How Much Information about the Future Is Needed?*

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

[1] Institute of Mathematics,
Slovak Academy of Sciences
Stefan.Dobrev@savba.sk
[2] Department of Computer Science,
Comenius University, Bratislava, Slovakia
{kralovic,pardubska}@dcs.fmph.uniba.sk

Abstract. We propose a new way of characterizing the complexity of online problems. Instead of measuring the degradation of 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 between the algorithm and 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 tight bounds in both communication modes.

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 [12], and has received much attention over the years (see e.g. [1], [4]). The standard measure used for evaluating online algorithms is the competitive ratio [16], [20], i.e. the worst case ratio between the solution quality of the given online algorithm and that of the optimal offline algorithm. The competitive

* Supported by APVV-0433-06 and VEGA 1/3106/06.

V. Geffert et al. (Eds.): SOFSEM 2008, LNCS 4910, pp. 247–258, 2008.

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.

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 i-th 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 between the algorithm and 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.

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 is this price 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

[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.

algorithm may be able to process large parts of the input optimally without any advice, requiring only occasional help from the oracle.

In the paper, we define two modes of interaction with the oracle. In the *helper* mode, the algorithm itself cannot activate the oracle; instead, the oracle oversees the progress of the algorithm, and occasionally sends some pieces of advice. In the *answerer* mode the oracle remains passive, and the algorithm may, in any particular step, ask for advice.

To model the impact of the timing of the communication, let the algorithm work in a synchronous setting: in the i-th step, it receives the i-th 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. [22] 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 $B_H(\mathcal{P}) \leq B_A(\mathcal{P}) \leq 0.92 + B_H(\mathcal{P})$ 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 Figure 1. Due to space constraints some of the proofs have been omitted and can be found in the technical report [7].

	competitive ratio	helper	answerer
PAGING	K [24]	$(0.1775, 0.2056)$	$(0.4591, 0.5 + \varepsilon)$
DIFFSERV	≈ 1.281[8]	$\frac{1}{K}$	$\left(\frac{\log K}{2K}, \frac{\log K}{K}\right)$

Fig. 1. Communication complexities of some online problems compared with competitive ratio (asymptotics for large K)

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 [13]. Hence, several modifications of competitive ratio have been proposed, either tailored to some particular problems (e.g. loose competitiveness [26]), or usable in a more general setting. Among the more general models, many forms of *resource augmentation* have been studied (e.g. [15],[21]). 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 [3],[15],[25]. The

[2] Alternatively, we might require that both the input requests, and the oracle advices come in a self-delimited form. This would alter our upper bounds by a factor of at most 4, as discussed in the appendix.

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 [3], relative worst-order ratio [6]; see also [9]). Still another approach is to limit the power of the adversary as e.g. in the access graph model [5,14], statistical adversary model [23], diffuse adversary model [18], 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, Gavoille, Ilcinkas, and Pelc [10,11].

2 Definitions and Preliminaries

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 [4]:

Definition 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* $y_i = f(x_1, \ldots, x_i)$. *The* cost *of the solution is given by a function* $C_{\mathcal{A}}(\boldsymbol{x}) = COST(\boldsymbol{y})$.

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 an algorithm $\mathcal{A}$ is the competitive ratio:

Definition 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 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 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. $r_i = f_r(x_1, \ldots, x_i, r_1 \star \boldsymbol{a_1}, \ldots, r_{i-1} \star \boldsymbol{a_{i-1}})$*[3]
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 $B^A_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = \sum_{i=1}^{n} |r_i \star \boldsymbol{a_i}|$*

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 5. *Consider an online algorithm $\mathcal{A}$ with an oracle $\mathcal{O}$ using communication mode $M \in \{H, A\}$*[4]*. 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 6. *Consider a 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 $\forall \boldsymbol{x} : C_{(\mathcal{A},\mathcal{O})}(\boldsymbol{x}) = C_{OPT}(\boldsymbol{x})$*

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

Claim 1. *For each problem $\mathcal{P}$, $B_H(\mathcal{P}) \leq B_A(\mathcal{P}) \leq 0.92 + B_H(\mathcal{P})$.*

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 gives the number of different behaviors of the algorithm on a given input.

[3] The function "$\star$" is defined $c \star \alpha = \begin{cases} \textit{empty string} & \text{if } c = 0 \\ \alpha & \text{otherwise} \end{cases}$.

[4] In the description of communication modes, H stands for helper and A for answerer.

Definition 7 (Communication pattern – helper). *Consider an algorithm with helper. The* communication pattern *is 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 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. The number* X *of distinct communication patterns with this property is at most* $\log X \leq s\left(\log(1+\alpha)+1+\frac{1}{\ln 2}\right)+\frac{1}{2}\left[\log\left(1+\frac{1}{\alpha}\right)+\log s\right]+c$ *where* $\alpha=\frac{n}{s}>1$*, and* c *is some constant.*

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

Definition 8 (Communication pattern – answerer). *For each execution of an algorithm with* q *queries to the answerer, the* communication pattern *is the sequence* $\langle \boldsymbol{a}_{i_1}, \ldots, \boldsymbol{a}_{i_q} \rangle$ *of non-empty answers, where* i_j *is the step in which the* j*'th question was asked.*

The behavior of the algorithm is clearly completely determined by the input, the communication pattern and a mapping that assigns for each $\boldsymbol{a}_{i_j}$ the step j in which the answer was delivered. 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. Hence, the behavior of an algorithm with answerer is completely determined by its input and communication pattern.

Lemma 2. *Consider an algorithm with answerer. For a fixed* q*, and* $s \geq q$*, consider only communication patterns, in which the algorithm ask* q *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*[5]*.*

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[6] 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

[5] Note that the formula does not depend on q.

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

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 9 (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.* $COST(\boldsymbol{y}) = |\{y_i : y_i > 0\}|$.

It is a well known fact [24] 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 [2]. 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. *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. □

Now we are going to further reduce the advice complexity. The algorithm will still receive the required one bit for every input, however, it is possible to encode the bits in a more efficient way using larger strings as advice:

Lemma 4. *For* r *large enough, the helper can communicate a binary string of length* αr *using* r *bits over a period of* αr *steps, where* $\alpha \approx 4.863876183$.

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

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

Theorem 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* $\lim_{K \mapsto \infty} \alpha_K \approx 5.632423693$

Sketch of the 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, the next request d_j is again an unused page. The last part, S_j is a sequence of length $2K$ consisting of any subsequence of $S_{j-1} \cdot D_j$ of length $2K-1$, followed by d_j.

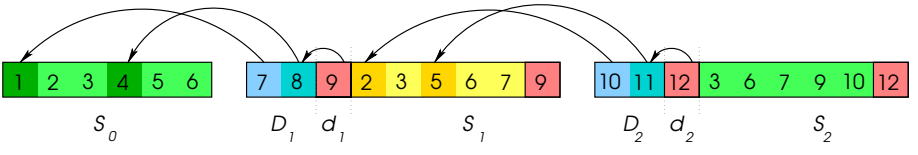

Fig. 2. An example of first two frames for $K = 3$, i.e. with buffer of size 6. The arrows indicate which pages are replaced during faults.

It is easy to see that no optimal algorithm can generate a page fault in S_j, which means that at the beginning of S_j, the content of the buffer of any optimal algorithm is uniquely determined.

Since any optimal algorithm needs a different communication pattern for each input, a simple calculation shows that there must be at least $Y = \left[\frac{2}{3}\binom{3K}{K}\right]^i$ different communication patterns. However, using Lemma 1, we get that there are at most X different communication patterns of length $n+1$ using at most s bits. Comparing these two numbers concludes after some calculations the proof. □

Let us proceed now with the analysis of the answerer mode. First, we give an upper bound by refining Lemma 3:

Theorem 3. *For each* $\varepsilon > 0$, $B_A(\textsc{Paging}(K)) \leq \frac{1}{2} + \varepsilon$

To conclude this section, the same technique as used in Theorem 2 can be employed to deliver the corresponding lower bound:

Theorem 4. $B_A(\textsc{Paging}(2K)) \geq 0.4591 - O\left(\frac{\log K}{K}\right)$

4 Diff-Serv

DiffServ is another problem widely studied using competitive analysis (see [8],[19] 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 [19], 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 10 (DIFF-SERV 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* $B_i = \langle b_1, \ldots, b_K \rangle$ *of* 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 some subsequence* $B'_i \preceq B_i \cdot \boldsymbol{x}_i$ *of length at most* $K+1$*. The first item (if* B'_i *is nonempty) of* B'_i *is* submitted*, and the remainder of the sequence forms the new buffer, i.e.* $B'_i = y_i \cdot B_{i+1}$*. The process ends if there are no more requests*[7] *and the buffer is empty.*

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. Lotker and Patt-Shamir [19] presented an optimal greedy offline algorithm. We first present another optimal offline algorithm, and then show how to transform it to an online algorithm with a helper.

Let us start with a simple greedy algorithm that never discards more items than necessary (Algorithm 1 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[8]. These situations, however, can easily be recognized:

Definition 11. *Consider a buffer* B *at time* t_0 *and the remainder* $\{\boldsymbol{x}_{t_0+i}\}_{i=1}^{n}$ *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 sequecne 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. Our algorithm processes requests sequentially. Each request is processed as shown in Algorithm 1, and it can be proven that this algorithm is optimal.

Now we turn this offline algorithm into an online algorithm with helper. We are going to simulate Algorithm 1 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,

[7] In this case some number of virtual empty requests is added until the buffer is emptied.

[8] Consider a situation with a buffer of size 3 containing one light and two heavy items. If there are no more requests, the best solution is to submit all three of them in the next three steps. However, if there is another request coming, containing two heavy items, the best solution is to discard the light one and submit heavy items in the next four steps.

Algorithm 1. 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'

we show that situation in which a bit must be sent can occur at most once in every $K + 1$ steps.

Theorem 5. $B_H(\textsc{DiffServ}(K)) \leq \frac{1}{K+1}$

Using a technique similar to the proof of Theorem 2, we can show the following:

Theorem 6. *For* $K \geq 4$ *it holds* $B_H(\textsc{DiffServ}(K)) \geq \frac{1}{\gamma_K \cdot K}$, *where* $\gamma_K \leq 6.13$ *and* $\lim_{K \mapsto \infty} \gamma_K = 1$

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

Theorem 7. $B_A(\textsc{DiffServ}(K)) \leq \frac{1+\log(K+1)}{K+1}$

Theorem 8. *For each fixed* $K \geq 4$ *there exists a* $\gamma_K \leq 3.822$ *such that* $B_A(\textsc{DiffServ}(K)) \geq \frac{\log(K+2)}{\gamma_K(K+2)}$ *Moreover* $\lim_{K \mapsto \infty} \gamma_K = 2$.

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 [17], 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 1 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. Albers, S.: Online algorithms: A survey. Mathematical Programming 97, 3–26 (2003)
2. Belady, L.A.: A study of replacement algorithms for virtual storage computers. IBM Systems Journal 5, 78–101 (1966)
3. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11(1), 73–91 (1994)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. In: Proc. 23rd Annual ACM Symp. on Theory of Computing, pp. 249–259 (1991)
6. Boyar, J., Favrholdt, L.M.: The Relative Worst Order Ratio for Online Algorithms. In: Petreschi, R., Persiano, G., Silvestri, R. (eds.) CIAC 2003. LNCS, vol. 2653, pp. 58–69. Springer, Heidelberg (2003)
7. Dobrev, S., Královič, R., Pardubská, D.: How Much Information About the Future is Needed?, Technical report TR-2007-007, Faculty of Mathematics, Physics, and Informatics, Comenius University, Bratislava, `http://kedrigern.dcs.fmph.uniba.sk/reports/display.php?id=22`
8. Englert, M., Westermann, M.: Lower and Upper Bounds on FIFO Buffer Management in QoS Switches. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 352–363. Springer, Heidelberg (2006)
9. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive Paging Algorithms. J. Algorithms 12, 685–699 (1991)
10. Fraigniaud, P., Gavoille, C., Ilcinkas, D., Pelc, A.: Distributed computing with advice: information sensitivity of graph coloring. In: Arge, L., et al. (eds.) ICALP 2007. LNCS, vol. 4596, Springer, Heidelberg (2007)
11. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: a new measure of difficulty for communication problems. In: PODC 2006. Proc. 25th Ann. ACM Symposium on Principles of Distributed Computing, pp. 179–187 (2006)
12. Graham, R.L.: Bounds for Certain Multiprocessing Anomalies. Bell Systems Technical Journal 45, 1563–1581 (1966)
13. Irany, S., Karlin, A.R.: Online Computation. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 521–564. PWS Publishing Company (1997)

14. Irani, S., Karlin, A.R., Phillips, S.: Strongly competitive algorithms for paging with locality of reference. In: Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 228–236 (1992)
15. Kalyanasundaram, B., Pruhs, K.: Speed is as Powerful as Clairvoyance. In: IEEE Symposium on Foundations of Computer Science, pp. 214–221 (1995)
16. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive Snoopy Caching. Algorithmica 3, 79–119 (1988)
17. Karp, R.: On-line algorithms versus off-line algorithms: how much is it worth to know the future? In: Proc. IFIP 12th World Computer Congress, vol. 1, pp. 416–429 (1992)
18. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proc. 34th Annual Symp. on Foundations of Computer Science, pp. 394–400 (1994)
19. Lotker, Z., Patt-Shamir, B.: Nearly Optimal FIFO Buffer Management for DiffServ. In: PODC 2002, pp. 134–143 (2002)
20. Manasse, M.M., McGeoch, L.A., Sleator, D.D.: Competitive Algorithms for Online Problems. In: Proc. 20th Annual Symposium on the Theory of Computing, pp. 322–333 (1988)
21. Philips, C.A., Stein, C., Torng, E., Wein, J.: Optimal Time-Critical Scheduling via Resource Augmentation. In: Proc. 29th Annual ACM Symp on the Theory of Computing, pp. 140–149 (1997)
22. O'Reilly, U.M., Santoro, N.: The Expressiveness of Silence: Tight Bounds for Synchronous Communication of Information Using Bits and Silence. In: Mayr, E.W. (ed.) WG 1992. LNCS, vol. 657, pp. 321–332. Springer, Heidelberg (1993)
23. Raghavan, P.: A statistical adversary for on-line algorithms. In: On-Line Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 79–83 (1991)
24. Sleator, D.D., Tarjan, R.E.: Amortized Efficiency of Update and Paging Rules. Comm. of the ACM 28(2), 202–208 (1985)
25. Torng, E.: A Unified Analysis of Paging and Caching. Algorithmica 20, 175–200 (1998)
26. Young, N.: The k-server dual and loose competitiveness for paging. Algorithmica 11, 525–541 (1994)