RAIRO-Theor. Inf. Appl. 43 (2009) 585–613
DOI: 10.1051/ita/2009012

Available online at:
www.rairo-ita.org

MEASURING THE PROBLEM-RELEVANT INFORMATION IN INPUT *

Stefan Dobrev[1], Rastislav Královič[2] and Dana Pardubská[2]

Abstract. We propose a new way of characterizing the complexity of online problems. Instead of measuring the degradation of the output quality caused by the ignorance of the future we choose to quantify the amount of additional global information needed for an online algorithm to solve the problem optimally. In our model, the algorithm cooperates with an oracle that can see the whole input. We define the advice complexity of the problem to be the minimal number of bits (normalized per input request, and minimized over all algorithm-oracle pairs) communicated by the algorithm to the oracle in order to solve the problem optimally. Hence, the advice complexity measures the amount of problem-relevant information contained in the input. We introduce two modes of communication between the algorithm and the oracle based on whether the oracle offers an advice spontaneously (helper) or on request (answerer). We analyze the Paging and DiffServ problems in terms of advice complexity and deliver upper and lower bounds in both communication modes; in the case of DiffServ problem in helper mode the bounds are tight.

Mathematics Subject Classification. 68Q25, 68W40, 68Q10, 68Q17.

Keywords and phrases. Online algorithms, communication complexity, advice complexity, paging.

* *Supported by TH-18 07-3 and APVV-0433-06. Preliminary version of this paper appeared at SOFSEM 2008.*

[1] Institute of Mathematics, Slovak Academy of Sciences, Slovakia; Stefan.Dobrev@savba.sk

[2] Department of Computer Science, Comenius University, Bratislava, Slovakia; [kralovic;pardubska]@dcs.fmph.uniba.sk

Article published by EDP Sciences

1. Introduction

The term "online" is used to describe algorithms that operate without the full knowledge of the input: a typical scenario would be a server that must continually process a sequence of requests in the order they arrive. More formally, an online algorithm processing an input sequence of requests $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$ produces an output sequence $\boldsymbol{y} = \langle y_1, y_2, \ldots, y_n \rangle$ in such a way that each y_i is computed as a function of the prefix $\langle x_1, x_2, \ldots, x_i \rangle$. On the other hand, an algorithm computing the whole output sequence $\boldsymbol{y}$ from the entire input sequence $\boldsymbol{x}$ is termed "offline". The systematic study of online problems began in the late sixties [16], and has received much attention over the years (see *e.g.* [3,6]). The standard measure used for evaluating online algorithms is the competitive ratio [20,24], *i.e.* the worst case ratio between the solution quality of the given online algorithm and that of the optimal offline algorithm. The competitive complexity of an online problem is the best competitive ratio attainable by an online algorithm solving the problem. Intuitively, this measure describes the price, in terms of solution quality, that has to be paid for not knowing the whole input from the beginning.

Consider, for example, the well known SkiRental problem [21]: a skier can rent a set of skis for one day for a unit price, or buy them for a fixed price c. However it is only the morning of each day when it becomes clear if the skier wants to continue skiing or not. A classical result of the competitive analysis [21] shows that the optimal worst case performance is achieved by first renting the skis for $c-1$ days, and then buying them, which gives a competitive ratio of $2-1/c$. The interpretation of this result is that the input carries a global information relevant to the problem; ignoring this information leads to a degradation of the best possible solution quality by a factor of $2-1/c$.

In this paper we propose a new way of characterizing the complexity of online problems. The hardness incurred by the online setting comes from the fact that there is some information about the future input that is not available to the algorithm. In our approach we measure the amount of this hidden information. However, the input contains also information that is irrelevant to the problem at hand, and we have to find a way of distilling the problem-relevant information from the input.

Our approach to measure the relevant information is inspired by the communication complexity research. We consider, in addition to the algorithm itself, an oracle that sees the whole input and knows the algorithm. When computing the ith output y_i, the algorithm not only sees the sequence $\langle x_1, x_2, \ldots, x_i \rangle$, but can also communicate with the oracle. We require that the algorithm always computes an optimal solution. The advice complexity of the algorithm is the number of bits communicated by the algorithm to the oracle, normalized per request. The advice complexity of an online problem is the minimum advice complexity over all oracle-algorithm pairs that together solve the problem.

Apart from its theoretical significance, this measure can be of use in some semi-online scenarios where the input is available, but has to be accessed sequentially by the algorithm. As a motivation example, consider the scenario where a simple

device (*e.g.* a remote robot) is supposed to process a large amount of data (*e.g.* a series of orders) in an online fashion. The data are stored and sequentially fed to the device from a powerful entity (base station) over a (wireless) communication link. In order to guide the robot in the processing, the base station may pre-process the data and send some additional information together with each data item. However, since communication rapidly depletes the robots battery, the amount of this additional communication should be kept as small as possible.

To return to the original example, the structure of the SkiRental problem is very simple from the advice complexity point of view: the optimal offline solution is either to buy the skis on the first day or rent them during all days. Hence, although the competitive ratio of the SkiRental problem is asymptotically 2, a single bit of information about the whole input is sufficient for the online algorithm to achieve optimal performance. As we show later, there are other problems, for which this amount of required information is much higher.

We are primarily interested in the relationship between the competitive ratio and the advice complexity. If the competitive ratio measures the price paid for the lack of information about future, the advice complexity quantifies for how much information this price is paid.

Note that there are two ways to achieve trivial upper bounds on advice complexity: (1) the oracle can send, in some compressed way, the whole input to the algorithm, which then can proceed as an optimal offline algorithm; and (2) the oracle can tell the algorithm exactly what to do in each step. However, both these approaches can be far from optimum. In the first case all information about the future input is communicated, although it may not be relevant[1]. In the second case, the power of the online algorithm is completely ignored. Indeed, an online algorithm may be able to process large parts of the input optimally without any advice, requiring only occasional help from the oracle.

In this paper, we define two modes of interaction with the oracle. In the *answerer* mode the algorithm may, in any step, ask for an advice, and it receives a non-empty string from the oracle. The overall number of bits obtained by the algorithm is a measure of the complexity of the input with respect to the problem. In the *helper* mode the measuring of the amount of problem-specific information contained in the input comes from the following intuition: consider an online problem, and suppose there is a small family of algorithms such that each of them solves the problem optimally for some class of instances. In order to obtain an optimal algorithm, it is sufficient to be able to predict which algorithm to use for the upcoming requests. If the number of algorithms is small and so is the number of times they have to be switched on any particular input, the problem can intuitively be considered as having a simple structure. In order to model this intuition, in the *helper* mode the algorithm does not activate the oracle; instead, the oracle oversees the progress of the algorithm, and occasionally sends some pieces of advice.

[1]Consider, *e.g.* the Paging problem. There may be a long incompressible sequence of requests that do not result in a page fault; the information about the actual requests in this sequence is useless for the algorithm.

	competitive ratio	helper	answerer
PAGING	K [29]	$(0.1775, 0.2056)$	$(0.4591, 0.5 + \varepsilon)$
DIFFSERV	$\approx$1.281[11]	$\frac{1}{K}$	$\left(\frac{\log_2 K}{2K}, \frac{\log_2 K}{K}\right)$

TABLE 1. Advice complexities of PAGING and DIFFSERV problems compared with the competitive ratio. Parameter K in the PAGING and DIFFSERV problems is the number of pages that fit into physical memory, and the size of the input buffer, respectively. The intervals represent lower and upper bounds. The results are asymptotics taken for large K.

To model the impact of the timing of the communication, let the algorithm work in a synchronous setting: in the ith step, it receives the ith input request x_i, and possibly some advice a_i, based on which it produces the output y_i. In a manner usual in the synchronous distributed algorithms (see *e.g.* [26] and references therein) we count the number of bits communicated between the oracle and the algorithm, relying upon the timing mechanism for delimiting both input and advice sequences[2]. We show that these two modes are different, but are related by $B_H(\mathcal{P}) \leq B_A(\mathcal{P}) \leq 0.86 + B_H(\mathcal{P})$ where $B_H(\mathcal{P})$ is the advice complexity of a problem $\mathcal{P}$ in the helper mode, $B_A(\mathcal{P})$ is the complexity in the answerer mode. Moreover, we analyze two well-studied online problems from the point of view of advice complexity, obtaining the results shown in Table 1. To conclude this section we note that there has been a significant amount of research devoted to developing alternative complexity measures for online problems. The competitive ratio has been criticized for not being able to distinguish algorithms with quite different behavior on practical instances, and giving too pessimistic bounds [17]. Hence, several modifications of competitive ratio have been proposed, either tailored to some particular problems (*e.g.* loose competitiveness [32]), or usable in a more general setting. Among the more general models, many forms of *resource augmentation* have been studied (*e.g.* [9,19,25]). The common idea of these approaches is to counterbalance the lack of information about the input by granting more resources to the online algorithm (*e.g.* by comparing the optimal offline algorithm to an online algorithm that works k-times faster). Another approach was to use a *look-ahead* where the online algorithm is allowed to see some limited number of future requests [2,5,19,30]. The main problem with the look-ahead approach is that a look-ahead of constant size generally does not improve the worst case performance measured by the competitive ratio. Yet another approach is based on not comparing the online algorithms to offline ones, but to other online algorithms instead (*e.g.* Max/Max ratio [5], relative worst-order ratio [10]; see also [8,12]). Still another approach is to limit the power of the adversary as *e.g.* in the access

[2]Alternatively, we might require that both the input requests, and the oracle advices come in a self-delimited form as discussed later.

graph model [7,18], statistical adversary model [27], diffuse adversary model [22], etc.

Finally, a somewhat similar approach of measuring the complexity of a problem by the amount of additional information needed to solve it has been recently pursued in a different setting by Fraigniaud *et al.* [13–15].

2. Definitions and preliminaries

Unless stated otherwise all logarithms in the paper are considered in base 2. For the technical parts we shall use the following inequality which comes from the Stirling formula:

Claim 2.1. For each $n \geq 4$ it holds

$$\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n < n! < 1.05 \cdot \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n.$$

Proof. The well known Stirling formula leads in a straightforward way to this double inequality [28]:

$$\sqrt{2\pi}n^{n+\frac{1}{2}}\mathrm{e}^{-n+\frac{1}{12n+1}} < n! < \sqrt{2\pi}n^{n+\frac{1}{2}}\mathrm{e}^{-n+\frac{1}{12n}}$$

which in turn gives the statement of the claim, since for $n = 4$, $\mathrm{e}^{\frac{1}{12n}} \approx 1.021051862$, and it decreases with increasing n. □

An online algorithm receives the input incrementally, one piece at a time. In response to each input portion, the algorithm has to produce output, not knowing the future input. Formally, an online algorithm is modeled by a *request-answer* game [6]:

Definition 2.1. Consider an input sequence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n\rangle$. An *online algorithm* $\mathcal{A}$ computes the output sequence $\boldsymbol{y} = \mathcal{A}(\boldsymbol{x}) = \langle y_1, y_2, \ldots, y_n\rangle$, where $y_i = f(x_1, \ldots, x_i)$ for some function f. The *cost* of the solution is given by a function $C_{\mathcal{A}}(\boldsymbol{x}) = COST(\boldsymbol{y})$ where $COST$ is a function measuring the cost of a given output vector.

In the competitive analysis, the online algorithm $\mathcal{A}$ is compared with an optimal offline algorithm OPT, which knows the whole input in advance (*i.e.* $\boldsymbol{y} = f(\boldsymbol{x})$) and can process it optimally. The standard measure of the quality of an algorithm $\mathcal{A}$ is the competitive ratio:

Definition 2.2. An online algorithm is *c-competitive*, if for each input sequence $\boldsymbol{x}$, $C_{\mathcal{A}}(\boldsymbol{x}) \leq c \cdot C_{OPT}(\boldsymbol{x})$.

Let us suppose that the algorithm $\mathcal{A}$ is equipped with an oracle $\mathcal{O}$, which knows $\mathcal{A}$, can see the whole input, and can communicate with $\mathcal{A}$. We shall study pairs $(\mathcal{A}, \mathcal{O})$ such that the algorithm (with the help of the oracle) solves the problem optimally. We are interested in the minimal amount of communication between $\mathcal{A}$ and $\mathcal{O}$, needed to achieve the optimality.

We distinguish two modes of communication: the *helper mode*, and the *answerer mode*. In the helper mode, the oracle (helper) sends in each step i a binary *advice* string $\boldsymbol{a_i}$ (possibly empty), thus incurring a communication cost of $|\boldsymbol{a_i}|$. $\mathcal{A}$ can use this advice, together with the input $x_1, \ldots, x_i$ to produce the output y_i.

Definition 2.3 (Online algorithm with a helper)**.** Consider an online algorithm $\mathcal{A}$, an input sequence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n\rangle$, and a *helper sequence* $\mathcal{O}(\boldsymbol{x}) = \langle \boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_n}\rangle$ of binary strings $\boldsymbol{a_i}$. The *online algorithm with helper* $(\mathcal{A}, \mathcal{O})$ computes the output sequence $\boldsymbol{y} = \langle y_1, y_2, \ldots, y_n\rangle$, where $y_i = f(x_1, \ldots, x_i, \boldsymbol{a_1}, \ldots, \boldsymbol{a_i})$. The *cost* of the solution is $C_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = COST(\boldsymbol{y})$, and the *advice (bit) complexity* is

$$B^H_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = \sum_{i=1}^{n} |\boldsymbol{a_i}|.$$

In the answerer mode, on the other hand, the oracle is allowed to send an advice only when asked by the algorithm. However, this advice must be a non-empty string. For the ease of presentation we define the answerer oracle as a sequence of non-empty strings. However, only those strings requested by the algorithm are ever considered.

Definition 2.4 (Online algorithm with an answerer)**.** Consider an algorithm $\mathcal{A}$, an input sequence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n\rangle$, and an *answerer sequence* $\mathcal{O}(\boldsymbol{x}) = \langle \boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_n}\rangle$ of non-empty binary strings $\boldsymbol{a_i}$. The *online algorithm with answerer* $(\mathcal{A}, \mathcal{O})$ computes the output sequence $\boldsymbol{y} = \langle y_1, y_2, \ldots, y_n\rangle$ as follows:

(1) In each step i, a query $r_i \in \{0, 1\}$ is generated first as a function of previous inputs and advices, *i.e.* $r_i = f_r(x_1, \ldots, x_i, r_1 \star \boldsymbol{a_1}, \ldots, r_{i-1} \star \boldsymbol{a_{i-1}})$, where the function "$\star$" is defined

$$c \star \alpha = \begin{cases} \textit{empty string} & \text{if } c = 0, \\ \alpha & \text{otherwise.} \end{cases}$$

(2) Then, the output is computed as $y_i = f(x_1, \ldots, x_i, r_1 \star \boldsymbol{a_1}, \ldots, r_i \star \boldsymbol{a_i})$.

The *cost* of the solution is $C_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = COST(\boldsymbol{y})$, and the *advice (bit) complexity* is

$$B^A_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = \sum_{i=1}^{n} |r_i \star \boldsymbol{a_i}|.$$

In the sequel we shall denote the communication modes as H for helper, and A for answerer. As already mentioned, we are interested in the minimal amount of information the algorithm must get from the oracle, in order to be optimal. For an algorithm $\mathcal{A}$ with an oracle (helper or answerer) $\mathcal{O}$, the communication cost is the worst case bit complexity, amortized per one step:

Definition 2.5. Consider an online algorithm $\mathcal{A}$ with an oracle $\mathcal{O}$ using communication mode $M \in \{H, A\}$. The *bit complexity of the algorithm* is

$$B^M_{(\mathcal{A},\mathcal{O})} = \limsup_{n \mapsto \infty} \max_{|\boldsymbol{x}|=n} \frac{B^M_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x})}{n}.$$

The advice complexity of an online problem $\mathcal{P}$ is the minimum bit complexity of an optimal pair $(\mathcal{A}, \mathcal{O})$:

Definition 2.6. Consider an online problem $\mathcal{P}$. The *advice complexity of* $\mathcal{P}$ in communication mode $M \in \{H, A\}$ is

$$B_M(\mathcal{P}) = \min_{(\mathcal{A},\mathcal{O})} B^M_{(\mathcal{A},\mathcal{O})},$$

where the minimum is taken over all $(\mathcal{A}, \mathcal{O})$ such that $\forall \boldsymbol{x} : C_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = C_{OPT}(\boldsymbol{x})$.

In the previous definitions we measure the amount of communication by simply summing up the number of communicated bits, and rely upon the timing mechanism to delimit particular (possibly empty) advice strings. In this way, which is common in the area of synchronous distributed computing, the silence has also an expressive power which can be exploited by the algorithm. In the helper model, this power is essential: indeed, the helper model measures in some way the amount of *critical decisions* made by the algorithm, *i.e.* the times where the information about the future is essential for a correct decision. In the answerer model, on the other hand, the silence itself cannot carry any information.

We start analyzing the advice complexity with an immediate observation that the answerer model is more restrictive in the following sense:

Claim 2.2. For each problem $\mathcal{P}$, $B_H(\mathcal{P}) \leq B_A(\mathcal{P}) \leq 0.86 + B_H(\mathcal{P})$.

Proof. The left-hand inequality is obvious. Consider an algorithm $\mathcal{A}$ with an answerer. The same algorithm can be used with a helper with the same bit complexity: the helper can simulate $\mathcal{A}$ and locally compute in each step, if the advice is requested. If so, it sends the advice, otherwise it sends an empty string.

For the right-hand part, the inequality $B_A(\mathcal{P}) \leq 1 + B_H(\mathcal{P})$ is easy to see: an algorithm with an answerer can ask a question in every step. The first bit of the answer indicates, if a helper would send a non-empty advice, and the rest is the actual advice as would be given by the helper.

Now we show how to reduce the constant to 0.86. Consider a fixed constant γ such that $0 < \gamma < 1/2$. In the first step, the algorithm asks a question and receives, together with the "usual" advice also some initialization information of $o(n)$ bits. First of all, this information allows the algorithm to distinguish two cases:

Case 1: *the number of empty strings sent by the helper is less than* γn *or more than* $(1-\gamma)n$.
The answerer can, in the initial information, encode the positions of the non-empty strings. Then, the algorithm can ask only in those steps, in which the helper would sent a non-empty advice. To encode the positions of non-empty strings means to encode a binary string of length n with at most γn ones or zeroes. There are

$$2\sum_{i=0}^{\gamma n} \binom{n}{i} \leq n \cdot \binom{n}{\gamma n} \approx n \cdot \left(\frac{1}{\gamma^{\gamma}(1-\gamma)^{(1-\gamma)}}\right)^n \cdot O(1)$$

such strings so that asymptotically

$$n \log \left(\frac{1}{\gamma^{\gamma}(1-\gamma)^{(1-\gamma)}} \right) + O(\log n)$$

bits are sufficient to encode the string.

Case 2: *the number of empty strings sent by the helper is between* γn *and* $(1-\gamma)n$. For a given i, let μ_i be a binary string of length i, such that it is the least frequently occurring string among the advices of length i given by the helper[3]. The initial information includes the sequence $\mu_1, \mu_2, \ldots, \mu_{\log n}$, which is $\log^2 n$ bits overall. The algorithm asks a question in every step, and receives the same answer as the helper would give with the exception that instead of an empty string, the string μ_1 is sent, and instead of μ_i, μ_{i+1} is sent. Obviously, since the string $\mu_{\log n}$ is the least frequent of all n strings of length $\log n$, it is never used by the helper, and so does not need to be re-mapped.

To bound the communication overhead, let there be x empty strings sent by the helper. These incur x additional bits in the answerer model. From the remaining $n-x$ strings, there may be at most $(n-x)/2^i$ occurrences of μ_i, thus incurring at most $(n-x)/2^i$ additional bits. Obviously, the worst case is when all non-empty strings are of length 1, in which case we get the overhead $x+(n-x)/2 = (x+n)/2$. Since $x \leq (1-\gamma)n$ the communication overhead is at most $n\left(1-\frac{\gamma}{2}\right)$.

To minimize the maximum of the two cases, we solve numerically for

$$\log \left(\frac{1}{\gamma^{\gamma}(1-\gamma)^{(1-\gamma)}} \right) = \left(1 - \frac{\gamma}{2}\right)$$

obtaining a value $\gamma \approx 0.28245$. Choosing this value gives the statement of the theorem. □

In the lower bound arguments, we shall use the notion of a *communication pattern*. Informally, a communication pattern is the entire information that the algorithm receives from the oracle. Since the algorithms are deterministic, the number of different communication patterns bounds the number of different behaviors of the algorithm on a given input. The structure of a typical lower bound is then as follows: consider an arbitrary algorithm and construct a family of inputs such that, in order to be optimal, the algorithm needs a distinct communication pattern for each input from the family. Then compare the number of the constructed inputs with the number of communication patterns with at most s bits, and argue that at least some s_0 bits are needed by the algorithm. In the rest of this section we present the definitions of communication patterns, and some technical lemmata that will be used in the next sections.

Definition 2.7 (Communication pattern – helper)**.** Consider an algorithm with helper. The *communication pattern* of an input sequence $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$ is

[3]If there is more than one such string, the choice is arbitrary but fixed.

defined as the sequence of advices given at each particular step, *i.e.* $\langle \boldsymbol{a}_1, \ldots, \boldsymbol{a}_n \rangle$, where $\boldsymbol{a}_i$ is a (possibly empty) binary string.

Obviously, the input and communication pattern completely determine the behavior of the algorithm.

Lemma 2.1. *Consider an algorithm with helper, and let the input sequence be of length $n+1$. For a fixed s, consider only communication patterns in which the helper sends in total at most s bits over all $n+1$ advices. For the number X of distinct communication patterns with this property it holds*

$$\log X \leq s\left(\log(1+\alpha)+1+\frac{1}{\ln 2}\right)+\frac{1}{2}\left[\log\left(1+\frac{1}{\alpha}\right)+\log s\right]+c$$

where $\alpha = \frac{n}{s}$, and c is some constant.

Proof. For a particular a, a communication pattern is a string of a bits, distributed among $n+1$ time slots. Hence there are

$$X = \sum_{a=0}^{s} 2^a \binom{a+n}{a} \leq s2^s \binom{s+n}{s}$$

different communication patterns that use at most s bits. Using Claim 2.1 we get

$$\begin{aligned}\log X &\leq \log s + s + \log\tbinom{s+n}{s} \leq \cdots \\ &\leq s + (s+n)\log(s+n) - s\log s - n\log n + \tfrac{1}{2}\left[\log(s+n)+\log s-\log n\right]+c\end{aligned} \tag{2.1}$$

where $c = \log\left(\frac{1.05}{\sqrt{2\pi}}\right) \approx -1.255358737$. Let, for $x > 0$, $f(x) = x\log x$, and the derivatives $f'(x) = \frac{1}{\ln 2} + \log x$, $f''(x) = \frac{1}{x \ln 2}$. Since $f(x)$ is convex, it holds

$$(s+n)\log(s+n) - n\log n = f(s+n) - f(n) \leq s \cdot f'(s+n) = s\left(\log(s+n)+\frac{1}{\ln 2}\right). \tag{2.2}$$

Combining (2.1) and (2.2), we get

$$\log X \leq s\left(1+\log(s+n)+\frac{1}{\ln 2}-\log s\right)+\frac{1}{2}\left[\log(s+n)+\log s-\log n\right]+c.$$

Denote $\alpha = \frac{n}{s}$, then the previous equation becomes

$$\log X \leq s\left(\log(1+\alpha)+1+\frac{1}{\ln 2}\right)+\frac{1}{2}\left[\log\left(1+\frac{1}{\alpha}\right)+\log s\right]+c. \qquad \square$$

The situation in the answerer mode is slightly more complicated due to the fact that answers are delivered only when requested.

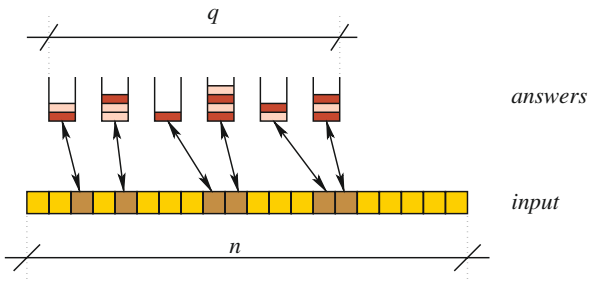

FIGURE 1. (Color online) Communication pattern mapped to input.

Definition 2.8 (Communication pattern – answerer)**.** For each execution of an algorithm with q queries to the answerer, the *communication pattern* is the sequence $\langle \boldsymbol{a}_1, \ldots, \boldsymbol{a}_q \rangle$ of non-empty answers.

The behavior of the algorithm is clearly completely determined by the input, the communication pattern and a mapping that assigns to each $\boldsymbol{a}_i$ the step j_i in which the answer was delivered (see Fig. 1).

However, this mapping bears no relevant information: for a given input and communication pattern, the algorithm always receives identical answers, and hence it also asks identical questions. We have the following:

Claim 2.3. The behavior of an algorithm with answerer is completely determined by its input and communication pattern.

Lemma 2.2. *Consider an algorithm with answerer. For a given s consider only communication patterns, in which the algorithm asks $q \leq s$ questions, and s is the total number of bits in all answers. Then there are*

$$X = \frac{1}{3}\left(2^{2s+1} + 1\right)$$

different communication patterns with this property.

Proof. Since each answer is non-empty, there are

$$2^a \cdot \binom{a-1}{q-1}$$

possible communication patterns with q questions and exactly a bits[4]. So there are

$$X = 1 + \sum_{q=1}^{s}\sum_{a=q}^{s} 2^a \cdot \binom{a-1}{q-1} = 1 + \sum_{a=1}^{s} 2^a \cdot \sum_{q=0}^{a-1}\binom{a-1}{q} = 1 + \sum_{a=1}^{s} 2^{2a-1} = \frac{1}{3}\left(2^{2s+1}+1\right)$$

possible communication patterns. □

[4]In order to cut a string of a bits into q non-empty parts $q-1$ cuts must be made in some of the $a-1$ positions.

In the rest of the paper we assume that the algorithm knows the length of the input. Indeed, it is always possible to alter the oracle in such a way that it sends the length of the input[5] in the first step. Since there are $O(\log n)$ additional bits sent, the normalized contribution to one request is $O(\log n/n)$ which is asymptotically zero.

3. Paging

Paging and its many variants belong to the classical online problems. The virtual memory of a computer is divided into logical pages. At any time K logical pages can reside in the physical memory. A paging algorithm is the part of the operating system responsible for maintaining the physical memory. If a program requests access to a logical page that is not currently in the physical memory, a *page fault* interrupt occurs and the paging algorithm has to transfer the requested page into physical memory, possibly replacing another one. Formally, we define the paging problem as follows:

Definition 3.1 (paging problem)**.** The input is a sequence of integers (logical pages) $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$, $x_i > 0$. The algorithm maintains a buffer (physical memory) $B = \{b_1, \ldots, b_K\}$ of K integers. Upon receiving an input x_i, if $x_i \in B$, $y_i = 0$. If $x_i \notin B$ a *page fault* is generated, 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 is the number of faults, *i.e.* $COST(\boldsymbol{y}) = |\{i : y_i > 0\}|$.

It is a well known fact [29] that there is a K-competitive paging algorithm, and that K is the best attainable competitive ratio by any deterministic online algorithm. The optimal offline algorithm is due to [4]. Let us consider the advice complexity of this problem for both helper and answerer modes. We prove that for the helper mode the complexity is between 0.1775 and 0.2056, and for the answerer mode the complexity is between 0.4591 and $0.5 + \varepsilon$ bits per request. Let us first analyze the helper mode. We start with a simple algorithm that uses one bit per request:

Lemma 3.1. *Consider the* Paging *problem. There is an algorithm $\mathcal{A}$ with a helper $\mathcal{O}$, such that $\mathcal{O}$ sends an advice of exactly one bit each step.*

Proof. Consider an input sequence $\boldsymbol{x}$, and an optimal offline algorithm OPT processing it. In each step of OPT, call a page currently in the buffer *active*, if it will be requested again, before OPT replaces it by some other page. We design $\mathcal{A}$ such that in each step i, the set of OPT's active pages will be in B, and $\mathcal{A}$ will maintain with each page an *active* flag identifying this subset. If $\mathcal{A}$ gets an input x_i that causes a page fault, some passive page is replaced by x_i. Moreover, $\mathcal{A}$ gets with each input also one bit from the helper telling whether x_i is active for OPT. Since the set of active pages is the same for OPT and $\mathcal{A}$, it is immediate that $\mathcal{A}$ generates the same sequence of page faults. □

[5] In self-delimited form to distinguish it from the possible advice.

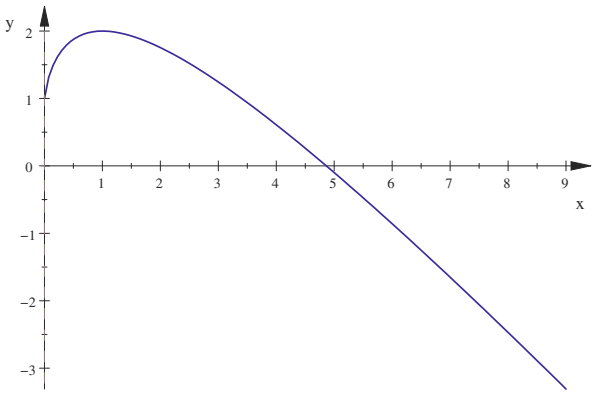

FIGURE 2. Illustration of function $f(x) = (1+x)\log(1+x) - x\log x - x + 1$.

Now we are going to further reduce the advice complexity. The algorithm shall receive basically the same advice as in Lemma 3.1; the reduction of the advice complexity is possible due to a more sophisticated encoding.

Lemma 3.2. *For r large enough, the helper can communicate a binary string of length αr using r bits over a period of αr steps, where $\alpha \approx 4.863876183$.*

Proof. Let $f(x) = (1+x)\log(1+x) - x\log x - x + 1$. $f(x)$ has exactly one positive root $x_0 \approx 4.863876183$. For any $\varepsilon > 0$, let $\alpha = x_0 - \varepsilon$ and consider, for the sake of clarity, that αr is integer.

There are

$$X = 2^r \binom{\alpha r + r - 1}{r} = 2^r \frac{\alpha}{\alpha+1} \binom{r(\alpha+1)}{r}$$

possible tuples $\langle \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\alpha r} \rangle$ of advices containing r bits in total. Using Claim 2.1, we get

$$X \geq 2^r \frac{(1+\alpha)^{r(1+\alpha)}}{\alpha^{r\alpha}} \frac{1}{1.05^2} \frac{\alpha}{\alpha+1} \sqrt{\frac{1+\alpha}{2\pi r \alpha}}.$$

We show that for large enough r it holds $\log X > \alpha r$ which means that $X > 2^{\alpha r}$. After some calculations we get

$$\log X - \alpha r \geq r\left((1+\alpha)\log(1+\alpha) - \alpha\log\alpha - \alpha + 1\right) - \frac{1}{2}\log r + c = r f(\alpha) - \frac{1}{2}\log r + c$$

for some constant c. Since $f(\alpha)$ is a positive constant it holds $\lim_{r\mapsto\infty} \log X - \alpha r = \infty$, and for large enough r the inequality $\log X > \alpha r$ follows. □

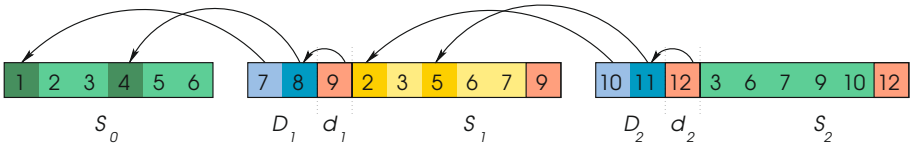

FIGURE 3. (Color online) An example of first two frames for $K = 3$, *i.e.* with buffer of size 6. The arrows indicate which pages are replaced during faults.

Theorem 3.1. $B_H(\text{PAGING}(K)) \leq \frac{1}{\alpha}$, *where* $\alpha \approx 4.863876183$.

Proof. Divide the input into frames of length $\alpha \log n$. The helper sends the first $\alpha \log n$ bits at the beginning. During ith frame, $\log n$ bits are used to communicate the string of $\alpha \log n$ for the next frame using Lemma 3.2. Overall, there are $\alpha \log n + \frac{n}{\alpha}$ bits. □

On the lower bound side, we can prove the following:

Theorem 3.2. *For every fixed* K, *there is a constant* $\alpha_K < 20.742$ *such that* $B_H(\text{PAGING}(2K)) \geq \frac{1}{\alpha_K}$. *Moreover,* α_K *is a decreasing function in* K *and* $\lim_{K \mapsto \infty} \alpha_K \approx 5.632423693$.

Proof. We shall consider a particular subset of input sequences $\boldsymbol{x} = \{x_k\}_{k=1}^{K(2+3i)}$ for some i. Each input sequence consists of the sequence $S_0 = \langle 1, 2, \ldots, 2K \rangle$ followed by i *frames*, each of length $3K$, where the jth frame has the form $D_j \cdot \langle d_j \rangle \cdot S_j$. The first part of each frame, D_j is of length $K-1$ and contains unused pages that generate page faults: $D_j = \langle (j+1)K + 1, \ldots, (j+2)K - 1 \rangle$. The next request $d_j = (j+2)K$ is again an unused page. The last part, S_j is a sequence of length $2K$ consisting of any subsequence[6] of $S_{j-1} \cdot D_j$ of length $2K - 1$, followed by d_j.

Clearly, since D_j and d_j contain values that have never been used before, any algorithm must generate at least K page faults in each frame. Moreover, there is an algorithm that generates exactly K page faults in each frame as follows. Suppose that after $3Kj$ steps the buffer contains exactly the elements from S_j. The following $2K$ steps do not generate any page faults. The next K steps generate a page fault every request, and because from among the $3K-1$ elements of $S_j \cdot D_{j+1}$ only $2K - 1$ are used in S_{j+1}, the buffer always contains at least one element not in S_{j+1}. Hence, after $3K(j+1)$ steps the buffer contains elements from S_{j+1}.

From the above reasoning it follows that no algorithm computing an optimal solution can generate a page fault in S_j, which means that at the beginning of S_j, the content of the buffer of any such algorithm is uniquely determined.

Now we show that in order to obtain an optimal solution it must hold for any two different inputs that the communication patterns of the algorithm must be different. Consider two executions of the algorithm with the same communication patterns. We claim that the inputs must have been the same, too. By contradiction, let j be the first frame in which the inputs differ. By the construction of

[6] Note that S_j is always an increasing sequence.

the input sequence, the first two components (D_j, d_j) of the frame are identical. Hence, after $3Kj$ steps (*i.e.* at the beginning of S_j) the executions are in the same state. However, since the algorithm computes an optimal solution, the buffer of both executions contains exactly S_j. Hence the jth frame is identical in both inputs – a contradiction.

For each S_j, there are $\binom{3K-1}{2K-1}$ different possible S_{j+1}, hence there are

$$Y = \binom{3K-1}{2K-1}^i = \left[\frac{2}{3}\binom{3K}{K}\right]^i$$

different inputs of length $2K + 3Ki$, and using Claim 2.1 we get

$$\log Y \geq i\left[K(3\log 3 - 2) - \frac{\log K}{2} - c_1\right] \quad \text{where} \quad c_1 = \log(1.05^2 \cdot \sqrt{3\pi}) \approx 1.759. \tag{3.1}$$

Since any algorithm computing an optimal solution needs a different communication pattern for each input, it needs at least Y different communication patterns on inputs with i frames. However, using Lemma 2.1, we get that there are at most X different communication patterns of length $n+1$ using at most s bits, where

$$\log X \leq s\left(\log(1+\alpha) + 1 + \frac{1}{\ln 2}\right) + \frac{1}{2}\left[\log\left(1 + \frac{1}{\alpha}\right) + \log s\right] + c$$

and $\alpha = n/s$. Since we may restrict our attention only to values $\alpha \geq 1$ the previous inequality becomes

$$\log X \leq s\left(\log\left(1 + \frac{n}{s}\right) + 1 + \frac{1}{\ln 2}\right) + \frac{1}{2}\log s + O(1). \tag{3.2}$$

Using (3.1), and the fact that

$$i = \frac{n + 1 - 2K}{3K}$$

we get

$$\log Y \geq \left(\frac{n}{3K} - \frac{2K-1}{3K}\right)\left[K(3\log 3 - 2) - \frac{\log K}{2} - c_1\right]. \tag{3.3}$$

Combining (3.2) and (3.3), we get that for inputs of length $n+1$ the (worst case) number of advice bits that an optimal algorithm receives is s such that

$$0 \leq \log X - \log Y \leq s\left[\log\left(1 + \frac{n}{s}\right) + C - \frac{n}{s}\kappa\right] + \frac{1}{2}\log s + O(K) \tag{3.4}$$

where

$$C = 1 + \frac{1}{\ln 2} \approx 2.443$$

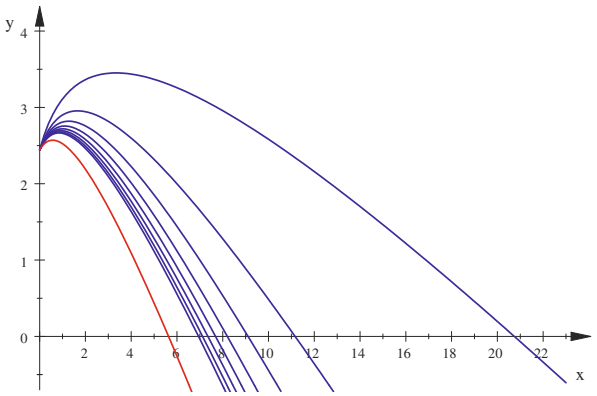

FIGURE 4. (Color online) Functions $F_K(\alpha)$ for $K = 1, \ldots, 8, 10^{100}$. The x axis represents the value of α. The roots α_K are smaller than $\alpha_1 \approx 20.741$ and converge to $\alpha_\infty \approx 5.632$.

$$\kappa = \frac{3\log 3 - 2}{3} - \frac{\log K}{6K} - \frac{c_1}{3K} \approx 0.918 - \frac{\log K}{6K} - \frac{0.586}{K}.$$

κ as a function of K is increasing and it holds $0.332 \leq \kappa \leq 0.918$. For a fixed K one can regard the right-hand side of inequality (3.4) as a function of n where $s = s(n) > 1$ is a function of n. If $s(n)$ is bounded it holds

$$\limsup_{n\mapsto\infty} \log\left(1 + \frac{n}{s}\right) + C - \frac{n}{s}\kappa = -\infty$$

contradicting the inequality (3.4). Hence $s(n)$ is unbounded, and in order to satisfy the inequality it must hold[7]

$$\liminf_{n\mapsto\infty} \log\left(1 + \frac{n}{s}\right) + C - \frac{n}{s}\kappa \geq 0.$$

Again, denote $\alpha = n/s$ and consider a function

$$F_K(\alpha) := \log(1+\alpha) + C - \alpha\kappa. \tag{3.5}$$

It holds $F_K(0) = C > 0$, and the derivative is $F'_K(\alpha) = \frac{1}{\ln 2(1+\alpha)} - \kappa$. Hence, $F_K(\alpha)$ is increasing up to some point $\alpha_{\max}$ and then decreasing, and because $0.332 \leq \kappa \leq 0.918$ there is a single positive root α_K. Moreover, since $\log(1+\alpha)+C$ is increasing, α_K is a decreasing function in κ. Because κ as a function of K is increasing, α_K is decreasing in K, and by numerically solving for α_1 we get $\alpha_K \leq 20.741$.

[7] If $\liminf_{n\mapsto\infty} \log\left(1 + \frac{n}{s}\right) + C - \frac{n}{s}\kappa = z < 0$ then, since $s(n)$ is unbounded, one can find n_0 such that for $n > n_0$ it holds $s(n)z/2 + \log s(n)/2 + O(K) < 0$, and one can find $n > n_0$ such that $\log\left(1 + \frac{n}{s}\right) + C - \frac{n}{s}\kappa < z/2$.

Since $\lim_{K\mapsto\infty}\kappa = \frac{3\log 3-2}{3}$, we have $\lim_{K\mapsto\infty}\alpha_K = \alpha_\infty$, where α_∞ is the positive root of

$$\log(1+\alpha)+1+\frac{1}{\ln 2}-\alpha\frac{3\log 3-2}{3}=0.$$

Numerically solving, we get

$$\alpha_\infty \approx 5.632423693.$$

Since for any K

$$\liminf_{n\mapsto\infty}\log\left(1+\frac{n}{s}\right)+C-\frac{n}{s}\kappa\geq 0$$

it must hold

$$\limsup_{n\mapsto\infty}\frac{n}{s}\leq\alpha_K$$

from which it follows for the bit complexity that

$$B_{\mathcal{A},\mathcal{O}}=\limsup_{n\mapsto\infty}\max_{|\boldsymbol{x}|=n}\frac{B_{\mathcal{A}}(\boldsymbol{x},\mathcal{O}(\boldsymbol{x}))}{n}\geq\limsup_{n\mapsto\infty}\frac{s}{n+1}\geq\frac{1}{\alpha_K}. \qquad \square$$

There is one issue connected with the previous proof, namely that the number of logical pages in the constructed sequence was unbounded (since every frame used new values, the number of values depended on n). We argue that this feature can be avoided by reusing the values after a constant number of frames, since every optimal algorithm has to replace all pages from a given frame within the next three frames. Assume the contrary, and consider an optimal algorithm that leaves a page p from a frame i in the buffer during frames $i+1$ and $i+2$. Obviously, there must be at least $K+1$ faults in both of them. However, having p in the buffer for frame $i+3$ can save at most one page fault.

Let us proceed now with the analysis of the answerer mode. First, we give an upper bound by refining Lemma 3.1:

Theorem 3.3. *For each $\varepsilon>0$, $B_A(\textsc{Paging}(K))\leq\frac{1}{2}+\varepsilon$.*

Proof. Let $k=\frac{2}{2\varepsilon+1}$. Without loss of generality, let $n=2ks$. Following Lemma 3.1, there is an optimal helper algorithm which receives 1 bit of advice each request. Obviously, the same algorithm (with a question posed in every step) works also in the answerer mode. We show how to supply this information using $2s$ bits.

In the first step, the algorithm asks, and receives $O(\log n)$ bits of information containing s and n. Amortized, these $O(\log n)$ bits will contribute $O(\frac{\log n}{n})$ bits per request, and thus can be neglected.

During each of the first s steps, the algorithm asks, and receives a non-empty string of advice. Let $\langle \boldsymbol{a}_1,\ldots,\boldsymbol{a}_s\rangle$ be the s-tuple of advices, such that $\sum_{i=1}^{s}|\boldsymbol{a}_i|=2s$. Because each $\boldsymbol{a}_i$ is non-empty, the concatenation of $\boldsymbol{a}_i$'s provides one bit every step for the first $2s$ steps. The remaining $n-2s$ bits are encoded in the lengths of $\boldsymbol{a}_i$'s as follows: consider all s-tuples of integers $\langle i_1,\ldots,i_s\rangle$ such that $\sum_{j=1}^{s}i_j=s$, ordered lexicographically. Let z be the number of the tuple $\langle|\boldsymbol{a}_1|-1,\ldots,|\boldsymbol{a}_s|-1\rangle$

in this ordering, then the representation of z as a $(n-2s)$-bit-long binary string gives the remaining $n-2s$ bits.

What remains to be shown is that there are enough possible combinations of lengths to encode all 2^{n-2s} possible binary strings for the last $n-2s=2s(k-1)$ steps. There are

$$X=\binom{2s-1}{s}=\frac{1}{2}\binom{2s}{s}$$

possible s-tuples $\langle i_1,\ldots,i_s\rangle$ such that $\sum_{j=1}^{s} i_j=s$. Using Claim 2.1 it follows

$$X\geq 2^{2s}\frac{1}{2\times 1.05^2\cdot\sqrt{\pi s}}$$

for s large enough, it holds

$$X\geq 2^{2s}\frac{1}{2^{s\frac{8\varepsilon}{2\varepsilon+1}}}=2^{2s\left(1-\frac{4\varepsilon}{2\varepsilon+1}\right)}=2^{2s(k-1)}.$$

The bit complexity of the algorithm is

$$B_{\mathcal{A},\mathcal{O}}=\limsup_{n\mapsto\infty}\max_{|\boldsymbol{x}|=n}\frac{B_{\mathcal{A}}(\boldsymbol{x},\mathcal{O}(\boldsymbol{x}))}{n}=\lim_{s\mapsto\infty}\frac{2s}{2ks}=\frac{1}{2}+\varepsilon. \qquad \square$$

To conclude this section, the same technique as used in Theorem 3.2 can be employed to deliver the corresponding lower bound:

Theorem 3.4. $B_A(\textsc{Paging}(2K))\geq 0.4591-O\left(\frac{\log K}{K}\right)$.

Proof. Consider the same input sequences as in the proof of Theorem 3.2. For each S_j, there are $\binom{3K-1}{2K-1}$ different possible S_{j+1}, hence there are

$$Y=\binom{3K-1}{2K-1}^i=\left[\frac{2}{3}\binom{3K}{K}\right]^i$$

different inputs of length $2K+3Ki$.

Let the answerer use s bits overall. Using Lemma 2.2, there are

$$X=\frac{1}{3}\left(2^{2s+1}+1\right)$$

possible communication patterns, and obviously it must hold $X\geq Y$. Using Claim 2.1 we get that

$$\log Y=i\log\left[\frac{2}{3}\binom{3K}{K}\right]\geq i\log\left[\frac{3^{3K}}{2^{2K}\sqrt{K}}\frac{1}{1.05^2\cdot\sqrt{3\pi}}\right]$$
$$=i\left[K(3\log 3-2)-\frac{\log K}{2}-c_1\right]$$

for some constant c_1. Because

$$\log X \leq 2s + c_2 \quad \text{for some constant } c_2$$

from the fact that $\log X \geq \log Y$ it follows that

$$s \geq i \left[K \frac{3\log 3 - 2}{2} - \frac{\log K}{4} - \frac{c_1}{2} \right] - \frac{c_2}{2}.$$

The bit complexity of any optimal algorithm is then

$$\begin{aligned} B_{\mathcal{A},\mathcal{O}} &= \limsup_{n\mapsto\infty} \max_{|\boldsymbol{x}|=n} \frac{B_{\mathcal{A}}(\boldsymbol{x}, \mathcal{O}(\boldsymbol{x}))}{n} \geq \lim_{i\mapsto\infty} \frac{i\left[K\frac{3\log 3-2}{2} - \frac{\log K}{4} - \frac{c_1}{2}\right] - \frac{c_2}{2}}{2K + 3Ki} \\ &= \frac{3\log 3 - 2}{6} - \frac{1}{12}\frac{\log K}{K} - \frac{c_1}{6K} \end{aligned}$$

□

4. DiffServ

DiffServ is another problem widely studied using competitive analysis (see [11,23] and references therein). The setting involves a server processing an incoming stream of packets of various values. If the processing speed of the server is slower than the arrival rate, some packets must be dropped, ideally those least valuable. For our purposes, following [23], the packets arrive in discrete time steps. In each step a number of packets can arrive, one packet can be processed, and at most K packets can be stored in a buffer. Moreover, it is required that the packets are processed in FIFO manner. The formal definition is as follows:

Definition 4.1 (DiffServ problem)**.** Consider a sequence of items $\langle p_1, \ldots, p_m \rangle$, partitioned into a series of subsequences, called requests. The input is the sequence of requests $\boldsymbol{x} = \langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \rangle$, where each $\boldsymbol{x}_i = \langle p_{j_{i-1}+1}, \ldots, p_{j_i} \rangle$ is a (possibly empty) request. Each item p_i has a value $v(p_i)$. In each step i, the algorithm maintains an ordered buffer $B_i = \langle b_1, \ldots, b_{K_i} \rangle$ of $K_i \leq K$ items. Upon receiving a request sequence $\boldsymbol{x}_i$, the algorithm *discards* some elements from the sequence $B_i \cdot \boldsymbol{x}_i$, keeping the remaining subsequence $B'_i \preceq B_i \cdot \boldsymbol{x}_i$ of length at most $K + 1$. The first item of B'_i (if B'_i is nonempty) is *submitted*, and the remainder of B'_i forms the new buffer, *i.e.* $B'_i = y_i \cdot B_{i+1}$. When there are no more requests, the contents of the buffer are submitted, and the process ends.

The cost of the solution is the sum of the values of all submitted elements, *i.e.* $COST(\boldsymbol{y}) = \sum_{i>0} v(y_i)$.

For the remainder of this section we shall consider only the case of two distinct item values; we shall refer to them as heavy and light items. Without loss of generality we may assume that each request contains at most $K + 1$ heavy items.

It was shown by Lotker and Patt-Shamir [23] that Algorithm 1 is optimal for the DiffServ problem. We first present another optimal offline algorithm, and then show how to transform it to an online algorithm with a helper.

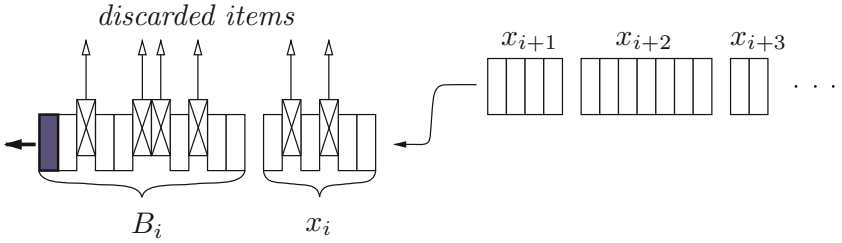

FIGURE 5. A schematic representation of the request processing in the DIFFSERV problem.

Algorithm 1 OPT

1: sort all items by decreasing value, within each value class by increasing time
2: $S \leftarrow$ empty schedule
3: **while** the list is non empty **do**
4: $p \leftarrow$ head of the sorted list
5: **if** $S \cup \{p\}$ is a feasible schedule **then** $S \leftarrow S \cup \{p\}$
6: remove head of the list
7: output S

Let us start with a simple greedy algorithm that never discards more items than necessary (Algorithm 2 without line 4). This algorithm is not optimal in situations where it is favorable to discard leading light items even if the buffer would not be filled: Consider *e.g.* a situation with a buffer of size 3 such that $B_i \cdot \boldsymbol{x}_i$ contains one light item and two heavy items in some step i. If there are no more requests, the best solution is to submit the three items in the steps $i, i+1, i+2$. However, if there is another request x_{i+1}, containing three heavy items, the best solution is to discard the leading light item in step i and submit a heavy item instead (thus leaving B_{i+1} containing just one heavy item), and submit four heavy items in steps $i+1, \ldots, i+4$. The greedy algorithm behaves the same way in both situations, *i.e.* submits the leading light item from $B_i \cdot \boldsymbol{x}_i$; in the second scenario is therefore forced to discard a heavy item in the next step.

However, the situations in which greedy strategy fails can easily be recognized:

Definition 4.2. Consider a buffer B at time t_0 and the remainder $\boldsymbol{x} = \{\boldsymbol{x}_{t_0+i}\}_{i=1}^{n-t_0}$ of the input sequence. Let a_0 be the number of heavy elements in B (before $\boldsymbol{x}_{t_0+1}$ has arrived), and $a_i \leq K+1$ be the number of heavy elements in x_{t_0+i}. The remainder of sequence $\boldsymbol{x}$ is called *critical* (w.r.t. B), if there exists $t > 0$ such that $\sum_{i=0}^{t} a_i \geq K + t$, and for each t' such that $0 < t' \leq t$ it holds $\sum_{i=0}^{t'} a_i \geq t'$.

Informally, an input sequence is critical w.r.t. an initial buffer if the buffer gradually fills with heavy items even if the algorithm submits a heavy item in each step.

We use Algorithm 2 to process the requests.

Algorithm 2 Processing of a request $\boldsymbol{x}_i$ with a buffer B

1: $B' \leftarrow B \cdot \boldsymbol{x}_i$
2: starting from left, discard light items from B' until $|B'| = K+1$ or there are no light items left
3: **if** $|B'| > K+1$ **then** discard last $|B'| - K - 1$ (heavy) items
4: **if** *the remainder of the input sequence is critical and there are some heavy items in* B' **then** discard leading light items from B'
5: submit the first item of B' (if exists)
6: $B \leftarrow$ remainder of B'

Lemma 4.1. *Algorithm 2 computes an optimal solution to the DiffServ problem.*

Proof. First, we prove that if Algorithm 2 does not submit any item in step i, no optimal algorithm can submit anything in step i. Consider a step i such that Algorithm 2 does not submit (*i.e.* has empty buffer). Let j be the last time before i when the algorithm had full buffer (or, $j = 0$ if the buffer has never been full). We argue that during the steps $j, j+1, \ldots, i$, no items have been discarded by the algorithm. The algorithm only discards items in such a way that the length of the resulting buffer is less than K only if the input is critical. Moreover, if the buffer is shorter than K in the case of critical input, only light items are discarded. However, starting from a critical input, the buffer eventually (after t steps) fills with heavy items, and is never emptied in-between (if the buffer is shorter than K, no heavy items are discarded). Since j was the last time before i when the buffer was full, clearly there was no critical input between j and i, and hence no item was discarded (since the length of the buffer has always been strictly less than K).

Now consider an arbitrary optimal algorithm A. It is easy to see that during steps $j, j+1, \ldots, i$, the length of the buffer of A is not longer than that of Algorithm 2. Hence, A has empty buffer in step i, too.

Second, we prove that the number of submitted heavy items is optimal. Consider the ordering of the heavy items as they appear in the input sequence. Let an item p be discarded. Let S_p be the set of submitted items ordered before p. We show that it is not possible to submit all items from $S_p \cup \{p\}$. Let i be the arrival time of the item p. Because p was discarded, the buffer was full of heavy items after step i, and the algorithm submitted a heavy item. Consider the longest sequence of steps $j, j+1, \ldots, i$ such that in each step a heavy item was submitted. At the beginning of the sequence, the buffer contained no heavy elements: the input was critical, so a heavy element would have been submitted. Because in all consecutive steps, heavy items were submitted, it is not possible to submit all $S_p \cup \{p\}$ items. Hence, Algorithm 2 maintains for the heavy items the same optimality property as Algorithm 1, and the number of submitted heavy items is optimal.

We have proved that Algorithm 2 submits the maximal possible amount of heavy items, and moreover, for an optimal algorithm A, if A submits in some step, Algorithm 2 submits in this step, too. So it follows that Algorithm 2 gives the optimal cost. □

Now we turn this offline algorithm into an online algorithm with helper. We are going to simulate Algorithm 2 with an algorithm and a helper. The only place where the algorithm needs information about the future is on line 4, where the algorithm tests the criticality of the input. Clearly, one bit per request (indicating whether the input is critical or not) is sufficient to achieve optimality. However, we show that the situation in which a bit must be sent can occur at most once in every $K+1$ steps.

Theorem 4.1. $B_H(\textsc{DiffServ}(K)) \leq \frac{1}{K+1}$.

Proof. The presented algorithm will operate in two modes: *standard* and *critical*. It starts in standard mode and behaves the same way as Algorithm 2 (leaving out line 4). If it receives a bit from the helper (indicating that the current input forms a critical sequence), it switches into critical mode where it always discards leading light items and submits only heavy items. Once the buffer is full of heavy items, it switches back into standard mode. Obviously, this algorithm is optimal.

Moreover, the helper has to send a bit only when the algorithm is in standard mode, the input is critical and there are leading light items. How often a bit can be sent? Clearly, after the algorithm switches into critical mode, the buffer eventually fills with heavy items. Then there are at least K requests where there are no leading light items so no bit is sent. Hence, the bit is sent at most every $K+1$ steps. □

Using a technique similar to the proof of Theorem 3.2, we can show the following:

Theorem 4.2. *For* $K \geq 12$ *it holds* $B_H(\textsc{DiffServ}(K)) \geq \frac{1}{\gamma_K \cdot K}$, *where* $\gamma_K \leq 4.851$ *and* $\lim_{K \mapsto \infty} \gamma_K = 1$.

Before proceeding to the proof of Theorem 4.2 let us first present a technical claim that will be used in the proof.

Claim 4.1. Consider the following function

$$\Psi(K) := \frac{\ln K(K+2)\left(1 + \frac{1}{\ln \frac{K+2}{2}}\right)}{K\left[\ln(K+2) - \ln\left(2\ln\frac{K+2}{2}\right) + 1\right]}.$$

The function $\Psi(K)$ converges to 1 and for $K \geq 12$ is strictly decreasing. The value $\Psi(12) \approx 1.92479457$.

Proof of Theorem 4.2. Consider a sequence of frames of two types. Both types start with a request with one light item followed by one heavy item. The frame of type A continues with a request with $K+1$ heavy items and K empty requests. The type B frame continues with one empty request. The input consists of β frames of type A and $(c-1)\beta$ frames of type B for some $c > 1$, so the length of the input is $\beta(K+2c)$. First note that for two different inputs the communication patterns have to be different: if this is not the case consider two different inputs with the same communication pattern, and focus on the first frame where the

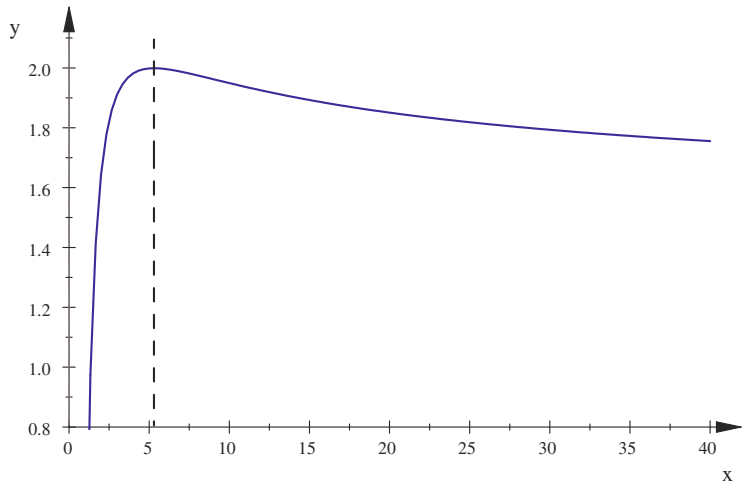

FIGURE 6. Numerical analysis of the function $y = \Psi(x)$. The maximum is attained at $K \approx 5.303720076$ and has value ≈ 1.999174149.

inputs differ. Clearly, the algorithm is in the same state in both executions and since it is deterministic, it must make the same decision. However, the optimal decision in frame A is to discard the initial light item, and submit $K + 2$ heavy items in the whole frame, whereas the only optimal decision in frame B is to submit the light and heavy item.

Hence, we have

$$Y = \frac{(c\beta)!}{\beta!\,((c-1)\beta)!}$$

different inputs of length $n + 1 = \beta(K + 2c)$, each of them requiring a different communication pattern. Using Claim 2.1 we get

$$\log Y \geq \beta\,(c\log c - (c-1)\log(c-1)) + \frac{1}{2}\,(\log c - \log(c-1)) - \frac{1}{2}\log\beta + r$$

for some constant r. Let

$$\psi_c = c\log c - (c-1)\log(c-1). \tag{4.1}$$

Since $\log c - \log(c-1) < 1$ we get $\log Y \geq \beta\psi_c - \frac{1}{2}\log\beta + r'$ for some constant r'. Expressing the parameter β in terms of the length of the input we get

$$\log Y \geq \frac{n+1}{K+2c}\psi_c - \frac{1}{2}\log\frac{n+1}{K+2c} + r'. \tag{4.2}$$

However, following Lemma 2.1 there are at most

$$X = \sum_{a=0}^{s} 2^a \binom{a+n}{a} \leq s2^s \binom{s+n}{s}$$

different communication patterns that use at most s bits and it holds

$$\log X \leq \frac{n}{\alpha}\left(\log(1+\alpha) + 1 + \frac{1}{\ln 2}\right) + \frac{1}{2}\left[\log\left(1+\frac{1}{\alpha}\right) + \log\frac{n}{\alpha}\right] + r'' \tag{4.3}$$

where $\alpha = \frac{n}{s}$ and r'' is some constant.

Since $X \geq Y$, by combining (4.2) and (4.3) we get

$$0 \leq \frac{n}{\alpha}\left[\log(1+\alpha) + 1 + \frac{1}{\ln 2} - \frac{\alpha\psi_c}{K+2c}\right] + \frac{1}{2}\left[\log n + \log(n+1) + \log(\alpha+1) - 2\log\alpha\right] + Q_{K,c}$$

where $Q_{K,c}$ does not depend on n. Further, we can consider only values $1 < \alpha < n$ in which case it holds

$$\log n + \log(n+1) + \log(\alpha+1) - 2\log\alpha < \log n(n+1)^2.$$

So the previous inequality becomes

$$0 \leq \frac{n}{\alpha}\left[\log(1+\alpha) + 1 + \frac{1}{\ln 2} - \frac{\alpha\psi_c}{K+2c}\right] + O(\log n). \tag{4.4}$$

If α (as a function of n) is $\alpha = \Omega(n)$ then, because $\alpha < n$, there exists some n_0 for which the right-hand side of (4.4) is negative. Hence, the function n/α is unbounded. Let us define the function

$$F(x) := \log(1+x) + 1 + \frac{1}{\ln 2} - x\frac{\psi_c}{K+2c}.$$

Then it holds

$$\liminf_{n\mapsto\infty} F(\alpha(n)) \geq 0$$

because otherwise there would be some n_0 violating (4.4). In the rest of the proof we find an upper bound of the form t_0K, for some t_0, on the values of x for which $F(x) \geq 0$. From that we conclude that $\limsup_{n\mapsto\infty} n/s \leq t_0K$, and that the bit complexity is at least $1/(t_0K)$.

First, we try to find a value of c such that the term $\frac{\psi_c}{K+2c}$ is maximized[8]. Let

$$h(c) := \frac{\psi_c}{K+2c} = \frac{c\log c - (c-1)\log(c-1)}{K+2c} > \frac{\log(c-1) + \frac{1}{\ln 2}}{K+2c} =: g(c).$$

[8]Note that c is an arbitrary parameter, hence we don't need to find the exact maximum, only a value where the term $\frac{\psi_c}{K+2c}$ is large enough.

The last inequality is due to the fact that $f(c) - f(c-1) > f'(c-1)$ where $f(x) = x \log x$ and $f'(x) = \log x + \frac{1}{\ln 2}$. Since

$$g'(c) = \frac{K + 2 - 2(c-1)\ln(c-1)}{\ln 2 (K+2c)^2 (c-1)}$$

the maximum is attained for

$$c - 1 = \exp\left[\mathcal{W}\left(\frac{K+2}{2}\right)\right]$$

where $\mathcal{W}$ is the Lambert function[9]. Taking into account the first two terms in the series expansion of $\mathcal{W}$, we set

$$c := \frac{K+2}{2\ln\frac{K+2}{2}} + 1.$$

Summarizing, we get

$$h(c) > \frac{\log(c-1) + \frac{1}{\ln 2}}{K + 2c} = \frac{\ln(K+2) - \ln\left(2\ln\frac{K+2}{2}\right) + 1}{\ln 2 (K+2)\left[1 + \frac{1}{\ln\frac{K+2}{2}}\right]}.$$

In the sequel, we would like to estimate the value $h(c) > \frac{\log K}{\varepsilon K}$ for some ε, so that we can easily bound $F(x)$ from above. By solving

$$\frac{\ln(K+2) - \ln\left(2\ln\frac{K+2}{2}\right) + 1}{\ln 2 (K+2)\left[1 + \frac{1}{\ln\frac{K+2}{2}}\right]} \geq \frac{\log K}{\varepsilon K}$$

for ε we get that we need to choose ε such that

$$\varepsilon \geq \frac{\ln K(K+2)\left(1 + \frac{1}{\ln\frac{K+2}{2}}\right)}{K\left[\ln(K+2) - \ln\left(2\ln\frac{K+2}{2}\right) + 1\right]} = \Psi(K).$$

Now, since $h(c) > \frac{\log K}{\varepsilon K}$, we can write

$$0 \leq F(\alpha) < \log(1+\alpha) + 1 + \frac{1}{\ln 2} - \frac{\alpha \log K}{\varepsilon K}.$$

[9] The Lambert function $\mathcal{W}$ is the inverse function of $y \mapsto y \cdot e^y$, *i.e.* $\mathcal{W}(x) \cdot e^{\mathcal{W}(x)} = x$. The asymptotics is given by

$$\mathcal{W}(x) = \ln x - \ln\ln x + \frac{\ln\ln x}{\ln x} - \frac{\ln\ln x - \frac{(\ln\ln x)^2}{2}}{(\ln x)^2} + O\left(\ln\left(\frac{1}{x}\right)^{-3}\right).$$

In the sequel we shall bound the value α from above. First let us substitute $\alpha = t \cdot K$ for some $t > 1$ and consider a function

$$G(t) := \log(1 + tK) + 1 + \frac{1}{\ln 2} - \frac{t}{\varepsilon} \log K.$$

Since the derivative is

$$G'(t) = \frac{K}{\ln 2(tK + 1)} - \frac{\log K}{\varepsilon}$$

and $G(0) > 0$, we have that $G(t)$ is increasing up to some point and then monotonically decreasing into $-\infty$. Hence, if we find any t_0 such that $G(t_0) < 0$, if must hold that $\alpha < t_0 K$.

Using the fact that $\log(1 + x) \leq \log x + \frac{1}{x \ln 2}$ we compute

$$G(t) \leq \log t + \log K \left(1 - \frac{t}{\varepsilon}\right) + \frac{1}{\ln 2} \left(\frac{1}{tK} + 1\right) + 1$$

and, in order to ensure $G(t_0) < 0$, we want to find t_0 such that

$$t_0 \geq \frac{\varepsilon}{\ln K} \ln t_0 + \varepsilon \left[\frac{1}{\ln K} \left(1 + \frac{1}{t_0 K}\right) + \frac{1}{\log K} + 1\right].$$

Since $t_0 > 1$, it is sufficient to find t_0 such that

$$t_0 \geq P \ln t_0 + Q$$

where

$$P = \frac{\varepsilon}{\ln K} \quad \text{and} \quad Q = \varepsilon \left[\frac{1}{\ln K} \left(1 + \frac{1}{K}\right) + \frac{1}{\log K} + 1\right].$$

It holds

$$2P + Q = \varepsilon \left[\frac{1}{\ln K} \left(3 + \frac{1}{K}\right) + \frac{1}{\log K} + 1\right].$$

Since $K \geq 12$ and $\varepsilon < 1.92479457$ it holds

$$2P + Q < 4.851.$$

Let $t_0 = 2P + Q$; we verify that

$$2P + Q \geq P \ln(2P + Q) + Q.$$

From this we know that $\alpha < (2P + Q)K < 4.851K$, and that the bit complexity is at least $1/\alpha$. Moreover the value of t_0 is a decreasing function in K which converges to 1. □

In a similar fashion, the following results can be shown for the answerer mode:

Theorem 4.3. $B_A(\text{DIFFSERV}(K)) \leq \frac{1+\log(K+1)}{K+1}$.

Proof. Consider the helper algorithm from Theorem 4.1, in which the helper sends one bit at certain time intervals that are at least $K+1$ steps apart. Modify the algorithm in such a way that, initially, the algorithm asks in which step t_1 the helper would send a bit for the first time. In step t_1 the algorithm asks for the next step t_2 in which the helper would send the next bit etc. Let $t_0 = 0$ and $\delta_i = t_i - t_{i-1}$. Then clearly the number of communicated bits is

$$B_n \leq \sum_{i=1}^{r} \lceil \log \delta_i \rceil \leq r + \sum_{i=1}^{r} \log \delta_i$$

where r is the number of "critical decisions"[10]. Clearly, it holds

$$\sum_{i=1}^{r} \delta_i = n$$

and

$$r \leq \frac{n}{K+1} + 1.$$

Given these constraints, the function $\sum_{i=1}^{r} \log \delta_i$ is maximized when all δ_i are equal, and r is biggest possible. The maximum value is

$$B_n \leq r + r \log \frac{n}{r} \leq \left(1 + \frac{n}{K+1}\right)\left(1 + \log \frac{n(K+1)}{n+K+1}\right)$$

where the last inequality comes from the fact that function $f(r) = r + r \log \frac{n}{r}$ is increasing for $r \leq 2n/e$ (the derivative is $f'(r) = \frac{1}{\ln 2}(\ln(2n/r) - 1)$), and that $r \leq n$. Taking the limit $\lim_{n \mapsto \infty} B_n/n$, the proof is concluded. □

Theorem 4.4. *For each fixed $K \geq 12$ there exists a $\gamma_K \leq 3.5044$ such that $B_A(\text{DIFFSERV}(K)) \geq \frac{\log(K+2)}{\gamma_K(K+2)}$. Moreover $\lim_{K \mapsto \infty} \gamma_K = 2$.*

Proof. By copying the proof of Theorem 4.2, and using Lemma 2.2 we get

$$0 \leq \log X - \log Y \leq \frac{n}{\alpha}\left[2 - \frac{\alpha \psi_c}{K+2c}\right] + O(\log n) + Q_{K,c}.$$

So it must hold

$$\alpha \leq \frac{2(K+2c)}{\psi_c}.$$

Minimizing the same way as in Theorem 4.2 we set

$$c := \frac{K+2}{2 \ln \frac{K+2}{2}} + 1$$

[10] In order to simplify things let us consider also the end of input as a critical decision point.

from which follows

$$\alpha \leq \frac{2\left(K + \frac{K+2}{\ln\left(\frac{K+2}{2}\right)} + 2\right)}{\log(K+2) - \log\left(\ln\left(\frac{K+2}{2}\right)\right) + \frac{1}{\ln 2} - 1} \leq \frac{\gamma_K(K+2)}{\log(K+2)}$$

where

$$\gamma_K = \frac{2\left(1 + \frac{1}{\ln\frac{x+2}{2}}\right)}{1 - \frac{\ln\left(\ln\left(\frac{x+2}{2}\right)\right)}{\ln(x+2)} + \frac{\frac{1}{\ln 2}-1}{\log(x+2)}}.$$

Since for $K \geq 12$ the function γ_K is decreasing, it holds $\gamma_K \leq \gamma_{12} \approx 3.504340818$. □

5. Conclusion

We have proposed a new way to evaluate online problems, based on the communication complexity. While the competitive analysis is an algorithmic measure evaluating the output quality degradation incurred by the requirements to produce the output online, our measure is a structural one quantifying the amount of additional information about the input needed to produce optimal output in an online fashion. The study of the relation between those two measures can lead to a deeper understanding of the nature of online problems. We have shown that there are problems like Paging and DiffServ where the advice complexity (in the helper mode) is proportional to the competitive ratio. On the other hand, there are problems with simple structure like SkiRental [21], which has competitive ratio $2 - \varepsilon$, but a single bit of information is sufficient to solve the problem optimally (*i.e.* it has zero advice complexity).

Studying advice complexity of a problem can lead to exposure of the critical decisions to be made (like in Algorithm 2 for DiffServ) and subsequently to better understanding of the problem and possibly more efficient algorithms. Moreover, we expect that in certain situations involving cooperating devices of uneven computational power communicating over a costly medium (as *e.g.* in sensor networks), the advice complexity might be of practical interest.

The proposed topic presents a number of intriguing open questions. Is it, for example, possible to characterize a class of problems where the competitive ratio is proportional to the advice complexity? Another whole research area is to study the tradeoff between the amount of communicated information and the achieved competitive ratio.

There is also a number of variations of the model that could be investigated. One potential modification would be to limit the size of advice given in one step. In our model this size is unbounded, and this fact is heavily relied upon (sending the length of the input in one step). However, for modelling potentially infinite inputs it would be more appealing to limit the size of advice to be independent of the input size.

References

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

[2] S. Albers, On the influence of lookahead in competitive paging algorithms. *Algorithmica* **18** (1997) 283–305.

[3] S. Albers, Online algorithms: A survey. *Math. Prog.* **97** (2003) 3–26.

[4] L.A. Belady, A study of replacement algorithms for virtual storage computers. *IBM Systems Journal* **5** (1966) 78–101.

[5] S. Ben-David and A. Borodin, A new measure for the study of on-line algorithms. *Algorithmica* **11** (1994) 73–91.

[6] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press (1998).

[7] A. Borodin, S. Irani, P. Raghavan and B. Schieber, Competitive paging with locality of reference. In *Proc. 23rd Annual ACM Symposium on Theory of Computing* (1991) 249–259.

[8] J. Boyar, M.R. Ehmsen and K.S. Larsen, Theoretical Evidence for the superiority of LRU-2 over LRU for the paging problem. In *Fourth Workshop on Approximation on Online Algorithms*. Lecture Notes Comput. Sci. **4368** (2006) 95–107.

[9] J. Boyar, K.S. Larsen and M.N. Nielsen, The accommodating function: a generalization of the competitive ratio. *SIAM J. Comput.* **31** (2001) 233–258.

[10] J. Boyar and L.M. Favrholdt, The relative worst order ratio for online algorithms, *Algorithms and Complexity, 5th Italian Conference, CIAC 2003, Rome, Italy*. Lect. Notes Comput. Sci. **2653** (2003) 58–69.

[11] M. Englert and M. Westermann, lower and upper bounds on FIFO buffer management in QoS switches, In *Proc. ESA 2006*. Lect. Notes Comput. Sci. **4168** (2006) 352–363.

[12] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator and N.E. Young, Competitive paging algorithms. *J. Algorithms* **12** (1991) 685–699.

[13] P. Fraigniaud, C. Gavoille, D. Ilcinkas and A. Pelc, Distributed computing with advice: information sensitivity of graph coloring. In *Proc. 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)* (2007).

[14] P. Fraigniaud, D. Ilcinkas and A. Pelc, Tree exploration with an oracle. In *Proc. 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006)*. Lect. Notes Comput. Sci. **4162** (2006) 24–37.

[15] P. Fraigniaud, D. Ilcinkas and A. Pelc, Oracle size: a new measure of difficulty for communication problems. In *Proc. 25th Ann. ACM Symposium on Principles of Distributed Computing (PODC 2006)* (2006) 179–187.

[16] R.L. Graham, Bounds for certain multiprocessing anomalies. *Bell Systems Technical Journal* **45** (1966) 1563–1581.

[17] S. Irany and A.R. Karlin, Online computation. In *Approximation Algorithms for NP-Hard Problems*, D.S. Hochbaum, Ed. PWS Publishing Company (1997) 521–564.

[18] S. Irani, A.R. Karlin and S. Phillips, Strongly competitive algorithms for paging with locality of reference. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms* (1992) 228–236.

[19] B. Kalyanasundaram and K. Pruhs, Speed is as Powerful as Clairvoyance. *IEEE Symposium on Foundations of Computer Science* (1995) 214–221.

[20] A.R. Karlin, M.S. Manasse, L. Rudolph and D.D. Sleator, Competitive Snoopy Caching. *Algorithmica* **3** (1988) 79–119.

[21] R. Karp, On-line algorithms versus off-line algorithms: how much is it worth to know the future? *Proc. IFIP 12th World Computer Congress* **1** (1992) 416–429.

[22] E. Koutsoupias and C.H. Papadimitriou, Beyond competitive analysis. In *Proc. 34th Annual Symposium on Foundations of Computer Science* (1994) 394–400.

[23] Z. Lotker and B. Patt-Shamir, Nearly optimal FIFO buffer management for DiffServ. *PODC* **2002** (2002) 134–143.

[24] M.M. Manasse, L.A. McGeoch and D.D. Sleator, Competitive Algorithms for Online Problems. In *Proc. 20th Annual Symposium on the Theory of Computing* (1988) 322–333.
[25] C.A. Philips, C. Stein, E. Torng and J. Wein, Optimal time-critical scheduling via resource augmentation. In *Proc. 29th Annual ACM Symposium on the Theory of Computing* (1997) 140–149.
[26] U.M. O'Reilly and N. Santoro, The expressiveness of silence: tight bounds for synchronous communication of information using bits and silence. In *Proc. 18th International Workshop on Graph-Theoretic Concepts in Computer Science* (1992) 321–332.
[27] P. Raghavan, A statistical adversary for on-line algorithms. In *On-Line Algorithms*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science (1991) 79–83.
[28] H. Robbins, A Remark of Stirling's Formula. *Amer. Math. Month.* **62** (1955) 26–29.
[29] D.D. Sleator and R.E. Tarjan, Amortized efficiency of update and paging rules. *Commun. ACM* **28** (1985) 202–208.
[30] E. Torng, A Unified Analysis of Paging and Caching. *Algorithmica* **20** (1998) 175–200.
[31] N. Young, On-line paging against adversially biased random inputs. *J. Algorithms* **37** (2000) 218–235.
[32] N. Young, The k-server dual and loose competitiveness for paging. *Algorithmica* **11** (1994) 525–541.

Communicated by J. Hromkovic.
Received August 1, 2007. Accepted January 19, 2009.