# On the Advice Complexity of the Set Cover Problem

# On the Advice Complexity of the Set Cover Problem*

Dennis Komm[1], Richard Královič[1], and Tobias Mömke[2]

[1] Department of Computer Science, ETH Zurich, Switzerland,
{dennis.komm,richard.kralovic}@inf.ethz.ch
[2] School of Computer Science and Communication, KTH Royal Institute of Technology, Sweden,
moemke@kth.se

**Abstract.** Recently, a new approach to get a deeper understanding of online computation has been introduced: the study of the *advice complexity* of online problems. The idea is to measure the information that online algorithms need to be supplied with to compute high-quality solutions and to overcome the drawback of not knowing future input requests. This concept was successfully applied to many prominent online problems including paging, the $k$-server problem, metrical task systems, or job shop scheduling.
In this paper, we study the advice complexity of an online version of the well-known set cover problem introduced by Alon et al.: for a ground set of size $n$ and a set family of $m$ subsets of the ground set, we obtain bounds in both $n$ and $m$. We prove that $n$ bits of advice are both sufficient and necessary to perform optimally. Furthermore, for any constant $c$, we prove that $n - c$ bits are enough to construct a $c$-competitive online algorithm and this bound is tight up to a constant factor (only depending on $c$). Moreover, we show that a linear number of advice bits is both necessary and sufficient to be optimal with respect to $m$, as well. We further show lower and upper bounds for achieving $c$-competitiveness also in $m$.

## 1 Introduction and Preliminaries

The idea of online algorithms is both a natural concept in practical applications and of great theoretical interest. In this model, the algorithm designer faces the problem that an algorithm A has to read the input instance of some problem piecewise and has to create parts of the output before knowing the whole input string. Furthermore, A is not allowed to redeem any pieces of the output it already created.

Online scenarios and online algorithms have been studied extensively (for an overview, we refer the reader to the standard literature [1, 5, 9, 12, 14]). Similar to the concept of the approximation ratio of algorithms dealing with (hard) offline problems, *competitive analysis* (introduced in 1985 by Sleator and Tarjan [17]) is usually used to measure the quality of online algorithms (ignoring the runtime of the algorithm): for any instance $I$ of some minimization online problem $P$, we define the competitive ratio of A on $I$ as the fraction of the cost of A on $I$ and the optimal cost $\textsc{Opt}(I)$. Here, Opt is an *offline* algorithm that sees the whole input in advance, which is a huge advantage compared to the knowledge A possesses. This exactly captures the dilemma online algorithms face.

Clearly, we may see the competitive ratio as a value that tells us what we forfeit for not knowing the future. As pointed out in [4, 13], when dealing with the *advice complexity* of an online problem, we are interested in getting a deeper understanding of *what* it is we pay for. For sure, a straightforward answer to this question is "the remainder of the instance", but we want to measure the essence of what makes the problem hard with respect to online computation.

* This work was partially supported by ETH grant TH 18 07-3.

In the model used throughout this paper, we consider online algorithms that are able to receive extra information from some *advice tape* that is thought of as being created by an oracle O that sees the whole input in advance. In every time step, an online algorithm A may sequentially read some piece of information from this tape together with the input. More formally, we are dealing with the following model.

**Definition 1 (Online Algorithm with Advice).** *Consider an input sequence* $I = (x_1, \ldots, x_n)$. *An* online algorithm A with advice *computes the output sequence* $\mathcal{A}^\phi = \mathtt{A}^\phi(I) = (y_1, \ldots, y_n)$ *such that* $y_i$ *is computed from* $\phi, x_1, \ldots, x_i$, *where* $\phi$ *is the content of the advice tape, i. e., an infinite binary sequence.* A *is* $c$-competitive with advice complexity $s(n)$ *if there exists a constant* $\alpha$ *such that, for every* $n$ *and for each input sequence* $I$ *of length at most* $n$, *there exists some* $\phi$ *such that* $\mathrm{cost}(\mathtt{A}^\phi(I)) \leq c \cdot \mathrm{cost}(\mathtt{OPT}(I)) + \alpha$ *and at most the first* $s(n)$ *bits of* $\phi$ *have been accessed during the computation of* $\mathtt{A}^\phi(I)$. *Moreover, if* $\alpha = 0$, A *is called* strictly $c$*-competitive. An algorithm is* optimal *if it is strictly* 1*-competitive.*

In this paper, we consider *strictly* competitive algorithms only, when talking about both lower and upper bounds.

The advice complexity was introduced by Dobrev, Pardubská, and Královič [6, 7] in 2008 and a revised version was since then used to study various online problems. A detailed survey on this topic is given by Hromkovič, Královič, and Královič [13]. The paging problem and the disjoint path allocation problem have been examined in [4], the job shop scheduling problem was studied in [4,16]. Metrical task systems were investigated in [8], and the $k$-server problem was analyzed in [3,8]. Some connections between computing with advice and randomization are found in [3, 16].

As usual, in order to construct hard instances for the online algorithms studied, we consider an *adversary* ADV that constructs the input in a malicious way [5, 9, 12, 14]. As mentioned, we add another player to the game that is classically played between an online algorithm A and ADV: the oracle O that collaborates with A by giving help using the advice tape. Note that, similar to randomized online algorithms [12, 15], we can think of an online algorithm A with advice as an algorithm that chooses, depending on the advice it is given, from a set of deterministic strategies denoted by Alg(A). Another view we might impose on *computations with advice* is that an optimal random choice is supplied by O for every input.

**Observation 1** *Let* A *be an online algorithm that uses* $b$ *bits of advice. Then* ADV *can handle* A *as* $2^b$ *different deterministic online algorithms without advice. In particular,* ADV *knows each of the* $2^b$ *algorithms.*

This observation enables us to use two different techniques to show lower bounds on the number of advice bits required to obtain a certain output quality; for instance, in [4], the hardness was shown by a combinatorial argument: Suppose that there is an online algorithm A that uses $b$ bits of advice. Then there must be an advice tape such that reading the first $b$ bits enables A to distinguish a class of relevant inputs sufficiently for the required properties of A. The goal is then to show that there are different inputs that cannot be distinguished sufficiently. Using Observation 1, we can also do a direct proof. We are able to design an adversary ADV that creates an input instance such that all deterministic algorithms from Alg(A) fail to meet all required properties.

Throughout this paper, for any set $Y$, by $\mathcal{P}(Y)$ we denote the power set of $Y$; by $\log x$ we denote the logarithm of $x$ with base 2.

**Table 1.** Overview of the results presented in this paper.

| Measured | Quality | Lower Bound | Upper Bound |
|---|---|---|---|
| in $n = \lvert X\rvert$ | optimal | $n - \log(n-1)/2 - 3$ | $n-1$ |
| | $c$-competitive | $(n - c^2 - 2c - 1)\log\frac{c+1}{c}/(c^2+c)$ | $n - c + 2, c \geq 2$ |
| in $m = \lvert\mathcal{S}\rvert$ | optimal | $(m \log 3)/3 - 2$ | $m$ |
| | $c$-competitive | $\log m/c - 1$ | $m - \frac{(c-1)m}{\lceil \log m\rceil + c - 1} + 1$ |

In the following, we consider an online version of the set cover problem, SetCover for short, introduced by Alon et al. [2].

**Definition 2 (SetCover).** *Given a* ground set $X = \{1, \ldots, n\}$ *of size* $n$*, a set of* requests $X' \subseteq X$*, and a family* $\mathcal{S} \subseteq \mathcal{P}(X)$ *of size* $m$*, a feasible solution for* SetCover *is any subset* $\{S_1, \ldots, S_k\}$ *of* $\mathcal{S}$ *such that*

$$\bigcup_{i=1}^{k} S_i \supseteq X'.$$

*The aim is to minimize* $k$*, i. e., to use as few sets as possible. In the online version of this problem, the elements of* $X'$ *arrive successively one by one in consecutive time steps. An online algorithm* A *solves* SetCover *if, immediately after each yet uncovered request* $j$*, it specifies a set* $S_i$ *such that* $j \in S_i$*.*

Note that Alon et al. [2] constructed a deterministic $\mathcal{O}(\log m \log n)$-competitive algorithm that even works for the weighted version of SetCover. In the following, we give bounds on the advice complexity in both the size of the ground set $X$ and the size of the family $\mathcal{S}$. Obviously, $n = |X|$ is an upper bound on the number of requests. Furthermore, note that we may assume that $\mathcal{S}$ does not contain any sets that are subsets of any other set from $\mathcal{S}$ (if not, sets that are contained in other sets may be removed from the input by a preprocessing step; since the sets are unweighted, it is never worse to prefer supersets). From this, it directly follows, due to Sperner's Theorem [18], that we have

$$|\mathcal{S}| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor} =: N(n),$$

which trivially implies that $m$ may be exponential in $n$. However, due to the hardness of the vertex cover problem [11] (which is a sub-problem of SetCover) we know that also instances where $m \in \mathcal{O}(n)$ may be hard in the offline case. This motivates the study of advice depending on both $n$ and $m$.

Sometimes, when we need to communicate a number $x$ in a self-delimiting way, we use a well-known technique that enables us to use at most $2\lceil \log \lceil \log x \rceil \rceil + \lceil \log x \rceil$ bits (see, e. g., [4]).

### Organization of this Paper

In Section 2, we measure the advice complexity as a function of the size of the set of elements to cover, that is, $|X|$. We show that linear advice is both sufficient and necessary to create optimal output. As for the trade-off between an achievable constant competitive ratio $c$ and the number of advice bits, we show an upper bound of $n - c + 2$ and a lower bound that

matches up to some factor depending on $c$. In Section 3, we give bounds in the size of $\mathcal{S}$. Here, we also show that linear advice is necessary and sufficient to be optimal; this is tight up to a factor of $(\log 3)/3$. We prove a lower bound of $\log m/c - 1$ for achieving $c$-competitiveness and an upper bound of roughly $m - c \cdot m/\log m$. Our results are summarized in Table 1.

## 2 Bounds Measured in the Size of $X$

First, let us analyze the *information complexity* of the problem, that is, the advice complexity for calculating an optimal solution [13]. In the following, we show that a number of advice bits linear in $|X|$ is sufficient to create an optimal output by a very easy argument.

**Theorem 1.** *There exists an optimal online algorithm* A *with advice that uses* $n - 1$ *advice bits.*

*Proof.* The proof is straightforward. The oracle O simply writes the characteristic function of $X'$ on the advice tape, i. e., a one at position $i$ if and only if the $i$-th element from $X$ is contained in $X'$. However, A knows one requested element from $X$ before it has to take any action (and this is known by O). Thus, if the first element requested is $j$, the $j$th position is simply skipped on the tape (not written down by O). □

Although the above algorithm is trivial, it is interesting to note the we are able to complement this result with an almost matching lower bound to show that a linear number of advice bits is also necessary to be optimal.

**Theorem 2.** *At least* $\log(N(n-1)) > n - \frac{1}{2}\log(n-1) - 3$ *bits are necessary for any online algorithm* A *with advice to achieve optimality, for any even* $n$.

*Proof.* Let $n$ be even. Consider the set $\mathcal{S}$ that contains all subsets of $X$ of size exactly $n/2$. Clearly, there are exactly $N(n)$ such sets. Now ADV requests $n/2$ items, starting with one fixed item $x_1$ (after which A has to start with choosing a set from $\mathcal{S}$). Clearly, one single set from $\mathcal{S}$ is sufficient to cover all requests. Since A knows that $x_1$ is included in the set it has to choose, there are

$$\binom{n-1}{n/2-1}$$

remaining candidate sets. Observe that, since $n$ is even, it holds that $n/2 - 1 = \lfloor (n-1)/2 \rfloor$. Consequently, there are exactly $N(n-1)$ sets left from which A has to select one.

Therefore, A needs to distinguish between $N(n-1)$ different advices to distinguish this many sets and if it reads strictly less than $\log(N(n-1))$ bits, this merely enables A to choose among strictly less than $N(n-1)$ strategies which, by Observation 1, is equivalent to choosing among this many deterministic algorithms.

By the pigeonhole principle, there exist two distinct sets $S_1$ and $S_2$ (i. e., inputs which both contain $x_1$, but differ in at least one element) for which A chooses the same deterministic strategy $A \in \mathrm{Alg}(\mathtt{A})$. Clearly, $A$ cannot be optimal in both cases. Using Stirling's

approximation (see, e. g., [10]), we get

$$\begin{aligned}\log\left(N(n-1)\right) &= \log\left(\binom{n-1}{\lfloor\frac{n-1}{2}\rfloor}\right) \geq \log\left(\frac{4^{(n-1)/2}}{2\sqrt{(n-1)\pi/2}}\right)\\ &= (n-1) - \log\left(2\sqrt{(n-1)\pi/2}\right) > n - \log\left(\sqrt{8(n-1)}\right) - 1\\ &> n - \frac{1}{2}\log(n-1) - 3,\end{aligned}$$

which finishes the proof. □

The next question we are dealing with is how many bits are necessary and sufficient to achieve $c$-competitiveness.

**Theorem 3.** *There exists a $c$-competitive online algorithm* A *with advice that uses at most $n - ((c-1)k+1) + \lceil\log k\rceil + 2\lceil\log\lceil\log k\rceil\rceil$ bits of advice where $k$ is at most the size of the optimal solution.*

*Proof.* Suppose the optimal solution $\mathcal{O}pt$ covers all $|X'|$ requests with no more than $k$ elements from $\mathcal{S}$. The number $k$ can be communicated to A in a self-delimiting way using $\lceil\log k\rceil + 2\lceil\log\lceil\log k\rceil\rceil$ advice bits. Note that A may use at most $c\cdot\text{cost}(\mathcal{O}pt) \geq \text{cost}(\mathcal{O}pt) + (c-1)k$ sets to be $c$-competitive.

As in the proof of Theorem 1, O writes the characteristic function of $X'$ onto the advice tape. Recall that A knows the first element from $X'$ that is requested before it has to make any decision. Since A may use $(c-1)k$ additional sets in comparison to $\mathcal{O}pt$, it only needs to read the first $n-(c-1)k-1$ bits of advice, compute the optimal solution $\mathcal{O}pt^*$ for this sub-instance (which, of course, has less costs than $\mathcal{O}pt$) and take one additional set for every requested element not covered by $\mathcal{O}pt^*$. □

An easy calculation gives that, for $c \geq 2$, Theorem 3 implies the following corollary.

**Corollary 1.** *There exists a $c$-competitive online algorithm* A *with advice that uses at most $n - c + 2$ bits of advice, for $c \geq 2$.*

Next, generalizing the idea from the proof of Theorem 2, we show a lower bound on the number of advice bits required for any online algorithm that achieves $c$-competitiveness. To do so, we again consider an adversary ADV that plays against some online algorithm A with advice, i. e., against $2^b$ deterministic algorithms at once (see Observation 1).

**Theorem 4.** *Any deterministic online algorithm with advice that reads at most*

$$(n - c^2 - 2c - 1)\frac{\log\frac{c+1}{c}}{c^2+c},$$

*bits of advice is guaranteed to be strictly worse than $c$-competitive.*

*Proof.* For the ease of presentation, let $n$ be a multiple of $c+1$. In the following, for any given $c$, $\mathcal{S}$ consists of all possible sets of size $n/(c+1)$ in $\mathcal{P}(X)$. Similar to the previous discussion, an adversary ADV chooses exactly $n/(c+1)$ items such that there exists a unique optimal solution that consists of exactly one set. Thus, a $c$-competitive algorithm is allowed to choose $c$ sets at most.

Let A be an online algorithm that uses $b$ bits of advice. We will determine a number $b$ such that all algorithms in Alg(A) fail to be $c$-competitive, i. e., are forced to choose at least $c+1$ sets. To this end, ADV proceeds in rounds, where in each round all algorithms from Alg(A) are forced to choose an additional set. At the same time, there is an optimal solution that consists of only one single set. Thus, if each algorithm has to choose more than $c$ sets, none of them is $c$-competitive.

There are $c+1$ rounds numbered $0, \ldots, c$. At the end of round $i$, ADV has to ensure that all algorithms have chosen at least $i+1$ sets so far. In each round, the algorithms in Alg(A) are partitioned into those that already chose an additional set (these algorithms are said to be *out* for this round) and those that did not yet choose one additional set.

ADV creates an input instance $I = (i_1, \ldots, i_{n/(c+1)})$ that corresponds to exactly one set in $\mathcal{S}$. As in the proof of Theorem 2, the item $i_1$ is fixed as $x_1$ (in round zero, that is, before A produces any output) such that each algorithm in Alg(A) has to choose a set containing $x_1$. Subsequently, ADV enters the first round and it chooses items such that as many algorithms from Alg(A) as possible are out, because they do not have the corresponding item covered yet. ADV continues in this fashion until, eventually, the round is finished after all algorithms are out. The remaining rounds work analogously.

Let $p_0$ be 1 and $p_1, \ldots, p_c$ be natural numbers such that ADV chooses $p_i$ elements to finish round $i$, i. e., ADV chooses $p_i - p_{i-1}$ many items to make all algorithms in Alg(A) out in round $i$. At the beginning of round $r$, there are, in total, $2^b$ subsets of $\mathcal{S}$ that were determined by the algorithms from Alg(A) each of which containing the items of the earlier rounds. Suppose that there is no item that is covered by no algorithm; otherwise ADV requests exactly this one thus finishing round $r$. Clearly, we do not have to consider any algorithms (sets) that are out already at this step. Therefore, we may assume that each algorithm has chosen a total number of exactly $r$ sets and, thus, at most $rn/(c+1)$ items (otherwise, the corresponding algorithm is already out). From these items, $p_{r-1}$ are fixed already due to the previous rounds. Thus, ADV has to consider at most $2^b \cdot (rn/(c+1) - p_{r-1})$ items.

In the first step, ADV can choose from $n - p_{r-1}$ items. The average number of occurrences of one item is at most

$$\frac{2^b \left(\frac{rn}{c+1} - p_{r-1}\right)}{n - p_{r-1}}.$$

Therefore, the least frequently covered item $i_{p_{r-1}+1}$ has already been covered by at most that number of different algorithms in Alg(A). Since any algorithm is out if it has chosen a family of sets not containing this item, in the subsequent steps we only have to focus on the remaining algorithms in Alg(A).

More generally, ADV can reduce the number of algorithms that are not out in round $r$ by a fraction of

$$\frac{\frac{rn}{c+1} - j}{n - j} \tag{1}$$

or more when taking the $(j+1)$-th element. Hence, let $s \leq rn/(c+1) + 1$ be an integer such that

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1} - j}{n - j} < 1, \tag{2}$$

then, it follows that $p_r \leq s$. Note that, since $r < c+1$, $n > rn/(c+1) \geq s - 1$ holds in (2), thus the fraction is well-defined. Moreover, there exists $s$ such that (2) holds, e. g.,

$s = rn/(c+1) + 1$. Therefore, the value of (2) is at most

$$2^b \prod_{j=p_{r-1}}^{s-1} \frac{\frac{rn}{c+1}}{n} = 2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}}. \tag{3}$$

Hence, to satisfy (2), it is sufficient to choose $s$ such that

$$\begin{aligned} & 2^b \left(\frac{r}{c+1}\right)^{s-p_{r-1}} < 1 \\ \iff \quad & \log\left(\left(\frac{r}{c+1}\right)^{s-p_{r-1}}\right) < \log\left(2^{-b}\right) \\ \iff \quad & s - p_{r-1} > b\,\frac{1}{\log(c+1) - \log r} \end{aligned}$$

Following this, we can choose

$$s := \left\lfloor \frac{b}{\log(c+1) - \log r} \right\rfloor + 1 + p_{r-1}. \tag{4}$$

Clearly, `ADV` succeeds if it uses no more than $n/(c+1)$ items in total, which means that

$$1 + \sum_{r=1}^{c} (p_r - p_{r-1}) \leq \frac{n}{c+1}. \tag{5}$$

Since $p_r \leq s$, (5) is guaranteed by

$$1 + \sum_{r=1}^{c} \left( \left\lfloor \frac{b}{\log \frac{c+1}{r}} \right\rfloor + 1 \right) \leq \frac{n}{c+1}$$

which is implied by

$$\begin{aligned} & b \cdot \sum_{r=1}^{c} \frac{1}{\log \frac{c+1}{r}} \leq \frac{n}{c+1} - c - 1 \\ \iff \quad & b \leq \frac{\frac{n}{c+1} - c - 1}{\sum_{r=1}^{c} \frac{1}{\log \frac{c+1}{r}}} \end{aligned}$$

which again holds if

$$b \leq \frac{\frac{n-c^2-2c-1}{c+1}}{\frac{c}{\log \frac{c+1}{c}}} = (n - c^2 - 2c - 1)\frac{\log \frac{c+1}{c}}{c^2 + c}.$$

Hence, if $b$ is smaller than claimed in the statement of the theorem, `ADV` can make sure that any algorithm is worse than $c$-competitive, finishing the proof. □

Due to the fact that

$$(n - c^2 - 2c - 1)\frac{\log \frac{c+1}{c}}{c^2 + c} \in \mathcal{O}(n),$$

for any constant $c$, we immediately obtain the following statement.

**Corollary 2.** *Any online algorithm with advice that achieves a constant competitive ratio $c$, is required to use a number of advice bits linear in $n$. Therefore, the upper bound of Theorem 3 is tight up to a constant factor.*

## 3 Bounds Measured in the Size of $\mathcal{S}$

In the previous section, we measured the advice complexity in the number of elements, $|X|$. Now we look at the online SetCover problem from a different perspective, by measuring the advice in the number of sets given, $|\mathcal{S}|$. As we discuss at the end of this section, we obtain bounds that are not comparable with those presented in the previous section.

At first, let us consider the advice needed to create an optimal output. Encoding the characteristic function of the sets that are used by an optimal solution immediately gives the following result.

**Theorem 5.** *There exists an optimal online algorithm* `A` *that uses $m$ bits of advice.*

As in the previous section, this very naive approach is asymptotically the best we can hope for.

**Theorem 6.** *Any online algorithm* `A` *with advice needs at least $(m \log 3)/3 - 2$ advice bits to be optimal.*

*Proof.* For any $m$, let $m'$ be the largest number that is smaller than or equal to $m$ and that is a multiple of 3, i. e., $m' \geq m-2$. Moreover, let $n := 4m'/3$. `ADV` chooses $X$ such that there are $n/4$ items $y_1, \ldots, y_{n/4}$ and $3n/4$ items $x_{i,j}$, $i \in \{1,2,3\}$, $j \in \{1, \ldots, n/4\}$. After that, `ADV` sets $\mathcal{S}$ to contain exactly the following sets. For $k \in \{1, \ldots, n/4\}$, there are three sets $\{y_k, x_{1,k}\}$, $\{y_k, x_{2,k}\}$, and $\{y_k, x_{3,k}\}$. Since $\mathcal{S}$ has to be of size $m$, `ADV` adds $m - m'$ many *dummy* sets to $\mathcal{S}$ that each contain one unique item $y_j$ only; these sets are never considered by any optimal solution (which `A` may assume, because they are subsets of other sets).

Next, `ADV` requests all items $y_i$; hence, since each request has to be satisfied, `A` has to fix $n/4$ sets to cover all items. After that, for every $i$, `ADV` requests one of the three items $x_{1,i}$, $x_{2,i}$, and $x_{3,i}$. Note that any combination leads to a distinct optimal solution that consists of exactly $m'/3$ sets. This way, `A` has to correctly choose one of $3^{m'/3}$ possible families as answers for the first $m'/3$ requests.

If, however, `A` uses less than $\log 3^{m'/3}$ advice bits, then, by applying the pigeonhole principle, there must be two identical advices for two different sequences of correct answers. Thus, `ADV` can choose the one not chosen by the algorithm. Consequently, a lower bound on the advice complexity is

$$\log 3^{m'/3} \geq \frac{m-2}{3} \log 3 > \frac{m \log 3}{3} - 2$$

which is larger than $m/2$ for $m$ tending to infinity. □

Let us now consider algorithms that aim at achieving $c$-competitiveness.

**Theorem 7.** *For any $c \leq \log m + 1$,*

$$m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1} + 1$$

*bits are sufficient to achieve $c$-competitiveness.*

*Proof.* Suppose that $\mathcal{O}pt$ solves an arbitrary instance using exactly $k$ sets. Let $t := m/(\lceil \log m \rceil + c - 1)$. First, we observe that if $k \leq t$, it clearly suffices to communicate

$$\frac{m}{\lceil \log m \rceil + c - 1} \cdot \lceil \log m \rceil = m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1}$$

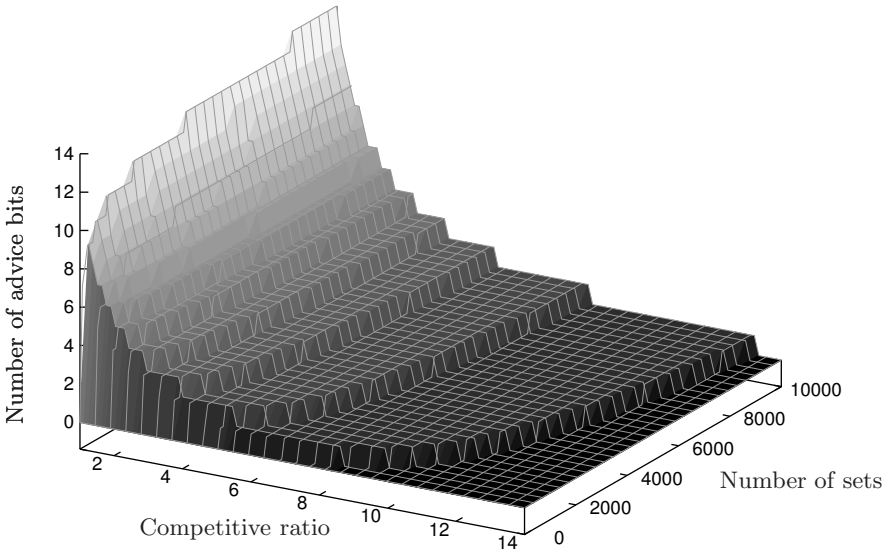


**Fig. 1.** The competitive ratio $c > 1$ and the advice bits $b$ required depending on $m$.

bits to A to be optimal by writing the indices of all sets taken by $\mathcal{O}pt$ on the advice tape. Clearly, we need one additional bit to indicate that this is the case. Note that we do not need to encode $k$ into the advice, but we can pad the advice by duplicating certain sets instead.

We may therefore assume the other case, i. e., $k > t$. Suppose that, in this case, O again writes the characteristic function of $\mathcal{S}$ on the advice tape. As in the proof of Theorem 3, A may use $(c-1)k \geq (c-1)t$ more sets than $\mathcal{O}pt$ to guarantee $c$-competitiveness. This means that, if A knows the first $m-(c-1)t$ bits of the advice tape, it acted optimal to this point and may safely take one set for each remaining uncovered element. We immediately verify that

$$m-(c-1)t = m - \frac{(c-1)m}{\lceil \log m \rceil + c - 1}.$$

Finally, as above, the number $k$ does not need to be communicated to A. Since $m$ and $c$ are known to A by construction, no further advice is necessary. □

We now show a lower bound on yielding $c$-competitiveness measured in $m$. This way, we obtain a slightly better lower bound than that one that is directly implied by the lower bound measured in $n$ from the previous section.

**Theorem 8.** *Any online algorithm* A *with advice needs to read at least* $\lceil \log m/c - 1 \rceil$ *bits of advice to be* $c$*-competitive.*

*Proof.* Let A use $b$ bits of advice and let $\alpha := 2^b$. ADV constructs a ground set $X$ that consists of all words with lengths up to $c$ (including the empty word $\lambda$) over an $(\alpha+1)$-ary alphabet. Every element $S$ from $\mathcal{S}$ is uniquely defined by a word $w(S)$ of length $c$. Moreover, $S$ contains *all* prefixes of $w(S)$. Note that, by definition, $\lambda$ is included in every set.

Without loss of generality, we assume that, in what follows, no algorithm chooses more than one set per request. Now at first, ADV requests the element $\lambda$ which causes every algorithm from Alg(A) to choose one set from $\mathcal{S}$. Every of these sets contains one element of length 1. However, the (at most) $2^b$ different choices left at least one such element $x'$ uncovered. ADV

now requests an element of length 2 that has the prefix $x'$. It immediately follows that every algorithm from Alg(A) has to choose a second set. ADV now inspects the words of length 2 within these sets. There is at least one word $x''$ (which has the prefix $x'$) not covered and ADV uses it as a prefix to choose a word of length 3 which it requests in what follows etc.

In general, after $i$ requests, all algorithms from Alg(A) used at least $i$ sets and thus ADV can ensure that A used $c$ sets in total after $c$ requests where all requested words are increasing prefixes of a distinct word $w$ of length $c$.

However, by construction, there exists *one* set in $\mathcal{S}$ that covers all requests, which is the unique optimal solution. We conclude that, in order to be $c$-competitive, $b = \log \alpha$ has to be chosen such that

$$m < (\alpha + 1)^c \iff m^{1/c} - 1 < 2^b \implies b > \log m/c - 1.$$

Hence, we may assume that $b$ is at least $\lceil \log m/c - 1 \rceil$ which proves our claim. The lower bound of the trade-off between $c$ and $b$ is shown in Figure 1. □

Compare these results to the ones measured in $n$ from the previous section. As for optimality, in Theorem 6, we had $4m/3 \geq n \geq 4(m-2)/3$, and thus $m \leq (3n+8)/4$ yielding a lower bound of less than $(n+3)\log 3/4$, which is worse than the one of Theorem 2. On the other hand, in the proof of Theorem 2, we had $m = N(n)$ which means that it merely gives a logarithmic lower bound in $m$, whereas Theorem 6 gives a linear lower bound. If we look at the trade-off between the advice bits and the competitive ratio, in Theorem 8, we clearly have $m = (\alpha+1)^c$ and $n = \sum_{i=0}^{c} (\alpha+1)^i = \frac{(\alpha+1)^{c+1}-1}{\alpha}$. Thus, Theorem 8 yields a lower bound that is logarithmic in $n$ and is therefore worse than the one of Theorem 4. Furthermore, inspecting this theorem, we observe that here $m = \binom{n}{n/(c+1)} \geq (c+1)^{n/(c+1)}$ and the bound of Theorem 8 is better with respect to $m$ and $c$.

## 4 Conclusion

In Section 2, we encoded information about the input to the advice tape such that A still had to compute the solution itself. Therefore, it was left with solving an $\mathcal{NP}$-hard problem. As usual in online computation, this concept of *unconditional hardness* is used to analyze the algorithms output quality while neglecting its runtime.

On the other hand, the upper bounds with respect to $|\mathcal{S}|$ presented in Section 3 encode information about the *solution*, and the resulting algorithms are running in polynomial time. In general, it makes sense to analyze the advice complexity of efficient online algorithms. Almost nothing is known about the comparison of online algorithms with advice with unrestricted and restricted runtime, opening an interesting field for further research.

## References


1. Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1):3–26, 2003.
2. Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.
3. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On the advice complexity of the $k$-server problem. In *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, volume 6755 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 2011.
4. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2011.
5. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6. Stefan Dobrev, Rastislav Královič, and Dana Pardubská. How much information about the future is needed? In *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. Springer-Verlag, 2008.
7. Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *Theoretical Informatics and Applications (RAIRO)*, 43(3):585–613, 2009.
8. Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.
9. Amos Fiat, Gerhard J. Woeginger (eds.). *Online Algorithms – The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
10. Ronald Graham, Donald Knuth, and Oren Patashnik. *Concrete Applied Mathematics*. Addison-Wesley, 2nd edition, 1994.
11. Juraj Hromkovič. *Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer-Verlag, 2nd edition, 2004.
12. Juraj Hromkovič. *Design and Analysis of Randomized Algorithms*. Springer-Verlag, 2005.
13. Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *Proc. of the 35th Symposium on Mathematical Foundations of Computer Science (MFCS 2010)*, volume 6281 of *Lecture Notes in Computer Science*, pages 24–36. Springer-Verlag, 2010.
14. Sandy Irani and Anna R. Karlin. On online computation. In *Approximation Algorithms for $\mathcal{NP}$-Hard Problems, chapter 13*, pages 521–564. PWS Publishing Company, 1997.
15. Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005.
16. Dennis Komm, Richard Královič. Advice complexity and barely random algorithms. *Theoretical Informatics and Applications (RAIRO)* 45(2):249–267, 2011.
17. Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
18. Emanuel Sperner. Ein Satz über Untermengen einer endlichen Menge. *Mathematische Zeitschrift*, 27(1):544–548, 1928.