Information Complexity of Online Problems*

Juraj Hromkovič[1], Rastislav Královič[2], and Richard Královič[1]

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

Abstract. What is information? Frequently spoken about in many contexts, yet nobody has ever been able to define it with mathematical rigor. The best we are left with so far is the concept of entropy by Shannon, and the concept of information content of binary strings by Chaitin and Kolmogorov. While these are doubtlessly great research instruments, they are hardly helpful in measuring the amount of information contained in particular objects. In a pursuit to overcome these limitations, we propose the notion of information content of algorithmic problems. We discuss our approaches and their possible usefulness in understanding the basic concepts of informatics, namely the concept of algorithms and the concept of computational complexity.

1 Introduction

Looking at the terms "informatics" or "information processing" as reasonable alternatives to "computer science" for naming our scientific discipline, we identify two basic notions describing the fundamental stones of informatics. The first one is the notion of "algorithm", which is related to "automatic processing", and the second one is "information". The relationships of computer scientists to these two notions are different. The notion of an algorithm as a method for solving a problem was fixed in formal mathematical definitions [11], and it seems that we understand the concept of algorithms very well. The formal concept of algorithms belongs to informatics; the development of this concept is a fundamental contribution of informatics to science.

The formal definition of an algorithm can be viewed as an axiom of the formal language of mathematics; we even relate the birth of computer science to the development of this fundamental concept. The situation with the notion of "information" is completely different. This notion is not a property of informatics or a contribution of informatics to other scientific disciplines, as it is used in physics, biology, technical sciences, and many other disciplines. The crucial point is that we work intuitively with this term and yet we do not really know what "information" is. We are even very far from giving a precise mathematical definition of "information". We have the concept of entropy due to Shannon [10] and

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

the concept of Kolmogorov complexity due to Kolmogorov [9] and Chaitin [4]. Both concepts are great research instruments, but at the very end they cannot be used to estimate the information content of particular objects (binary strings).

On the other hand, computer scientists frequently speak about automatic information processing. What does it mean to solve an instance of a computing problem? To compute a solution of a problem instance is nothing else than extracting a desired information about the problem instance that is somehow hidden in the problem instance description. Remember, we cannot speak about the computational complexity of a problem without fixing the representation of the problem instances. The aim of information processing in the usual sense is to express some part of information contained in the input in a prescribed way – from this point of view, computing is a transformation of one representation of information to another one. Complexity theory indicates that the cost of these transformations may be arbitrarily high, depending on the problem. The concept of NP-completeness shows that many problems can be "equally" hard from the computational point of view, because one can efficiently transform the representation of the instances of one problem into the representation of the instances of another problem.

All this suggests that in order to advance in understanding the nature of computing, we should intensify our efforts to figure out what information really is. What about introducing a concept of "information content of algorithmic problems" in order to classify the problems in this way, and then trying to learn something essential about the computational hardness of the corresponding problems?

Another reason to focus more on studying the meaning of information is the 40-years old unsuccessful effort to prove nontrivial lower bounds on the computational complexity of concrete algorithmic problems. On the one hand, we believe that NP-hard problems require exponential time to be solved, and on the other hand we are unable to prove a superlinear lower bound on the time complexity of at least one of them. Our lower bound methods are often strongly related to "information transfer" between different parts of hardware (memory) or between different parts of computations. It seems that arguments of this kind are not strong enough to reach our goals. Understanding more about the concept of information and its role in computation could be the way out.

In this paper, we introduce the notion of "information content of algorithmic problems" with the hope that the study of this information measure could be helpful in understanding the fundamentals of computing (information processing).

2 Information Content of Online Problems

Our first attempt starts with defining the information content of a problem as a measure that does not depend on the computational hardness of the problem. Because of that, we do not start with thinking about classical offline problems. In an online setting, the algorithm receives instances of the problem piece by

piece. After getting a part of the input, the algorithm is required to make a corresponding decision with respect to the problem instance solution and is not allowed to revise decisions later after seeing a new part of the input. This means that the algorithm must generate a part of the solution with zero information about the missing parts of the problem instance. Thinking in terms of optimization problems, one defines the competitive ratio $\mathrm{cr}_{\mathtt{A}}(I)$ of an online algorithm $\mathtt{A}$ on a problem instance I as the ratio between the cost of the solution computed by $\mathtt{A}$ and the cost of the optimal solution that can be computed based on the complete information about the instance. An algorithm $\mathtt{A}$, whose competitive ratio $\mathrm{cr}_{\mathtt{A}}(I)$ is at most r for any problem instance I, is called r-competitive.

Our first idea is to define the information content $\inf_P$ of an online problem P as the minimum number of advice bits sufficient and necessary to guarantee that it is possible to compute an optimal solution to any problem instance (to get the competitive ratio 1). The advice bits are nothing else but "information" about the not yet known part of the input. For sure, the Kolmogorov complexity of the instance is an upper bound on the number $\inf_P(I)$ of advice bits needed to solve the instance optimally in an online manner, however, this is only an upper bound on the information content of this problem instance. Often one does not need the full information about the instance to distinguish it from other ones. Moreover, one does not necessarily need to distinguish each instance from all other ones; it is enough to take the right decision, which can be common for many problem instances. And this is exactly what we are searching for, because this is what we intuitively relate to the desired notion of "information content of a problem".

One can define the information content of an online problem in the common worst-case manner, parameterized by the size of the problem instances. Hence, $\inf_P(n)$ would be the maximum of $\inf_P(I)$ over all instances I of size n. However, to get an exact definition of $\inf_P(n)$, one still has to fix the way in which the advice bits are communicated. The first paper [5] investigating the number of advice bits necessary for solving online problems optimally proposed that the online algorithm may pose a sequence of questions and it always gets the right answer as a non-empty sequence of bits. The sum of the lengths of all answers is the advice complexity. This kind of measurement is rough, because one can code also a lot of information in the lengths of the individual answers and in this way the whole communication can be viewed as a string over three symbols. In fact, the authors of [5] proved that this measurement is rough up to a multiplicative constant factor.

One possible solution to fix the problems of the model introduced in [5] has been proposed in [6]. Here, the algorithm receives an advice of fixed constant length k with each request. The length k is used as a measure of the advice complexity of the problem; for an input consisting of n requests, exactly kn bits of advice are used. This model enforces a smooth flow of the advice information, since the same amount of advice is made available to the algorithm in every request. It is not possible, however, to employ this model to analyze online algorithms that use less than n bits of advice. This is a significant drawback,

because many well-known online problems are very easy to solve optimally with one bit of advice available in every request.

Another approach to define the communication of the advice bits, introduced in [2], is to relax the requirement of a smooth advice information flow and to follow a way analogous to the definition of randomized Turing machines. In the case of randomization, the Turing machine is augmented with a separate tape containing random bits. It is then possible to measure the amount of randomness used as the number of bits accessed on the random tape. Similarly, we can provide an *advice tape* to the online algorithm. The algorithm can access this tape in an arbitrary way and the advice complexity is measured as the total number of advice bits accessed over the whole computation. In this way, it is perfectly possible to analyze problems that use less than 1 advice bit per request on average. The advice complexity can be sublinear as well.

In the sequel, we provide a more formal definition of the information content of an online problem which coincides with the advice complexity as defined in [2]. At first, we define the advice complexity of an online algorithm:

Definition 1. *An* online algorithm $\mathtt{A}$ with advice *computes the output sequence* $\mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where* ϕ *is the content of the advice tape, i. e., an infinite binary sequence, and* $I = (x_1, \ldots, x_n)$ *is an input instance of the considered online problem. Algorithm* $\mathtt{A}$ *is* c-competitive with advice complexity $s(n)$ *if, for every* n *and for each input sequence* I *of length* n*, there exists some* ϕ *such that the competitive ratio of algorithm* $\mathtt{A}$ *on* I *is at most* c*, i. e.,* $\mathrm{cr}_{\mathtt{A}}(I) \leq c$*, and no positions of* ϕ *except the first* $s(n)$ *bits are accessed during the computation of* $\mathtt{A}^{\phi}(I)$*.*

Now we are ready to formally define the information content $\inf_P(n)$ of an online problem P. More precisely, we define upper and lower bounds on the information content $\inf_P(n)$. The definition of the upper bound is straightforward:

Definition 2. *A function* $f(n)$ *is* an upper bound on $\inf_P(n)$ *if and only if there exists a* 1*-competitive online algorithm* $\mathtt{A}$ *with advice complexity* $s(n)$ *such that* $s(n) \leq f(n)$ *for all* n*.*

It is, however, more tricky to find a suitable definition of the lower bound on the information content. As a first attempt, we can make a simple modification to the definition of the upper bound:

Definition 3. *A function* $g(n)$ *is* a lower bound on $\inf_P(n)$ *if and only if, for every* 1*-competitive online algorithm* $\mathtt{A}$ *with advice complexity* $s(n)$*, it holds that* $s(n) \geq g(n)$ *for all* n*.*

Unfortunately, this definition has a serious drawback: It is possible to store a constant, but arbitrarily high, amount of information in the online algorithm itself. In certain cases, such information can be used to solve some input instances very efficiently. In such cases, no nontrivial lower bound on $\inf_P(n)$ can be given in the sense of Definition 3.

As an example of such a situation, consider the following (rather artificial) online problem:

Definition 4 (Guess the Input Length Problem). *An input instance of the* Guess the Input Length *problem (*GIL*) consists of n requests. Each of the first $n-1$ requests is the symbol* 0 *and the last request is the symbol* 1*. The online algorithm must guess the number of requests after the first request is received, i. e., it must output n as an answer to the first request. No answer is required for any other request.*

For any n, we can construct an algorithm that, given advice containing a single bit 0, outputs n. If the first bit of advice is 1, the algorithm reads the complete number from the advice tape and outputs it. In such a way, $\Theta(\log n')$ advice bits are used for all numbers $n' \neq n$, but a single bit of advice is sufficient for an input of length n. Hence, the only feasible lower bound on $\inf_{\mathrm{GIL}}(n)$ in the sense of Definition 3 is $g(n) = 1$.

One possible approach to avoid this problem is to allow the lower bound to be valid only from some minimal n_0 that depends on the online algorithm:

Definition 5. *A function $g(n)$ is* a lower bound on $\inf_P(n)$ *if and only if, for every* 1*-competitive online algorithm* `A` *with advice complexity $s(n)$, there exists some n_0 such that $s(n) \geq g(n)$ holds for all $n \geq n_0$.*

Easily, the Kolmogorov complexity $K(n)$ of the number n is an upper bound on $\inf_{\mathrm{GIL}}(n)$. Intuitively, $K(n)$ should be a lower bound on $\inf_{\mathrm{GIL}}(n)$ as well. Indeed, the online algorithm must output the number n without any a-priori information, the whole information about n must be contained in the used advice. To define the exact value of $K(n)$, we have to fix some "programming language" with respect to which we measure the Kolmogorov complexity. However, once this programming language is fixed, it might still be possible to create a more sophisticated online algorithm for GIL that saves some constant amount of advice for infinitely many inputs. Furthermore, we are not able to provide any bounds on such a constant. Hence, we cannot use the function $K(n)-c$ as a lower bound of $\inf_{\mathrm{GIL}}(n)$, regardless of the choice of the constant c. We may, however, relax Definition 5 in the following way:

Definition 6. *A function $g(n)$ is* a lower bound on $\inf_P(n)$ *if and only if, for every* 1*-competitive online algorithm* `A` *with advice complexity $s(n)$, there exists some constant Δ such that $s(n) + \Delta \geq g(n)$ holds for all n.*

It is not difficult to see that $K(n)$ is a lower bound on $\inf_{\mathrm{GIL}}(n)$ in the sense of Definition 6. A drawback of such a definition is that a lower bound can be in fact larger than an upper bound. The difference between the two, however, can be at most constant, hence it is not an issue if an asymptotic analysis is done.

Alternatively, we could use another way to relax Definition 5:

Definition 7. *A function $g(n)$ is* a lower bound on $\inf_P(n)$ *if and only if, for every* 1*-competitive online algorithm* `A` *with advice complexity $s(n)$, $s(n) \geq g(n)$ holds for infinitely many n.*

Such a definition of a lower bound is very weak – a lower bound can be arbitrarily larger than an upper bound on infinitely many values. Nevertheless, it is not possible that a lower bound is asymptotically larger than an upper bound. On a positive side, Definition 7 allows us to easily claim a lower bound of $\log_2(n)$ on $\inf_{\mathrm{GIL}}(n)$ due to the existence of incompressible strings.

Another approach to avoid the problems of Definition 3 is to actually modify the definition of advice complexity (Definition 1) to require that $s(n)$ bits of advice are sufficient to solve all input instances of length *at most* n. After such modification, we can easily claim $\log_2(n)$ to be a lower bound on $\inf_{\mathrm{GIL}}(n)$ in the sense of Definition 3.

We have presented several possible ways how to define a lower bound on the information content of an online problem. Nevertheless, there is usually little difference between them for the relevant online problems. In the rest of this paper, we stick to the strongest Definition 3, despite to its deficiencies, since all presented results hold with respect to this definition as well. As a shorthand, we write

$$\inf_P(n) \leq f(n)$$

if $f(n)$ is an upper bound on $\inf_P(n)$ and

$$\inf_P(n) \geq g(n)$$

if $g(n)$ is a lower bound on $\inf_P(n)$.

In the sequel, we show that the information content of online problems can be very different even for well-known natural problems. As an example of a problem with a very low information content, consider the SkiRental problem (for a definition, see, e. g., [3]). For any input instance, the optimal solution of the SkiRental problem has very simple structure: Either it consists of buying the skis at the very beginning, or it consists of renting the skis all the time. Hence, a single bit of advice is sufficient (and necessary) to solve the SkiRental problem optimally for input instances of arbitrary lengths. Thus, we have a tight bound on the information content of the SkiRental problem: $\inf_{\textsc{SkiRental}}(n) = 1$.

Next, we show that the well-known Paging problem is an example of an online problem that has linear information content, i. e., any optimal algorithm for this problem requires $\Theta(n)$ advice bits, where n is the number of requests.

Definition 8 (Paging Problem). *The input is a sequence of integers representing requests to logical pages $I = (x_1, \ldots, x_n)$, $x_i > 0$. An online algorithm* A *maintains a buffer (content of the physical memory) $B = \{b_1, \ldots, b_K\}$ of K integers, where K is a fixed constant known to* A*. Before processing the first request, the buffer gets initialized as $B = \{1, \ldots, K\}$. Upon receiving a request x_i, if $x_i \in B$, then $y_i = 0$. If $x_i \notin B$, then a* page fault *occurs, and the algorithm has to find some* victim *b_j, i. e., $B := B \setminus \{b_j\} \cup \{x_i\}$, and $y_i = b_j$. The cost of the solution $\mathcal{A} = \mathtt{A}(I)$ is the number of page faults, i. e., $C(\mathcal{A}) = |\{y_i : y_i > 0\}|$.*

In fact, the paging problem is not a single problem, but rather a collection of problems parameterized by the buffer size K. For simplicity, however, we omit

this parameter from the notation of the problem, and assume that K is a fixed constant known to the online algorithm. It is not difficult to prove a linear upper bound on the information content of PAGING [1, 2]:

Theorem 1.

$$\inf_{\text{PAGING}}(n) \leq n + K$$

Proof. It is sufficient to show that there is an optimal online algorithm (i. e., a 1-competitive algorithm) solving PAGING with advice complexity $n + K$.

Consider an input sequence I 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 A such that, in each step i, the set of OPT's active pages will be in B, and A will maintain with each page an *active* flag identifying this subset. If A gets an input x_i that causes a page fault, some *passive* (i. e., non-active) page is replaced by x_i. Moreover, A reads with each input also one bit from the advice tape telling whether x_i is active for OPT. Since the set of active pages is the same for OPT and A, it is immediate that A generates the same sequence of page faults.

Algorithm A consumes one bit of advice with every request. Moreover, it needs to receive the information about which pages that are stored in the buffer at the beginning of the computation are active. Hence, A reads $n + K$ bits of advice in total. □

Next, we provide a lower bound on $\inf_{\text{PAGING}}(n)$ that shows that the above-described upper bound is tight if the buffer size K is large [1, 2], that is, any optimal algorithm for PAGING must consume almost one bit of advice for every request:

Theorem 2. *There exists a fixed constant C that does not depend on K such that*

$$\inf_{\text{PAGING}}(n) \geq n \left(1 - \frac{\log(K-1) + C}{4(K-1)}\right) - \mathcal{O}(1)$$

holds. The constant of $\mathcal{O}(1)$ depends on K.

Proof. The main idea of the proof is, for any input length n, to construct a set of input instances that are sufficiently different, i. e., that any two instances from this set provably need a different advice if they are to be solved in an optimal way.

We now formally describe how to construct such a set of inputs $\mathcal{I}$. Let $\nu := \lfloor n/(2K-2) \rfloor$ and $Z := \binom{2K-2}{K-1}$. The set $\mathcal{I}$ consists of Z^ν inputs organized in a complete Z-ary tree of height ν, where each edge is labeled with a sequence of $2K-2$ requests. Each leaf of the tree represents one input from $\mathcal{I}$ which is obtained by concatenating the request sequences on all edges of the path from the root to this leaf. Let N_h denote the sequence of $K-1$ requests $(hK + 1, hK + 2, \ldots, hK + K - 1)$. Let us denote the root of the tree to be on level 1, the sons of the root to be on level 2, etc. Each edge e leading from a vertex v of level h is labeled with the sequence N_h followed by the sequence $S(e)$. The

sequences $S(e)$ are defined recursively as follows. For each vertex v, consider a set of K elements $m(v)$; the intuition is that $m(v)$ is the content of the memory of some optimal algorithm processing the given input. Let us define $m(root) = \{1, \ldots, K\}$. Inductively, consider a vertex v of level h with $m(v)$ already defined. The sequences $S(e)$ of the outgoing edges contain $(K-1)$-element subsets of the set $m(v) \cup N_h$ such that they do not contain the element $hK + K - 1$ (the ordering within the sequence is not important; to avoid ambiguity, let us assume the sequences $S(e)$ are increasing). For any edge $e = (v, v')$, let us set $m(v') := S(e) \cup \{hK + K - 1\}$. Since, by using this procedure, we keep the invariant that $m(v) \cap N_h = \emptyset$, there are exactly $Z = \binom{2K-2}{K-1}$ possible subsets, a unique one for every son of v.

In this way, we obtain a set of instances of length $\nu(2K-2)$. Each of these instances can be extended to the length n by repeating the last request $n - \nu(2K-2) = n \bmod (2K-2)$ times.

The intuition behind this construction is as follows. Every vertex of the tree represents some prefix of some input in $\mathcal{I}$. When an optimal algorithm has processed a prefix of an input instance that corresponds to a vertex v, the content of its buffer is $m(v)$. At that moment, a sequence N_h of requests arrives, where h is the level of v. All of the requested pages in N_h are new. Thus, they are not in the buffer and the algorithm must make a page fault for each of them. However, the algorithm can choose which pages in the buffer will be overwritten. In this way, after processing N_h, the algorithm can obtain any content of the buffer that is a subset of $m(v) \cup N_h$ that contains K elements and one of these elements is $hk + K - 1$ (the last request of N_h).

After processing N_h, the set of control requests $S(e)$ follows. If the algorithm made a correct choice while processing N_h, all elements of $S(e)$ are in the buffer after processing N_h, hence no page faults are generated while processing $S(e)$. On the other hand, if a wrong choice was made, at least one page fault is necessary. Easily, once a wrong choice is made, the algorithm cannot be optimal.

Assume that two inputs from $\mathcal{I}$ are given the same advice. Let v be the vertex representing the longest common prefix of these inputs, and h be the level of v. The online algorithm makes the same choice while processing N_h. Therefore, this choice is wrong in at least one of the inputs, and the online algorithm fails to achieve optimality.

Hence, the advice complexity of $\mathtt{A}$ has to be at least $\log(|\mathcal{I}|) = \log(Z^\nu) = \nu \log(Z)$ bits:

$$\begin{aligned} s(n) \geq \nu \log(Z) &= \left\lfloor \frac{n}{2K-2} \right\rfloor \log \binom{2K-2}{K-1} \\ &\geq \left(\frac{n}{2K-2} - 1 \right) \log \binom{2K-2}{K-1} = \frac{n}{2K-2} \log \binom{2K-2}{K-1} - \mathcal{O}(1). \end{aligned} \tag{1}$$

Using the Stirling formula, we have

$$\binom{2t}{t} = \frac{(2t)!}{(t!)^2} \geq \frac{\sqrt{2\pi 2t} \left(\frac{2t}{e}\right)^{2t}}{\left(\sqrt{2\pi t} \left(\frac{t}{e}\right)^t \left(1 + \mathcal{O}\left(\frac{1}{t}\right)\right)\right)^2} \geq 2^{2t} \cdot \frac{D}{\sqrt{t}} \tag{2}$$

for some constant D not depending on K. Substituting $t := K - 1$ into (2) and (1), we complete the proof:

$$\begin{aligned} s(n) &\geq n \left(\frac{1}{2K-2} \log \left(2^{2(K-1)} \frac{D}{\sqrt{K-1}} \right) \right) - \mathcal{O}(1) \\ &= n \cdot \frac{2(K-1) + \log(D) - \frac{1}{2}\log(K-1)}{2K-2} - \mathcal{O}(1) \\ &= n \left(1 - \frac{-2\log(D) + \log(K-1)}{4K-4} \right) - \mathcal{O}(1). \end{aligned}$$

□

The information content of some other online problems has been analyzed in [1,2]. In particular, the problem of *Disjoint Path Allocation* (DisPathAlloc for short) and the problem of *Job Shop Scheduling for Two Jobs* (JSSchedule for short) have been considered. For definitions and more information about these problems, see, e. g., [3] and [7,8], respectively.

The DisPathAlloc problem is an another example of an online problem with high information content: Any optimal algorithm for DisPathAlloc needs at least $n/2 - 1$ bits of advice to solve input instances with n requests. On the other hand, there is a very simple optimal algorithm that consumes one bit of advice for every request. Hence, we can write $n \geq \inf_{\text{DisPathAlloc}}(n) \geq n/2 - 1$.

The JSSchedule problem, on the other hand, has a lower information content. If there are n jobs to be scheduled, $2\lceil\sqrt{n}\rceil$ advice bits are sufficient to find the optimal solution. This bound is asymptotically tight, i. e., $\Omega(\sqrt{n})$ of advice bits are necessary.

The following table summarizes the known results about the information content of the discussed online problems:

P	Upper bound on $\inf_P(n)$	Lower bound on $\inf_P(n)$
Paging	$n + K$	$n\left(1 - \frac{\log(K-1)+C}{4(K-1)}\right) - \mathcal{O}(1)$
DisPathAlloc	n	$\frac{n}{2} - 1$
JSSchedule	$2\lceil\sqrt{n}\rceil$	$\Omega(\sqrt{n})$

Table 1: Upper and lower bounds on the information content of online problems.

3 Relative Information Content of Online Problems

So far, we have focused on the full information content of online problems, defined as the size of advice about future requests that is needed to solve the problem

optimally. The requirement of an optimal solution is, however, very strict. In the context of online problems, suboptimal solutions are often analyzed and their quality is measured by their competitive ratio. Hence, it makes sense to consider a trade-off between the competitive ratio of an online algorithm and the size of the advice needed to achieve this ratio. This leads us to the definition of *information content of an online problem P relative to the competitive ratio r*, denoted as $\inf_P^{(r)}(n)$. Again, we define upper and lower bounds on $\inf_P^{(r)}(n)$:

Definition 9. *A function $f(n)$ is* an upper bound on $\inf_P^{(r)}(n)$ *if and only if there exists an r-competitive online algorithm* A *with advice complexity $s(n)$ such that $s(n) \leq f(n)$ for all n.*

Definition 10. *A function $g(n)$ is* a lower bound on $\inf_P^{(r)}(n)$ *if and only if, for every r-competitive online algorithm* A *with advice complexity $s(n)$, it holds that $s(n) \geq g(n)$ for all n.*

The discussion about alternative definitions of the information content is valid for the definition of relative information content as well. Thus, it is possible to create variants of Definition 10 in an analogous way to Definitions 5, 6, and 7.

Several results about the relative information content of online problems are known as well [1, 2]. For the paging problem, it is known that linear advice is necessary to achieve competitive ratios close to 1. More precisely, for any $1 \leq r \leq 1.25$ and any fixed K, $\inf_{\text{PAGING}}^{(r)} \geq \Omega(n)$. The constant hidden in the asymptotic notation, however, depends on r and K. The precise bound can be formulated as follows [1, Theorem 2]:

Theorem 3. *Let r be any constant such that $1 \leq r \leq 1.25$. It holds that*

$$\inf_{\text{PAGING}}^{(r)} \geq \frac{n}{2K-2}\left[1+\log(3-2r)-(2r-2)\log\left(\frac{1}{2r-2}-1\right)\right]-\mathcal{O}(1).$$

The constant of $\mathcal{O}(1)$ depends on K and the parameter r.

The idea behind the proof of Theorem 3 is to consider the set of instances defined in the proof of Theorem 2 and adapt the analysis for the case of algorithms with a fixed competitive ratio. The lower bound of Theorem 3 can be complemented by the following upper bound [1, Theorem 1] which states that, for any r, it is possible to achieve an r-competitive algorithm for PAGING with linear advice. Moreover, the linearity constant tends to 0 with growing r:

Theorem 4. *For each constant $r \geq 1$, it holds that*

$$\inf_{\text{PAGING}}^{(r)} \leq n\log\left(\frac{r+1}{r^{\frac{r}{r+1}}}\right)+3\log n+\mathcal{O}(1).$$

Theorems 3 and 4 suggest that large advice is necessary for PAGING if a constant competitive ratio is required. Now we focus on the opposite end of the competitive ratio spectrum, where the competitive ratio is not a fixed constant.

It is a well-known fact that any deterministic algorithm for paging cannot be better than K-competitive [3]. It is, however, an interesting fact that even extremely small advice can significantly improve this ratio. For example, just two bits of advice for the whole input instance are sufficient to obtain an algorithm with competitive ratio $K/2 + 7.5$, regardless of the number of input requests. In general, b bits of advice are sufficient to obtain a competitive ratio of $\frac{2(K+1)}{2^b} + 3b + 1$ [1, Theorem 5]:

Theorem 5. *Let b be any fixed integer and let*

$$r := \frac{2(K+1)}{2^b} + 3b + 1.$$

Then

$$\inf^{(r)}_{\text{PAGING}} \leq b.$$

The main idea behind this result is to construct 2^b deterministic paging algorithms that are sufficiently different and that the total number of page faults they make together has a reasonable upper bound. It turns out that the idea of marking algorithms (as described in [3]) can be used to achieve this goal. Afterwards, we can argue that, for any input instance, at least one of these algorithms works well. The advice provided is then just an identification of the best possible algorithm from the constructed set.

On the other hand, it is possible to prove a lower bound corresponding to Theorem 5 [1, Theorem 6]. In fact, this lower bound shows that the upper bound of Theorem 5 is almost tight for small b:

Theorem 6. *Let b be any fixed integer and let $r := K/2^b$. Then*

$$\inf^{(r)}_{\text{PAGING}} \geq b.$$

The idea behind this lower bound is not very involved. It is sufficient to consider a set of all possible input instances of length n that use only numbers 1 to $K+1$ in their requests. It is not difficult to see that the optimal algorithm makes at most n/K page faults for any such instance. On the other hand, if the advice is short, there are many different inputs with the same advice, what can be used to infer that the algorithm makes sufficiently many page faults on at least one of them.

For the DISPATHALLOC problem, we have the following lower bound on the relative information content [1, Theorem 8]:

Theorem 7.

$$\inf^{(r)}_{\text{DISPATHALLOC}} \geq \frac{n+2}{2r} - 2$$

Furthermore, it is not difficult to provide an upper bound on $\inf^{(r)}_{\text{DisPathAlloc}}$ that is only a factor of $\log n$ away from the lower bound of Theorem 7 for large r, and that is asymptotically tight for constant r [1, Theorem 9]:

Theorem 8.

$$\inf^{(r)}_{\text{DisPathAlloc}} \leq \min\left\{\left(\frac{n}{r}+3\right)\log n + \mathcal{O}(1), n\log\frac{r}{(r-1)^{\frac{r-1}{r}}} + 3\log n + \mathcal{O}(1)\right\}$$

For the JSSchedule problem, it is straightforward to adapt the randomized algorithm of [7] to achieve an algorithm with small (logarithmic) advice and a competitive ratio close to 1:

Theorem 9. *There is an online algorithm for* JSSchedule *with advice complexity $s(n) \leq 1 + \log(n)$ that achieves competitive ratio $\mathcal{O}(1 + 1/\sqrt{n})$. Hence,*

$$\inf^{(r)}_{\text{JSSchedule}} \leq 1 + \log(n),$$

for any $r > 1$.

4 Relating Information Content to Computational Complexity

As already mentioned, the first fundamental contribution of informatics to science was the development of the formal concept of an algorithm. The second most important concept discussed in informatics is the concept of computational complexity. The above proposed definitions of the information content of online problems are not related to this concept. This is the same case as for the Kolmogorov complexity that is also not related to the complexity of generating strings from their representations. But for relating the Kolmogorov complexity to the computational complexity of string generation, the concept of resource-bounded Kolmogorov complexity has been developed. It is possible to apply a similar idea in order to combine the information content of online problems with the efficiency of online algorithms.

For any function $f\colon \mathbb{N} \to \mathbb{N}$, we define the *$f$-resource-bounded information content of an online problem P relative to the competitive ratio r*, denoted as $\inf^{(f,r)}_{P}(n)$:

Definition 11. *Consider the function $f\colon \mathbb{N} \to \mathbb{N}$. The function $h(n)$ is* an upper bound on $\inf^{(f,r)}_{P}(n)$ *if and only if there exists an r-competitive online algorithm* `A` *with advice complexity $s(n)$ such that $s(n) \leq h(n)$ for all n, and the running time of* `A` *is at most $f(n)$ for any input of length n.*

Definition 12. *Consider the function $f\colon \mathbb{N} \to \mathbb{N}$. The function $g(n)$ is* a lower bound on $\inf^{(f,r)}_{P}(n)$ *if and only if, for every r-competitive online algorithm* `A` *with advice complexity $s(n)$ such that the running time of* `A` *is at most $f(n)$ for any input of length n, it holds that $s(n) \geq g(n)$ for all n.*

In our examples of online problems such as PAGING, DISPATHALLOC, or JSSCHEDULE, the computational complexity does not matter, because all upper bounds on the number of advice bits can be achieved by efficient online algorithms with advice. But this is not necessarily true for any online problem. If one considers the special online problem GIL (estimating the length of the input), the Kolmogorov complexity of the lengths does not need to be an upper bound on the number of advice bits. An interesting research idea would be to study this phenomenon for "natural" optimization problems. If an NP-hard optimization problem has to be solved in polynomial time in an online manner, then it is not necessarily sufficient to get a compressed version of the whole input.

References

1. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. Online algorithms with advice. Technical Report 614, ETH Zürich, 2009.
2. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2009.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
4. G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13(4):547–569, 1966.
5. S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? In V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, and M. Bieliková, editors, *Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910 of *Lecture Notes in Computer Science*, pages 247–258, Berlin, 2008. Springer-Verlag.
6. Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes in Computer Science*, pages 427–438. Springer-Verlag, 2009.
7. J. Hromkovič. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag, New York, 2005.
8. J. Hromkovič, T. Mömke, K. Steinhöfel, and P. Widmayer. Job shop scheduling with unit length tasks: bounds and algorithms. *Algorithmic Operations Research*, 2(1):1–14, 2007.
9. A. N. Kolmogorov. Three approaches to the definition of the concept "quantity of information". *Problemy Peredachi Informatsii*, 1:3–11, 1965.
10. C. E. Shannon. A mathematical theory of communication. *Mobile Computing and Communications Review*, 5(1):3–55, 2001.
11. A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.