Advice Complexity for a Class of Online Problems*

Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen

University of Southern Denmark
{joan,lenem,kudahl,jesperwm}@imada.sdu.dk

Abstract

The advice complexity of an online problem is a measure of how much knowledge of the future an online algorithm needs in order to achieve a certain competitive ratio. We determine the advice complexity of a number of hard online problems including independent set, vertex cover, dominating set and several others. These problems are hard, since a single wrong answer by the online algorithm can have devastating consequences. For each of these problems, we show that $\log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n=\Theta(n/c)$ bits of advice are necessary and sufficient (up to an additive term of $O(\log n)$) to achieve a competitive ratio of c. This is done by introducing a new string guessing problem related to those of Emek et al. (TCS 2011) and Böckenhauer et al. (TCS 2014). It turns out that this gives a powerful but easy-to-use method for providing both upper and lower bounds on the advice complexity of an entire class of online problems.

Previous results of Halldórsson et al. (TCS 2002) on online independent set, in a related model, imply that the advice complexity of the problem is $\Theta(n/c)$. Our results improve on this by providing an exact formula for the higher-order term. Böckenhauer et al. (ISAAC 2009) gave a lower bound of $\Omega(n/c)$ and an upper bound of $O((n\log c)/c)$ on the advice complexity of online disjoint path allocation. We improve on the upper bound by a factor of $\log c$. For the remaining problems, no bounds on their advice complexity were previously known.

1998 ACM Subject Classification F.1.2 Models of Computation (online computation)

Keywords and phrases online algorithms, advice complexity, asymmetric string guessing, advice complexity class AOC, covering designs

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.116

1 Introduction

An online problem is an optimization problem in which the input is divided into small pieces, usually called requests, arriving sequentially. Throughout the paper, we let n denote the number of requests. An online algorithm must serve each request without any knowledge of future requests, and the decisions made by the online algorithm are irrevocable. The goal is to minimize or maximize some objective function. Traditionally, competitive analysis is used to measure the quality of an online algorithm: The solution produced by the algorithm is compared to the solution produced by an optimal offline algorithm, `OPT`, which knows the entire request sequence in advance. While competitive analysis has been very successful and led to the design of many interesting online algorithms, it sometimes gives overly pessimistic results. Comparing an online algorithm, which knows nothing about the future, to an optimal offline algorithm, which knows the entire input, can be rather crude.

* This work was partially supported by the Villum Foundation and the Danish Council for Independent Research, Natural Sciences. Most proofs have been omitted due to space restrictions. These can be found in the full version of the paper [9].

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 116–129

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As an example, consider the classical problem of finding a maximum independent set in a graph. Suppose that, at some point, an online algorithm decides to include a vertex v in its solution. It then turns out that all forthcoming vertices in the graph are connected to v and no other vertices. Thus, the online algorithm cannot include any of these vertices. On the other hand, OPT knows the entire graph, and so it rejects v and instead takes all forthcoming vertices. In fact, one can easily show that no online algorithm, even if we allow randomization, can obtain a competitive ratio better than $\Omega(n)$ for this problem.

This paper studies the maximum independent set problem and other similarly hard online problems. Many papers have studied special cases or relaxed versions of such problems, see e.g., [12, 20–22, 25, 33]. Often it is the online constraint that is relaxed, resulting in various semi-online models. The notion of advice complexity offers a quantitative and standardized, i.e., problem independent, way of relaxing the online constraint.

Advice complexity. The main idea of advice complexity is to provide an online algorithm, ALG, with some advice bits. These bits are provided by a trusted oracle, O, which has unlimited computational power and knows the entire request sequence. In the first model proposed [15], the advice bits were given as answers (of varying lengths) to questions posed by ALG. One difficulty with this model is that using at most 1 bit, three different options can be encoded (giving no bits, a 0, or a 1). This problem was addressed by the model proposed in [16], where the oracle is required to send a fixed number of advice bits per request. However, for the problems we consider, one bit per request is enough to guarantee an optimal solution, and so this model is not applicable. Instead, we will use the "advice-on-tape" model [7], which allows for a sublinear number of advice bits while avoiding the problem of encoding information in the length of each answer. Before the first request arrives, the oracle prepares an *advice tape*, an infinite binary string. The algorithm ALG may, at any point, read some bits from the advice tape. The advice complexity of ALG is the maximum number of bits read by ALG for any input sequence of at most a given length. When advice complexity is combined with competitive analysis, the central question is: How many bits of advice are necessary and sufficient to achieve a given competitive ratio c?

▶ **Definition 1** (Advice complexity [7, 24] and competitive ratio [26, 32])**.** The input to an online problem, P, is a request sequence $\sigma = (r_1, \ldots, r_n)$. An *online algorithm with advice*, ALG, computes the output $y = (y_1, \ldots, y_n)$, under the constraint that y_i is computed from $\varphi, r_1, \ldots, r_i$, where φ is the content of the advice tape. The *advice complexity*, $b(n)$, of ALG is the largest number of bits of φ read by ALG over all possible inputs of length at most n.

Each possible output for P is associated with a *score*. For a request sequence σ, $\texttt{ALG}(\sigma)$ ($\texttt{OPT}(\sigma)$) denotes the score of the output computed by ALG (OPT) when serving σ. If P is a maximization problem, then ALG is $c(n)$-*competitive* if there exists a constant α such that $\texttt{OPT}(\sigma) \leq c(n) \cdot \texttt{ALG}(\sigma) + \alpha$ for all request sequences σ of length at most n. If P is a minimization problem, then ALG is $c(n)$-*competitive* if there exists a constant α such that $\texttt{ALG}(\sigma) \leq c(n) \cdot \texttt{OPT}(\sigma) + \alpha$ for all request sequences σ of length at most n. In both cases, if the inequality holds with $\alpha = 0$, we say that ALG is *strictly* $c(n)$-*competitive*. For $c \geq 1$, the *advice complexity*, $f(n, c)$, of a problem P is the smallest possible advice complexity of a c-competitive online algorithm for P.

We only consider deterministic online algorithms (with advice). Note that both the advice read and the competitive ratio may depend on n, but, for ease of notation, we often write b and c instead of $b(n)$ and $c(n)$. Also, by this definition, $c \geq 1$, for both minimization and maximization problems. Lower and upper bounds on the advice complexity have been obtained for many problems, see e.g., [2, 4–8, 10, 11, 14–16, 18, 19, 24, 27, 28, 30, 31].

Online string guessing. In [5, 16], the advice complexity of the following string guessing problem, SG, is studied: For each request, which is simply empty and contains no information, the algorithm tries to guess a single bit (or more generally, a character from some finite alphabet). The correct answer is either revealed as soon as the algorithm has made its guess (known history), or all of the correct answers are revealed together at the very end of the request sequence (unknown history). The goal is to guess correctly as many bits as possible. This problem was first introduced (under the name generalized matching pennies) in [16], where a lower bound for randomized algorithms with advice was given. In [5], the lower bound was improved for the case of deterministic algorithms. In fact, the lower bound given in [5] is tight up to lower-order terms. While SG is rather uninteresting in the view of traditional competitive analysis, it is very useful in an advice complexity setting. Indeed, it has been shown that the string guessing problem can be reduced to many classical online problems, thereby giving lower bounds on the advice complexity for these problems. This includes bin packing [11], the k-server problem [19], list update [10], metrical task system [16], set cover [5] and a certain version of maximum clique [5].

Our contribution. In this paper, we introduce a new *asymmetric string guessing* problem, ASG, formally defined in Section 2. The rules are similar to those of the original string guessing problem with an alphabet of size two, but the score function is asymmetric: If the algorithm answers 1 and the correct answer is 0, then this counts as a single wrong answer (as in the original problem). On the other hand, if the algorithm answers 0 and the correct answer is 1, the solution is infeasible and has an infinite penalty. This asymmetry in the score function forces the algorithm to be very cautious when making its guesses. It turns out that ASG captures, in a very precise way, the hardness of problems such as online independent set and online vertex cover.

We give lower and upper bounds on the advice complexity of the new asymmetric string guessing problem, ASG. The bounds are tight up to an additive term of $O(\log n)$. More precisely, if b is the number of advice bits necessary and sufficient to achieve a (strict)[1] competitive ratio of $c > 1$, then we show that[2]

$$\frac{1}{e \ln 2}\frac{n}{c} - \Theta(\log n) \leq b = \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n \pm \Theta(\log n) \leq \frac{n}{c} + \Theta\left(\log n\right).$$

This holds for all variants of the asymmetric string guessing problem (minimization/-maximization and known/unknown history). See Figure 1 on page 122 for a graphical plot.

We introduce a class, AOC, of online problems. The class AOC essentially consists of those problems which can be reduced to ASG. In particular, for any problem in AOC, our upper bound on the advice complexity for ASG applies. This is one of the few known examples of a general technique for constructing online algorithms with advice, which works for an entire class of problems.

On the hardness side, we show that several online problems, including vertex cover, cycle finding, dominating set, independent set, set cover and disjoint path allocation (described in Section 5) are AOC-complete, that is, they have the same advice complexity as ASG. We

[1] The upper bound holds for being strictly c-competitive, while the lower bound also holds for being c-competitive. For the lower bound, the constant hidden in $\Theta(\log n)$ depends on the additive constant α of the c-competitive algorithm.

[2] We only consider $c > 1$ since in order to be strictly 1-competitive, an algorithm needs to correctly guess every single bit. It is easy to show that this requires n bits of advice (see e.g. [5]). By Remark 9, this also gives a lower bound for being 1-competitive.

prove this by providing reductions from ASG to each of these problems. The reductions preserve the competitive ratio and only increase the advice read by an additive term of $O(\log n)$. Thus, we obtain bounds on the advice complexity of each of these problems which are essentially tight.

As a key step in obtaining our results, we establish a connection between the advice complexity of ASG and the size of covering designs (a well-studied object from the field of combinatorial designs).

Comparison with previous results. The original string guessing problem, SG, can be viewed as a maximization problem, the goal being to correctly guess as many of the n bits as possible. Clearly, OPT always obtains a profit of n. With a single bit of advice, an algorithm can achieve a strict competitive ratio of 2: The advice bit simply indicates whether the algorithm should always guess 0 or always guess 1. This is in stark contrast to ASG, where linear advice is needed to achieve any constant competitive ratio. On the other hand, for both SG and ASG, achieving a constant competitive ratio $c < 2$ requires linear advice. However, the exact amount of advice required to achieve a particular competitive ratio $c < 2$ is larger for ASG than for SG. See Figure 1 for a graphical comparison.

The problems online independent set and online disjoint path allocation, which we show to be AOC-complete, have previously been studied in the context of advice complexity or similar models. In [7], the advice complexity of online disjoint path allocation is considered. It is shown that a strictly c-competitive algorithm must read at least $\frac{n+2}{2c} - 2$ bits of advice. On the other hand, the authors show that for any $c \geq 2$, there exists a strictly c-competitive online algorithm reading at most $\min\left\{n \log\left(\frac{c}{(c-1)^{(c-1)/c}}\right), \frac{n \log n}{c}\right\} + 3 \log n + O(1)$ bits of advice. We remark that $(n \log c)/c < n \log\left(c/(c-1)^{(c-1)/c}\right) < 2(n \log c)/c$, for $c \geq 2$. Thus, this older upper bound is $\Theta((n \log c)/c)$ and at least a factor of $2 \log c$ away from the lower bound. We improve on these bounds, removing the gap between the upper and lower bound.

In [21], the online independent set problem is considered in a multi-solution model. In this model, an online algorithm is allowed to maintain multiple solutions. The algorithm knows (a priori) the number n of vertices in the input graph, and the model is parametrized by a function $r(n)$. Whenever a vertex v is revealed, the algorithm can include v in at most $r(n)$ different solutions (some of which might be new solutions with v as the first vertex). At the end, the algorithm outputs the solution containing the most vertices. The multi-solution model is closely related to the advice complexity model. Simple conversions allow one to translate both upper and lower bounds between the two models almost exactly, up to an additive term of $O(\log n)$. Doing so, the results of [21] can be summarized as follows: For any $c \geq 1$, there is a strictly c-competitive independent set algorithm reading at most $\lceil n/c \rceil + O(\log n)$ bits of advice. On the other hand, any strictly c-competitive algorithm for independent set must read at least $\frac{n}{2c} - \log n$ bits of advice. Our improvement for this problem consists of determining the exact coefficient of the higher-order term.

One important feature of the framework that we introduce in this paper is that obtaining tight bounds on the advice complexity for problems like online independent set and online disjoint path allocation becomes very easy. We remark that the reductions we use to show the hardness of these two problems reduce instances of ASG to instances of online independent set (resp. disjoint path allocation) that are identical to the hard instances used in [21] (resp. [7]). What enables us to improve the previous bounds, even though we use the same hard instances, is that we have a detailed analysis of the advice complexity of ASG at our disposal.

Related work. The advice complexity of online disjoint path allocation has also been studied as a function of the length of the path (as opposed to the number of requests), see [3,7]. The advice complexity of online independent set on bipartite graphs and on sparse graphs has been determined in [14]. The advice complexity of an online set cover problem [1] has been studied in [27]. However, the version of online set cover that we consider is different and so our results and those of [27] are incomparable.

Preliminaries. Let log denote the binary logarithm $\log_2$ and ln the natural logarithm $\log_e$. By a *string* we always mean a bit string. For a string $x \in \{0,1\}^n$, we denote by $|x|_1$ the Hamming weight of x (that is, the number of 1s in x) and we define $|x|_0 = n - |x|_1$. Also, we denote the i'th bit of x by x_i, so that $x = x_1 x_2 \dots x_n$. For $n \in \mathbb{N}$, define $[n] = \{1, 2, \dots, n\}$. For a subset $Y \subseteq [n]$, the *characteristic vector* of Y is the string $y = y_1, \dots, y_n \in \{0,1\}^n$ such that, for all $i \in [n]$, $y_i = 1$ if and only if $i \in Y$. For $x, y \in \{0,1\}^n$, we write $x \sqsubseteq y$ if $x_i = 1 \Rightarrow y_i = 1$ for all $1 \leq i \leq n$.

If the oracle needs to communicate some integer m to the algorithm, and if the algorithm does not know of any upper bound on m, the oracle needs to use a self-delimiting encoding. For instance, the oracle can write $\lceil \log m \rceil$ in unary (a string of 1's followed by a 0) before writing m itself in binary. In total, this encoding uses $O(\log m)$ bits. Slightly more efficient encodings exist, see e.g. [6].

2 Asymmetric String Guessing

In this section, we formally define the asymmetric string guessing problem. There are four variants of the problem, one for each combination of minimization/maximization and known/unknown history. Collectively, these four problems will be referred to as ASG. We have deliberately tried to mimic the definition of the string guessing problem SG from [5].

▶ **Definition 2.** The *minimum asymmetric string guessing problem with unknown history*, minASGu, has input $(?_1, \dots, ?_n, x)$, where $x \in \{0,1\}^n$, for some $n \in \mathbb{N}$. For $1 \leq i \leq n$, round i proceeds as follows:

1. The algorithm receives request $?_i$ which contains no information.
2. The algorithm answers y_i, where $y_i \in \{0,1\}$.

The *output* $y = y_1 \dots y_n$ computed by the algorithm is *feasible*, if $x \sqsubseteq y$. Otherwise, y is *infeasible*. The *cost* of a feasible output is $|y|_1$, and the cost of an infeasible output is ∞.

▶ **Definition 3.** The *minimum asymmetric string guessing problem with known history*, minASGk, has input $x = (?, x_1, \dots, x_n)$, where $x \in \{0,1\}^n$, for some $n \in \mathbb{N}$. For $1 \leq i \leq n$, round i proceeds as follows:

1. If $i > 1$, the algorithm learns the correct answer, x_{i-1}, to the request in the previous round.
2. The algorithm answers $y_i = f(x_1, \dots, x_{i-1}) \in \{0,1\}$, where f is a function defined by the algorithm.

The *output* $y = y_1 \dots y_n$ computed by the algorithm is *feasible*, if $x \sqsubseteq y$. Otherwise, y is *infeasible*. The *cost* of a feasible output is $|y|_1$, and the cost of an infeasible output is ∞.

We collectively refer to minASGk and minASGu as minASG. The string x in either version of minASG will be referred to as the *input string* or the *correct string*. Note that the number of requests in both versions of minASG is $n + 1$, since there is a final request that does not require any response from the algorithm. This final request ensures that the

entire string x is eventually known. For simplicity, we will measure the advice complexity of minASG as a function of n (this choice is not important as it changes the complexity by at most one bit).

Without advice the situation is the following. For any deterministic minASG algorithm which sometimes answers 0, there exists an input string on which the algorithm gets a cost of ∞. However, if an algorithm always answers 1, the input string could consist solely of 0s. Thus, no deterministic algorithm can achieve a finite competitive ratio. One can easily show that the same holds for any randomized algorithm.

We now give a simple algorithm for minASG which reads $O(n/c)$ bits of advice and achieves a strict competitive ratio of $\lceil c \rceil$.

▶ **Theorem 4.** *For any $c \geq 1$, there is a strictly $\lceil c \rceil$-competitive algorithm for* minASGu *which reads $\lceil \frac{n}{c} \rceil + O(\log(n/c))$ bits of advice.*

Proof. Let $x = x_1 \dots x_n$ be the input string. The oracle encodes $p = \lceil n/c \rceil$ in a self-delimiting way, which requires $O(\log(n/c))$ bits of advice. For $0 \leq j < p$, define $C_j = \{x_i : i \equiv j \pmod p\}$. These p sets partition the input string, and the size of each C_j is at most $\lceil n/p \rceil \leq \lceil c \rceil$. The oracle writes one bit, b_j, for each set C_j. If C_j contains only 0s, b_j is set to 0. Otherwise, b_j is set to 1.

The algorithm learns p and the bits $b_0, \dots, b_{p-1}$ from the advice tape. In round i, the algorithm answers with the bit $b_{i \bmod p}$. Clearly, this algorithm is strictly $\lceil c \rceil$-competitive. ◀

The definition of the maximization version of ASG is similar to the definition of minASG:

▶ **Definition 5.** The *maximum asymmetric string guessing problem with unknown history*, maxASGu, is identical to the minASGu problem with unknown history, except that the score function is different. In the maxASGu problem, the score of a feasible output y is $|y|_0$. The score of an infeasible output is $-\infty$. The goal is to maximize the score. The *maximum asymmetric string guessing problem with known history*, maxASGk, is defined similarly.

We collectively refer to the two maximization problems as maxASG.

An algorithm for maxASG without advice cannot attain a finite competitive ratio. If such an algorithm would ever answer 0 in some round, an adversary would let the correct answer be 1 and the algorithm's output would be infeasible. On the other hand, answering 1 in every round gives an output with a profit of zero.

▶ **Theorem 6.** *For any $c \geq 1$, there is a strictly $\lceil c \rceil$-competitive algorithm for* maxASGu *which reads $\lceil n/c \rceil + O(\log n)$ bits of advice.*

Proof. The oracle partitions the input string $x = x_1 \dots x_n$ into $\lceil c \rceil$ disjoint blocks, each containing (at most) $\lceil \frac{n}{c} \rceil$ consecutive bits. Note that there must exist a block where the number of 0s is at least $|x|_0 / \lceil c \rceil$. The oracle uses $O(\log n)$ bits to encode the index i at which this block starts and the index i' at which it ends. Furthermore, the oracle writes the string $x_i \dots x_{i'}$ onto the advice tape, which requires at most $\lceil \frac{n}{c} \rceil$ bits, since this is the largest possible size of a block. The algorithm learns the string $x_i \dots x_{i'}$ and answers accordingly in rounds i to i'. In all other rounds, the algorithm answers 1. ◀

In the following sections, we determine the amount of advice necessary and sufficient to achieve some (strict) competitive ratio $c > 1$. It turns out that the algorithms from Theorems 4 and 6 use the asymptotically smallest possible number of advice bits, but the coefficient in front of the term n/c can be improved.

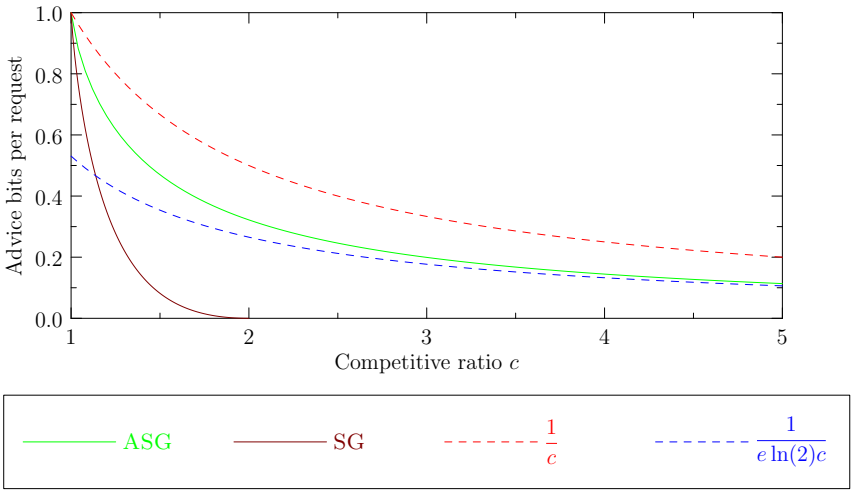

Figure 1 The upper solid line (green) shows the number of advice bits per request which are necessary and sufficient for obtaining a (strict) competitive ratio of c for ASG (ignoring lower-order terms). The lower solid line (brown) shows the same number for the original string guessing problem SG [5]. The dashed lines are the functions $1/c$ and $1/(e\ln(2)c)$.

3 Advice Complexity of ASG

In order to determine the advice complexity of ASG, we will use some basic results from the theory of combinatorial designs. Let $v \geq k \geq t$ be integers. A *(v,k,t)-covering design* is a family of k-subsets (called *blocks*) of a v-set, X, such that any t-subset of X is contained in at least one block. The *size* of a covering design, D, is the number of blocks in D. The *covering number*, $C(v,k,t)$, is the smallest possible size of a (v,k,t)-covering design. The connection to ASG is that for inputs to MINASG where the number of 1s is t, an (n,ct,t)-covering design can be used to obtain a strictly c-competitive algorithm. Many papers have been devoted to the study of covering numbers. See [13] for a survey. We make use of the following bounds on the size of a covering design:

▶ **Lemma 7** (Erdős, Spencer [17]). *For all natural numbers $v \geq k \geq t$,*

$$\frac{\binom{v}{t}}{\binom{k}{t}} \leq C(v,k,t) \leq \frac{\binom{v}{t}}{\binom{k}{t}}\left(1+\ln\binom{k}{t}\right) \tag{1}$$

We will state the obtained advice complexity bounds in terms of the following function:

$$B(n,c) = \log\left(1+\frac{(c-1)^{c-1}}{c^c}\right)n \tag{2}$$

For $c > 1$, we show that $B(n,c) \pm O(\log n)$ bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c, for any version of ASG. See Figure 1 for a graphical view. It can be shown that $n/(a \cdot c) \leq B(n,c) \leq n/c$, where $a = e\ln(2)$. In particular, if $c = o(n/\log n)$, we see that $O(\log n)$ becomes a lower-order term. Thus, for this range of c, we determine exactly the higher-order term in the advice complexity of ASG. Since this is the main focus of our paper, we will consider $O(\log n)$ a lower-order term. The case where $c = \Omega(n/\log n)$ is treated in the full version of the paper [9].

3.1 Advice Complexity of minASG

The following theorems show that covering numbers are closely related to the amount of advice needed for an algorithm, ALG, to achieve a strict competitive ratio of c for minASG.

▶ **Theorem 8.** *For any $c > 1$, there exists a strictly c-competitive algorithm for* minASGu *and* minASGk *reading b bits of advice, where*

$$b = \log\left(\max_{t:\lfloor ct\rfloor<n} C(n,\lfloor ct\rfloor,t)\right) + O(\log n) = B(n,c) + O(\log n).$$

Proof (sketch). Let $x = x_1 \dots x_n$ be an input string to minASG and set $t = |x|_1$. The oracle encodes n and t in a self-delimiting way. If $t = 0$ or $ct \geq n$, it is trivial to output a solution of the desired quality. In all other cases, ALG computes an optimal $(n, \lfloor ct\rfloor, t)$-covering design by making a systematic enumeration of all possible solutions (using lexicographic order, say). Note there must exist a $\lfloor ct\rfloor$-block Y from this covering design such that the characteristic vector y of Y satisfies $x \sqsubseteq y$. Using $\lceil \log C(n,\lfloor ct\rfloor,t)\rceil$ bits of advice, the oracle can encode the index of Y. A lengthy calculation using (1) allows us to express this upper bound in terms of the function $B(n,c)$ defined in (2). ◀

Note that an algorithm for minASGu reading $b(n)$ bits of advice can only produce $2^{b(n)}$ different outputs, one for each possible advice string. Consider the set of input strings $I_{n,t} = \{x \in \{0,1\}^n \colon |x|_1 = t\}$, and let $Y_{n,t}$ be the corresponding set of output strings produced by the algorithm. If the algorithm is strictly c-competitive, then $Y_{n,t}$ can be converted into an $(n,\lfloor ct\rfloor,t)$-covering design. This gives a lower bound of $\log\left(\max_{t:\lfloor ct\rfloor<n} C(n,\lfloor ct\rfloor,t)\right)$ on the advice needed to achieve a strict competitive ratio of c for minASGu.

Recall that for minASGk, the output produced by an algorithm can depend on both the advice read and the correct answer to requests in previous rounds. In particular, the proof of the lower bound for minASGu sketched above breaks down for minASGk. However, by using a more complicated argument, Theorem 10 gives a lower bound of $B(n,c) - O(\log n)$ on the advice needed to achieve a strict competitive ratio of c for minASGk. Note that this matches the upper bound from Theorem 8 up to an additive term of $O(\log n)$.

Before proving Theorem 10, we remark that there is a close connection between results on the competitive ratio and the strict competitive ratio:

▶ Remark 9. Suppose that a minASG algorithm, ALG, is c-competitive. By definition, there exists a constant, α, such that $\texttt{ALG}(\sigma) \leq c \cdot \texttt{OPT}(\sigma) + \alpha$. Then, one can construct a new algorithm, ALG′, which is *strictly* c-competitive and uses $O(\log n)$ additional advice bits as follows: Use $O(\log n)$ bits of advice to encode the length n of the input and use $\alpha \cdot \lceil \log n\rceil = O(\log n)$ bits of advice to encode the index of (at most) α rounds in which ALG guesses 1 but where the correct answer is 0. Clearly, ALG′ can use this additional advice to achieve a strict competitive ratio of c. This also means that a lower bound of b on the number of advice bits required to be *strictly* c-competitive implies a lower bound of $b - O(\log n)$ advice bits for being c-competitive (where the constant hidden in $O(\log n)$ depends on the additive constant α of the c-competitive algorithm). We will use this observation in Theorem 10. Note that the same technique can be used for maxASG.

▶ **Theorem 10.** *For any $c > 1$, a c-competitive algorithm for* minASGk *or* minASGu *must read at least b bits of advice, where*

$$b \geq \log\left(\max_{t:\lfloor ct\rfloor<n} \frac{\binom{n}{t}}{\binom{\lfloor ct\rfloor}{t}}\right) - O(\log n) = B(n,c) - O(\log n) \tag{3}$$

Proof (sketch). Fix n, c and let b be the maximum number of advice bits read by the strictly c-competitive algorithm, ALG, over all inputs of length n. Suppose, by way of contradiction, that there exists some t such that $\lfloor ct \rfloor < n$ and $b < \log(\binom{n}{t}/\binom{\lfloor ct \rfloor}{t})$. Let $I_{n,t}$ be the set of input strings of length n and Hamming weight t. We achieve the desired contradiction by showing that there is some input string in $I_{n,t}$ on which the algorithm incurs a cost of at least $\lfloor ct \rfloor + 1$.

Note that $|I_{n,t}| = \binom{n}{t}$. By the pigeonhole principle, there must exist some advice string, φ, such that the set of input strings, $I^{\varphi}_{n,t} \subseteq I_{n,t}$, of length n and Hamming weight t for which ALG reads the advice φ has size $\left|I^{\varphi}_{n,t}\right| > \binom{\lfloor ct \rfloor}{t}$.

We consider the computation of ALG, when reading the advice φ, as a game between ALG and an adversary, ADV. From the advice, ALG learns that the input string belongs to $I^{\varphi}_{n,t}$. Because of the known history, at the beginning of round i, ALG also knows the first $i-1$ bits of the input string. We say that a string $s \in I^{\varphi}_{n,t}$ is *alive* in round i if the first $i-1$ bits of s are identical to those revealed in the first $i-1$ rounds. If, in round i, there exists some string s such that s is still alive and $s_i = 1$, then ALG must answer 1. If not, ADV could pick s as the input string and hence ALG would incur a cost of ∞. On the other hand, if no such s exists in round i, we may assume (without loss of generality) that ALG answers 0. Intuitively, ADV wants to maximize the number of rounds where ALG is forced to answer 1 but where ADV can still give 0 as the correct answer.

Suppose that in some round, there are m strings from $I^{\varphi}_{n,t}$ which are alive. Let h be the number of 1s that have yet to be revealed in each of these strings (this is well-defined since all strings from $I^{\varphi}_{n,t}$ have the same number of 1s). We let $L_1(m,h)$ denote the minimum cost that the adversary can force ALG to incur in the remaining rounds when starting from this situation. The proof is finished by showing that for any $m, h \geq 1$,

$$L_1(m,h) \geq \min\left\{d : m \leq \binom{d}{h}\right\}. \tag{4}$$

Indeed, it follows from (4) that $L_1\left(\left|I^{\varphi}_{n,t}\right|, t\right) \geq \lfloor ct \rfloor + 1$. By induction on m and h, one can show that (4) is true by using the following adversary strategy: Let m the number of strings alive in the current round, i. Furthermore, let m_0 be the number of the m alive strings for which the ith bit is 0, and let $m_1 = m - m_0$. If $m_0 = m$, then ADV has to choose 0 as the correct bit in round i. If $m_0 < m$, let d_1 be the smallest integer such that $m_1 \leq \binom{d_1}{h-1}$ and let d be the smallest integer such that $m \leq \binom{d}{h}$. If $d_1 \leq d+1$, the adversary chooses 1 as the correct bit in round i and otherwise it chooses 0.

By Remark 9, the first inequality of (3) follows. The last equality follows from a simple but lengthy calculation showing that $\log\left(\max_{t:\lfloor ct \rfloor < n}\binom{n}{t}/\binom{\lfloor ct \rfloor}{t}\right) = B(n,c) - O(\log n)$. ◀

3.2 Advice Complexity of maxASG

The advice complexity of maxASG is the same as that of minASG, up to an additive $O(\log n)$ term. This is not immediately obvious, but one can show that computing a c-competitive solution for maxASG, on input strings where the number of 0s is u, requires roughly the same amount of advice as computing a c-competitive solution for minASG, on input strings where the number of 1s is $\lceil u/c \rceil$.

▶ **Theorem 11.** *For* maxASGu *and* maxASGk *and for any* $c > 1$*,*

$$b = B(n,c) \pm O(\log n) \tag{5}$$

bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c*.*

4 The Complexity Class AOC

In this section, we define a class of problems, AOC, and show that for each problem, P, in AOC, the advice complexity of P is at most that of ASG.

▶ **Definition 12.** A problem P is in AOC *(Asymmetric Online Covering)* if it can be defined as follows: The input to an instance of P consists of a sequence of n requests $\sigma = (r_1, \ldots, r_n)$ and possibly one final dummy request. An algorithm for P computes a binary output string $y = y_1 \ldots y_n \in \{0,1\}^n$, where $y_i = f(r_1, \ldots, r_i)$ for some function f.

For minimization (maximization) problems, the score function s maps a pair (σ, y) of input and output to a cost (profit) in $\mathbb{N} \cup \{\infty\}$ ($\mathbb{N} \cup \{-\infty\}$). For an input σ and an output y, y is *feasible* if $s(\sigma, y) \in \mathbb{N}$. Otherwise, y is *infeasible*. There must exist at least one feasible output. Let $S_{\min}(\sigma)$ ($S_{\max}(\sigma)$) be the set of those outputs that minimize (maximize) s for a given input σ.

If P is a minimization problem, then for every input σ, the following must hold:

1. For a feasible output y, $s(\sigma, y) = |y|_1$.
2. An output y is feasible if there exists a $y' \in S_{\min}(\sigma)$ such that $y' \sqsubseteq y$.
   If there is no such y', the output may or may not be feasible.

If P is a maximization problem, then for every input σ, the following must hold:

1. For a feasible output y, $s(\sigma, y) = |y|_0$.
2. An output y is feasible if there exists a $y' \in S_{\max}(\sigma)$ such that $y' \sqsubseteq y$.
   If there is no such y', the output may or may not be feasible.

The dummy request is a request that does not require an answer and is not counted when we count the number of requests. Most of the problems that we consider will not have such a dummy request, but it is necessary to make sure that ASG belongs to AOC.

The input σ to a problem P in AOC can contain any kind of information. However, for each request, an algorithm for P only needs to make a binary decision. If the problem is a minimization problem, it is useful to think of answering 1 as accepting the request and answering 0 as rejecting the request (e.g., vertices in a vertex cover). The output is guaranteed to be feasible if the accepted requests are a superset of the requests accepted in some optimal solution. If the problem is a maximization problem, it is useful to think of answering 0 as accepting the request and answering 1 as rejecting the request (e.g., vertices in an independent set). The output is guaranteed to be feasible if the accepted requests are a subset of the requests accepted in an optimal solution.

The key point of Definition 12 is that an ASGU algorithm works for every problem in AOC. Thus, by Theorems 8 and 11, we get the following upper bound on the advice complexity for problems in AOC.

▶ **Theorem 13.** *Let* P *be a problem in* AOC. *There exists a strictly c-competitive online algorithm for* P *reading b bits of advice, where*

$$b = \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n + O(\log n) = B(n,c) + O(\log n).$$

For all variants of ASG, we know that this upper bound is tight up to an $O(\log n)$ term. This leads us to the following definition of completeness.

▶ **Definition 14.** A problem P is AOC-*complete* if P belongs to AOC and if, for all $c > 1$, any c-competitive algorithm for P must read at least b bits of advice, where

$$b = \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n - O(\log n) = B(n,c) - O(\log n).$$

The constant hidden in $O(\log n)$ in Definition 14 is allowed to depend on the additive constant α of the c-competitive algorithm.

Note that the advice complexity of an AOC-complete problem must be identical to the upper bound from Theorem 13, up to a lower-order term of $O(\log n)$. By Theorems 10 and 11, all of minASGu, minASGk, maxASGu and maxASGk are AOC-complete. When we show that some problem P is AOC-complete, we do this by giving a reduction from a known AOC-complete problem to P, which preserves the competitive ratio and increases the number of advice bits by at most $O(\log n)$. ASGk is especially well-suited as a starting point for such reductions. We allow for an additional $O(\log n)$ bits of advice in Definition 14 in order to be able to use the reduction between the strict and non-strict competitive ratios as explained in Remark 9 and in order to encode some natural parameters of the problem, such as the input length or the score of an optimal solution. For most values of c, it seems reasonable to allow these additional advice bits. However, it does mean that for $c = \Omega(n/\log n)$, the requirement in the definition of AOC-complete is vacuously true.

5 Applications

By definition, showing that a problem is AOC-complete gives (almost) tight bounds on its advice complexity. We show that several natural online problems are AOC-complete.

Many of the problems that we consider are graph problems. Unless otherwise mentioned, the problems are studied in the *vertex-arrival model*. In this model, the vertices of an unknown graph are revealed one by one. That is, in each round, a vertex is revealed together with all edges connecting it to previously revealed vertices. For the problems we study in the vertex-arrival model, whenever a vertex, v, is revealed, an online algorithm `ALG` must (irrevocably) decide if v should be included in its solution or not. The individual graph problems are defined by specifying the set of feasible solutions. For all of the problems, the cost (or profit) of a feasible solution is the number of vertices in that solution. The cost (profit) of an infeasible solution is ∞ ($-\infty$). The problems we consider in this model are:

- Online Vertex Cover. A solution is feasible if all edges in the input graph have an endpoint at some vertex in the solution. The problem is a minimization problem.
- Online Cycle Finding. A solution is feasible if the subgraph induced by the vertices in the solution contains a cycle. We assume that the presented graph always contains a cycle. The problem is a minimization problem
- Online Dominating Set. A solution is feasible if each vertex in the input graph is in the solution or has a neighbor in the solution. The problem is a minimization problem.
- Online Independent Set. A solution is feasible if no two vertices in the solution are neighbors. The problem is a maximization problem.

We also consider the following online problems. Again, the cost (profit) of an infeasible solution is ∞ ($-\infty$).

- Online Disjoint Path Allocation. A path, P, is given. Each request is a subpath of P and must immediately be either accepted or rejected. A solution is feasible if the accepted subpaths are edge disjoint. The profit of a feasible solution is the number of accepted paths. The problem is a maximization problem.

- Online Set Cover (set-arrival version). A finite set U known as the *universe* is given. The input is a sequence of finite subsets of U, $(A_1, \ldots, A_n)$, where $\cup_{1 \leq i \leq n} A_i = U$. On arrival, a subset must either be accepted or rejected. Denote by S the set of indices of the subsets in some solution. The solution is feasible if $\cup_{i \in S} A_i = U$. The cost of a feasible solution is the number of accepted subsets. The problem is a minimization problem.

Note that the offline version of the problems we study have very different properties. Finding the shortest cycle in a graph can be done in polynomial time. There is a 2-approximation algorithm for finding a minimum vertex cover[3]. No $o(\log n)$-approximation algorithm exists for finding a minimum set cover (or a minimum dominating set), unless $\mathsf{P} = \mathsf{NP}$ [29]. For any $\varepsilon > 0$, no $n^{1-\varepsilon}$-approximation algorithm exists for finding a maximum independent set, unless $\mathsf{ZPP} = \mathsf{NP}$ [23].

The following theorem shows that all of the problems defined above are AOC-complete.

▶ **Theorem 15.** *For the problems* Online Vertex Cover, Online Cycle Finding, Online Dominating Set, Online Set Cover, Online Independent Set *and* Online Disjoint Path Allocation *and for any $c > 1$, possibly a function of the input length n,*

$$b = \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) n \pm O(\log n)$$

bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c.

It is easy to check that each of these problems belongs to AOC. To show completeness, we use reductions from ASG. Here, we sketch the reduction from minASGk to Online Vertex Cover[4]. For an input string $x = x_1 \ldots x_n \in \{0,1\}^n$, define $G_x = (V, E)$ as follows: $V = \{v_1, \ldots, v_n\}$ and $E = \{(v_i, v_j) \colon x_i = 1 \text{ and } i < j\}$. Furthermore, let $V_1 = \{v_i \colon x_i = 1\}$. The vertices will be revealed in the order $(v_1, \ldots, v_n)$. Note that $V_1 \setminus \{v_n\}$ is a minimum vertex cover of G_x. Also, if an algorithm rejects just a single vertex from V_1, it must accept all forthcoming vertices in order to produce a feasible solution. Using these observations, one can show that Online Vertex Cover is AOC-complete. The details, and the reductions for the remaining problems, can be found in the full version [9].

6 Conclusion and Open Problems

As with the original string guessing problem SG [5,16], we have shown that ASG is a useful tool for determining the advice complexity of online problems. It seems plausible that one could identify other variants of online string guessing and obtain classes similar to AOC. This could lead to an entire hierarchy of string guessing problems and related classes.

More concretely, there are various possibilities of generalizing ASG. One could associate some positive weight to each bit x_i in the input string. The goal would then be to produce a feasible output of minimum (or maximum) weight. Such a string guessing problem would model minimum weight vertex cover (or maximum weight independent set). Note that for maxASG, the algorithm from Theorem 6 works in the weighted version. However, the same is not true for any of the algorithms we have given for minASG. Thus, it remains an open problem if $O(n/c)$ bits of advice suffice to achieve a competitive ratio of c for the weighted version of minASG.

[3] We emphasize that the 2-approximation algorithm which greedily covers the edges (by selecting both endpoints) one by one cannot be used in the online vertex-arrival model.

[4] The graph used in this reduction is identical to the graph used in [21] to show hardness of Online Independent Set in the multi-solution model.

References

1. Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.
2. Kfir Barhum. Tight bounds for the advice complexity of the online minimum steiner tree problem. In *SOFSEM*, pages 77–88, 2014.
3. Kfir Barhum, Hans-Joachim Böckenhauer, Michal Forišek, Heidi Gebauer, Juraj Hromkovič, Sacha Krug, Jasmin Smula, and Björn Steffen. On the power of advice and randomization for the disjoint path allocation problem. In *SOFSEM*, pages 89–101, 2014.
4. Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. Online coloring of bipartite graphs with and without advice. *Algorithmica*, 70(1):92–111, 2014.
5. Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 2014.
6. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the k-server problem. In *ICALP (1)*, pages 207–218, 2011.
7. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *ISAAC*, pages 331–340, 2009.
8. Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.
9. Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. The Advice Complexity of a Class of Hard Online Problems. *arXiv*, 1408.7033 (cs.DS), August 2014.
10. Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. In *LATA*, pages 210–221, 2014.
11. Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. In *STACS*, pages 174–186, 2014. Full paper to appear in *Algorithmica*.
12. Marc Demange and Vangelis Th. Paschos. On-line vertex-covering. *Theor. Comput. Sci.*, 332(1-3):83–108, 2005.
13. Jeffrey H. Dinitz and Douglas R. Stinson, editors. *Contemporary Design Theory: a Collection of Surveys*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, New York, 1992.
14. Stefan Dobrev, Rastislav Královič, and Richard Královič. Independent set with advice: The impact of graph knowledge - (extended abstract). In *WAOA*, pages 2–15, 2012.
15. Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theor. Inf. Appl.*, 43(3):585–613, 2009.
16. Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.
17. Paul Erdős and Joel Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.
18. Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *LATA*, pages 228–239, 2012.
19. Sushmita Gupta, Shahin Kamali, and Alejandro López-Ortiz. On advice complexity of the k-server problem under sparse metrics. In *SIROCCO*, pages 55–67, 2013.
20. Magnús M. Halldórsson. Online coloring known graphs. *Electronic J. of Combinatorics*, 7(R7), 2000.
21. Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. Online independent sets. *Theor. Comput. Sci.*, 289(2):953–962, 2002.
22. Magnús M. Halldórsson and Hadas Shachnai. Return of the boss problem: Competing online against a non-adaptive adversary. In *FUN*, pages 237–248, 2010.

23 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.

24 Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *MFCS*, pages 24–36, 2010.

25 Sandy Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.

26 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

27 Dennis Komm, Richard Královič, and Tobias Mömke. On the advice complexity of the set cover problem. In *CSR*, pages 241–252, 2012.

28 Shuichi Miyazaki. On the advice complexity of online bipartite matching and online stable marriage. *Inf. Process. Lett.*, 114(12):714–717, 2014.

29 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

30 Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *CoRR*, abs/1311.7589, 2013.

31 Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In *CIAC*, pages 345–357, 2013.

32 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

33 Wen-Guey Tzeng. On-line dominating set problems for graphs. In D.-Z. Ding and Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, Boston, 1998.