2}+2l-1\right),$$

etc. Hence, we need to use $l-i$ rows and $l-i+1$ columns to build the block i (the first column of the block is empty, since block i is on a different diagonal than block $i-1$).

To be able to successfully build all of the blocks, we need at least

$$\frac{l}{2}+1+(l-1)+1+(l-2)+1+\ldots+\left(l-\frac{d-1}{2}\right)$$

columns (clearly, if there are enough columns available, there are enough rows as well). Since we have exactly $m/2$ columns, we have to make sure that

$$\begin{aligned} & \frac{l}{2}+\sum_{i=1}^{(d-1)/2} 1+l-i \leq \frac{m}{2} \\ \iff \quad & \frac{l}{2}+\frac{d-1}{2}(1+l)-\frac{d^2-1}{8} \leq \frac{m}{2} \\ \iff \quad & l \leq \frac{m+\frac{d^2+3}{4}-d}{d}. \end{aligned}$$

We can ensure this by taking l to be the smallest even integer such that

$$l \geq \frac{m+\frac{d^2+3}{4}-d}{d}-2.$$

The same construction can be performed in the top-left quadrant in a symmetric way. In every block, there is one free column. It remains to use the rows not used by any block (nor the obstacles in the main diagonal) to put a single obstacle to every such free column. To do so, we use the top-right and bottom-left quadrant. It is straightforward to observe that this is always possible, even without using any diagonal neighboring the main one.

An example of this construction for $m=20$, $l=4$, and $\mathcal{D}_5$ is shown in Figure 3. It is immediately clear that any optimal solution has costs of exactly $m+1$: an optimal solution follows the main diagonal until the first obstacle is hit. Afterwards, the solution makes one vertical step and follows the first diagonal below the main one (*i.e.*, diag_{-1}).

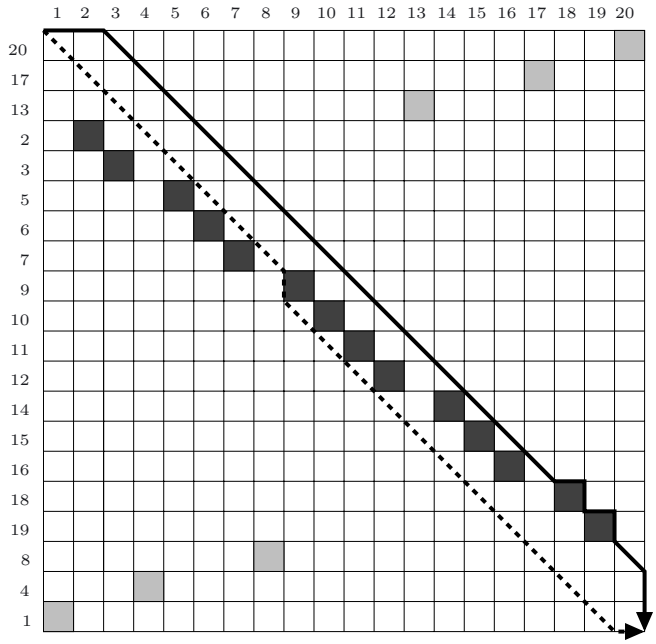

FIGURE 3. A hard instance for $\mathcal{D}_5$.

$\mathtt{A}_d$ calculates a solution with delay at least l, *i.e.*, with costs at least

$$m + l \geq m + \frac{m + \frac{d^2+3}{4} - d}{d} - 2 \geq (m+1)(1 + \frac{1}{d}) - 4 - \frac{1}{d} + \frac{d^2+3}{4d}.$$

Therefore, the competitive ratio of $\mathtt{A}_d$ on this instance is at least

$$\frac{(m+1)(1+\frac{1}{d}) - 4 - \frac{1}{d} + \frac{d^2+3}{4d}}{m+1} = 1 + \frac{1}{d} + \frac{d^2 - 16d - 1}{4d(m+1)} \geq 1 + \frac{1}{d} - \varepsilon$$

if m is large enough, which we can assume for infinitely many m. Note that if $d \geq 17$, the inequality always holds even if $\varepsilon = 0$. □

Up to this point, we have shown that, with a small constant number of advice bits, it is possible to perform very well. In [2], it was shown that the information content of JSS is at least

$$\left\lfloor \frac{\sqrt{16m+9} - 11}{8} \right\rfloor,$$

i.e., at least that many advice bits are needed for any online algorithm with advice to be optimal.

A naturally arising question is whether we can be $(1 + \Theta(1/m))$-competitive with reading a constant number of advice bits, *i.e.*, if it suffices to use a constant

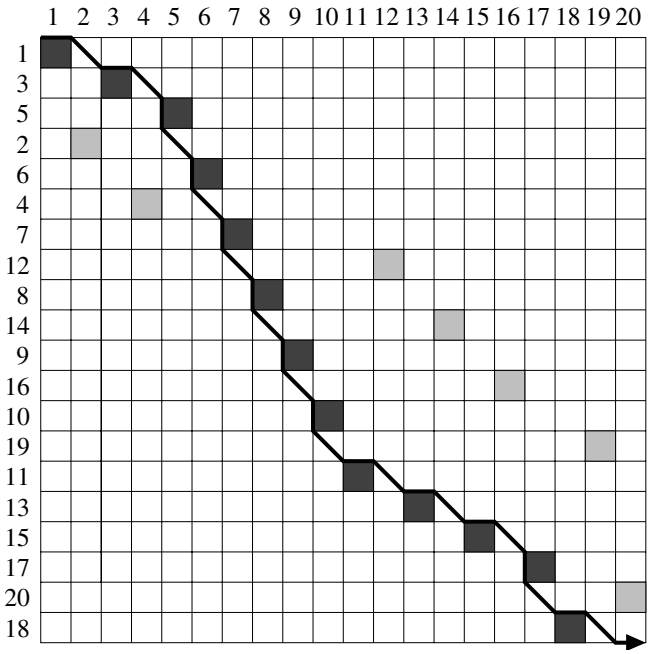

FIGURE 4. An example of how to place the obstacles in such a way that any deterministic algorithm cannot make two consecutive diagonal moves as presented in [8].

number of bits to get arbitrarily close to the optimal solution. In the following, we disprove this.

Theorem 2.7. *For any $\varepsilon > 0$, any online algorithm with advice that reads b bits of advice cannot be better than*

$$\left(1 + \frac{1}{3 \times 2^b} - \varepsilon\right)\text{-competitive.}$$

Proof. In [10], it was shown that any deterministic online algorithm A for JSS has a competitive ratio of at least 4/3. There always exists an adversary that can make sure that every second move of A is not a diagonal move: the intuitive idea is that, after every diagonal move of A, the algorithm reaches a column and a row in which the adversary has not yet placed an obstacle. This idea is shown in Figure 4. Recall that we already know that there always exists an optimal solution with costs of at most $m + \lceil \sqrt{m} \rceil$ (Lem. 2.1).

For any algorithm that reads b bits of advice and any $\varepsilon > 0$, we find some (arbitrarily large) m and construct an input instance of size $m \times m$ such that the algorithm has a delay of at least $\frac{m}{3 \times 2^b}$. Hence, the makespan of the algorithm is at least $m\left(1 + \frac{1}{3 \times 2^b}\right)$ and, since the optimal solution has a makespan of at most

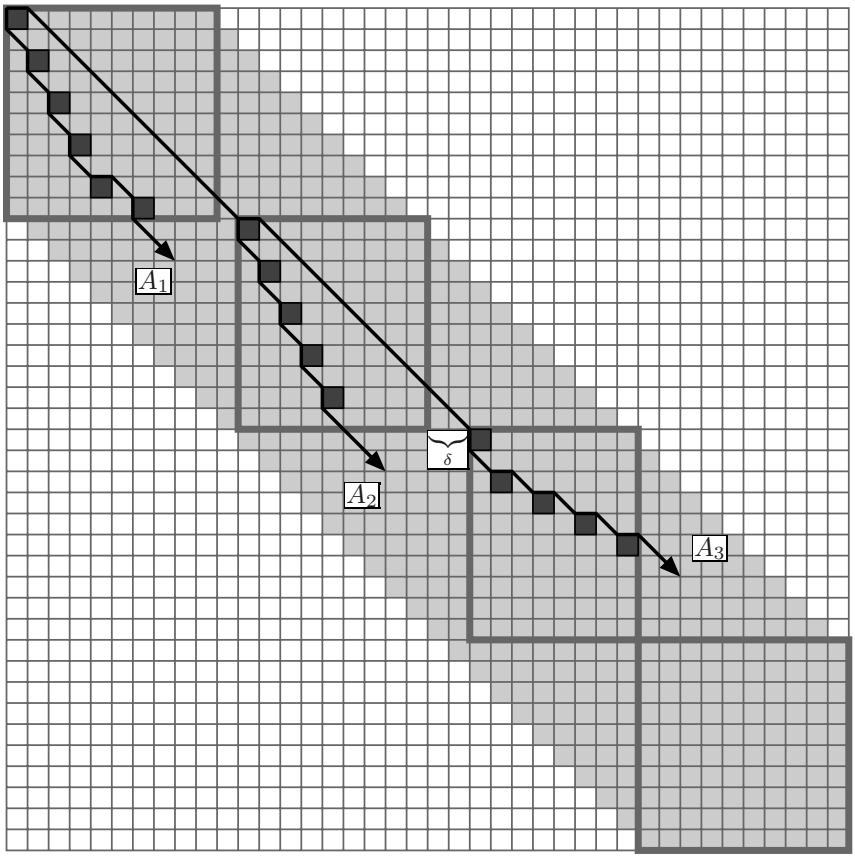

FIGURE 5. A hard instance for $\mathtt{A}_b$ as used in the proof of Theorem 2.7, which uses the construction of [8] (Fig. 4) 2^b times.

$m + \lceil \sqrt{m} \rceil$, the competitive ratio of the algorithm cannot be better than

$$\frac{m\left(1 + \frac{1}{3\times 2^b}\right)}{m + \lceil \sqrt{m} \rceil} \geq 1 + \frac{1}{3 \times 2^b} - \varepsilon$$

for any large enough m.

In the following, let m be a multiple of 2^b. Suppose that we are now dealing with any algorithm $\mathtt{A}_b$ that reads b bits of advice while processing an input of size m. We impose another virtual grid on the $(m \times m)$-grid, where each virtual cell consists of $m' := m/2^b$ original cells. Let us now consider the 2^b virtual cells on the main diagonal (as shown in Fig. 5). We call these cells *blocks* and label them $S_1, S_2, \ldots, S_{2^b}$.

Furthermore, we call all original cells that have a deviation of less than m' from the main diagonal the *active zone* (marked grey in Fig. 5). Any algorithm that leaves this zone at any point makes at least m' horizontal [vertical] moves and

thus has a delay of at least $m' > \frac{m}{3\times 2^b}$. We may therefore assume that the given algorithm never leaves the active zone.

Observe that we may think of $\mathtt{A}_b$ as 2^b deterministic algorithms we have to deal with. Without loss of generality, we may assume that each of these algorithms makes a diagonal move when possible [2], Lemma 2. We may label the algorithms $A_1, A_2, \ldots, A_{2^b}$ by sorting the corresponding advice in canonical order and assign each deterministic algorithm A_i to exactly one block S_i.

We now construct the input instance sequentially in such a way that all obstacles are located in some block S_i. Note that S_i spans the rows and the columns $m'i+1, \ldots, m'i+m'$. Recall that p_i denotes the ith task of the first job and q_i denotes the i-th task of the second job. We thus construct the input such that $p_{m'i+1}, \ldots, p_{m'i+m'}$, as well as $q_{m'i+1}, \ldots, q_{m'i+m'}$, are permutations of the numbers $m'i+1 \ldots, m'i+m'$.

Assume that, so far, we have constructed $S_1, \ldots, S_{i-1}$. Now we construct S_i in such a way that A_i has delay at least $\frac{m}{3\times 2^b}$, regardless of the content of any S_j for $j > i$. Without loss of generality, assume that A_i reaches the right border of S_{i-1} at distance δ above the main diagonal; the case when A_i reaches the bottom border of S_{i-1} is analogous. If $i = 1$, we define $\delta := 0$, since the first algorithm starts at the top left point of the main diagonal.

Since there are no obstacles outside the blocks and we assume that A_i makes a diagonal move whenever possible, A_i makes δ diagonal moves after leaving S_{i-1} until it reaches the top border of S_i, *i.e.*, the top left corner of the cell $(m'i + \delta + 1, m'i + 1)$. We assign the first δ tasks to the first job sequentially, *i.e.*, $p_{m'i+j} = m'i + j$ for all $j \in \{1, \ldots, \delta\}$.

After A_i reaches the cell $(m'i + x, m'i + y)$, the first $m'i + x$ tasks of the first job and first $m'i + y$ tasks of the second job must be assigned. In the sequel, we maintain the invariant that, in such a situation, only numbers up to $m'i+\max(x, y)$ are used in both jobs. This invariant holds before A_i reaches $(m'i+\delta+1, m'i+1)$.

After A_i reaches S_i, we employ the strategy of [10] to ensure that every second move of A_i is non-diagonal: at first, we assign

$$p_{m'i+\delta+1} = q_{m'i+1} := m'i + \delta + 1,$$

thus creating an obstacle and the next move of A_i will be a non-diagonal one. Every time A_i makes a horizontal [vertical] move, we assign the smallest possible task as the next task of the first [second] job. When A_i makes a diagonal move at cell $(m'i+x, m'i+y)$ (thus reaching the top-left corner of cell $(m'i+x+1, m'i+y+1)$), we assign

$$p_{m'i+x+1} = q_{m'i+y+1} := m'i + \max(x, y) + 1,$$

thus creating an obstacle and forcing A_i to make another non-diagonal move. It is easy to verify that we can always follow this strategy due to the validity of the invariant and that the invariant is never violated.

We follow this strategy until A_i reaches the right or bottom border of S_i. Assume that A_i makes h horizontal moves, v vertical moves, and d diagonal moves in this part of the computation (that is, in S_i). Since every diagonal move is followed

by a non-diagonal one and the first move is non-diagonal, we have $h + v \geq d$. We estimate the lower bound on the total delay D of A_i on the constructed input instance. Even though we have not constructed S_j for $j > i$ yet, we can proceed, since our bound will not depend on them. Since the total number of horizontal and vertical moves of A_i over the whole input must be equal, D is equal to the total number of horizontal [vertical] moves of A_i. We distinguish two cases:

(1) $h \geq v$: in this case, A_i reaches the right border of S_i. Since A_i entered S_i in column $m'i + \delta + 1$, there were $m' - \delta$ non-vertical moves, hence

$$m' - \delta = h + d \leq 2h + v \leq 3h.$$

Therefore, $h \geq (m' - \delta)/3$. Since A_i leaves S_{i-1} at distance δ above the main diagonal, it made at least δ horizontal steps before it entered S_i. Thus, we can bound the total number of horizontal steps of A_i, which is equal to D, as

$$D \geq \delta + h \geq \frac{m' + 2\delta}{3} \geq \frac{m'}{3}.$$

(2) $h < v$: assume that A_i leaves S_i at distance δ' above the bottom border of S_i; if A_i reaches the bottom border, $\delta' = 0$, otherwise $\delta' > 0$. Since A_i made $m' - \delta'$ non-horizontal moves in S_i, we have

$$m' - \delta' = v + d \leq 2v + h \leq 3v$$

and $v \geq (m' - \delta')/3$. After leaving S_i, algorithm A_i must make at least δ' vertical moves to end at the main diagonal. Hence, the total number of vertical steps of A_i, which is equal to D, can be bounded as

$$D \geq \delta' + v \geq \frac{m' + 2\delta'}{3} \geq \frac{m'}{3}.$$

In both cases, A_i has delay at least $m'/3$. Therefore, after constructing all S_i in the described way, we obtain an input instance where every A_i has makespan at least $m + \frac{m}{3 \times 2^b}$. □

Using this result, an easy calculation gives that the bound from Theorem 2.4 is tight up to a multiplicative constant of

$$\frac{3 \times 2^b + 3}{3 \times 2^b + 1}$$

which tends to 1 for an increasing b.

3. Barely random algorithms

At first, we again consider Jss and make use of ideas we have already considered within the advice complexity framework in Section 2.

3.1. Job shop scheduling

As above, we consider the class $\mathcal{D}_d$ of diagonal strategies as introduced in Section 1 for some odd constant $d \geq 1$. Consider a barely random algorithm $\mathtt{R}_d$ which randomly chooses a strategy from this class using at most $\lceil \log d \rceil$ random bits to do so. Our results from Section 2.2, together with Observation 1.4, imply that we cannot hope for anything significantly better than the competitive ratio $\mathtt{A}_d$ achieves. However, the following theorem holds.

Theorem 3.1. *The algorithm $\mathtt{R}_d$ achieves an expected competitive ratio of*

$$1 + \frac{1}{d} + \frac{d^2 - 1}{4dm}.$$

Proof. For every odd d, consider the following random variables X_1, X_2, X, Y: $\mathcal{D}_d \to \mathbb{R}$, where $X_1(D_i)$ is the delay caused by the initial horizontal [vertical] steps made by the strategy D_i, $X_2(D_i)$ is the delay caused by D_i hitting obstacles, $X(D_i) = X_1(D_i) + X_2(D_i)$ is D_i's overall delay, and $Y(D_i) = m + X(D_i)$ is D_i's overall cost. Recall that D_{-j} and D_j make the same amount of vertical [horizontal] moves at the beginning. Since there are exactly m obstacles in total for every instance, we immediately get

$$\mathrm{E}[X_2] = \frac{1}{d}\left(X_2(D_0) + 2\sum_{i=1}^{(d-1)/2} X_2(D_i)\right) \leq \frac{m}{d}$$

and since $X_1(D_0) = 0$, we get

$$\mathrm{E}[X_1] = \frac{1}{d}\left(X_1(D_0) + 2\sum_{i=1}^{(d-1)/2} X_1(D_i)\right) = \frac{2}{d}\sum_{i=1}^{(d-1)/2} i = \frac{d^2 - 1}{4d}.$$

Due to the linearity of expectation, it follows that

$$\mathrm{E}[Y] = m + \mathrm{E}[X] = m + \mathrm{E}[X_2] + \mathrm{E}[X_1] \leq m + \frac{m}{d} + \frac{d^2 - 1}{4d}.$$

Therefore, the expected competitive ratio of $\mathtt{R}_d$ is at most

$$\frac{\frac{(d+1)m}{d} + \frac{d^2-1}{4d}}{m} = 1 + \frac{1}{d} + \frac{d^2 - 1}{4dm}$$

which, for increasing m, tends to $1 + 1/d$. □

Please note that this bound is very close to the one shown for the corresponding online algorithm with advice in Theorem 2.4, but slightly worse. In fact, we can use the proof of Theorem 3.1 as a probabilistic proof of the upper bound on the advice complexity of Jss. On the other hand, we can apply Theorem 2.7 and

Observation 1.4 to obtain that, for any $\varepsilon > 0$, no randomized algorithm that uses at most b random bits can obtain a competitive ratio of $1 + 1/(3 \times 2^b) - \varepsilon$. Hence, barely random algorithms for JSS cannot have a competitive ratio that tends to 1 with growing m. This means they perform worse than the randomized algorithms from [10] (using an unrestricted number of random bits), which can reach a competitive ratio that tends to 1 with growing m.

3.2. PAGING

Next, we look at PAGING and show the existence of a barely random algorithm that achieves a low competitive ratio. It is well known that no deterministic algorithm for PAGING can be better than K-competitive, where K is the size of the cache, and that there exists an $\mathcal{O}(\log K)$-competitive randomized algorithm for PAGING [4].

More precisely, there is a H_K-competitive randomized algorithm for PAGING, where $H_K = \sum_{i=1}^{K} 1/i$ is the Kth harmonic number, and this bound is tight [1]. To the best of our knowledge, however, all randomized algorithms for PAGING known so far that reach a competitive ratio $\mathcal{O}(\log K)$ use $\Omega(n)$ random bits for inputs of length n, and no efficient barely random algorithm for PAGING is known up to now.

In [2,3], it was shown that there exists an online algorithm with advice $\mathtt{A}$ that reads $\log b$ bits of advice and has a competitive ratio of at most

$$3 \log b + \frac{2(K+1)}{b} + 1,$$

where K is the buffer size of $\mathtt{A}$ and b is a power of 2. This result can be easily adapted for the randomized case:

Theorem 3.2. *Consider* PAGING *with buffer size K, and let $b < K$ be a power of 2. There exists a barely random algorithm for* PAGING *that uses $\log b$ random bits, regardless of the input size, and achieves a competitive ratio of*

$$r \leq 3 \log b + \frac{2(K+1)}{b} + 1.$$

Proof. The proof is almost identical to the proof of Theorem 5 in [3]. The core idea of this proof is to construct b deterministic algorithms $\mathtt{A}_1, \ldots, \mathtt{A}_b$ such that, for any input instance, the total number of page faults generated by all algorithms together is limited. In particular, the proof of Theorem 5 in [3] describes a set of b algorithms such that, for any input instance I, the total number of page faults is bounded by

$$m\frac{b}{2}(3 \log b + 1) + m(K+1),$$

where m is a certain parameter depending on I. Furthermore, any algorithm makes at least $m/2$ page faults on I. Hence, selecting one of the b algorithms uniformly

at random and running it yields a randomized algorithm with an expected number of page faults

$$m\left(\frac{K+1}{b}+\frac{3}{2}\log b+\frac{1}{2}\right).$$

Thus, the expected competitive ratio of such randomized algorithm will be at most

$$\frac{m\left(\frac{K+1}{b}+\frac{3}{2}\log b+\frac{1}{2}\right)}{\frac{m}{2}}=\frac{2(K+1)}{b}+3\log b+1.$$

Obviously selecting the algorithm can be done with $\log b$ bits. □

The previous theorem shows that there exists a barely random algorithm for PAGING that uses only $\lfloor \log K \rfloor$ bits and reaches a competitive ratio of $\mathcal{O}(\log K)$, which is asymptotically equivalent to the best possible randomized algorithm (and this bound is furthermore known to be tight).

REFERENCES

[1] D. Achlioptas, M. Chrobak and J. Noga, Competitive analysis of randomized paging algorithms. *Theoret. Comput. Sci.* **234** (2000) 203–218.

[2] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič and T. Mömke, On the advice complexity of online problems, in *20th International Symposium on Algorithms and Computation (ISAAC 2009) Lect. Notes Comput. Sci.* **5878** (2009) 331–340.

[3] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič and T. Mömke, Online algorithms with advice. To appear.

[4] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, New York (1998).

[5] P. Brucker, An efficient algorithm for the job-shop problem with two jobs. *Computing* **40** (1988) 353–359.

[6] S. Dobrev, R. Královič and D. Pardubská, How much information about the future is needed?, in *34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)* (2008) 247–258.

[7] Y. Emek, P. Fraigniaud, A. Korman and A. Rosén, Online computation with advice. *Theoret. Comput. Sci.* **412** (2010) 2642–2656.

[8] J. Hromkovič, *Design and analysis of randomized algorithms: Introduction to design paradigms*. Springer-Verlag, New York (2006).

[9] J. Hromkovič, R. Královič and R. Královič, Information complexity of online problems, in *35th International Symposium on Mathematical Foundations of Computer Science (MFCS 2010). Lect. Notes Comput. Sci.* **6281** (2010) 24–36.

[10] J. Hromkovič, T. Mömke, K. Steinhöfel and P. Widmayer, Job shop scheduling with unit length tasks: bounds and algorithms. *Algorithmic Operations Research* **2** (2007) 1–14.

[11] D. Komm and R. Královič, Advice complexity and barely random algorithms, in *37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2011). Lect. Notes Comput. Sci.* **6543** (2011) 332–343.

[12] T. Mömke, On the power of randomization for job shop scheduling with k-units length tasks. *RAIRO-Theor. Inf. Appl.* **43** (2009) 189–207.

[13] N. Reingold, J. Westbrook and D. Sleator, Randomized competitive algorithms for the list update problem. *Algorithmica* **11** (1994) 15–32.

Communicated by J. Hromkovic.
Received December 21, 2010. Accepted March 23, 2011.