Electronic Colloquium on Computational Complexity, Report No. 162 (2012)

# The String Guessing Problem as a Method to Prove Lower Bounds on the Advice Complexity*

Hans-Joachim Böckenhauer[1], Juraj Hromkovič[1], Dennis Komm[1], Sacha Krug[1], Jasmin Smula[1], and Andreas Sprock[1]

Department of Computer Science, ETH Zurich, Switzerland
{hjb,juraj.hromkovic,dennis.komm,sacha.krug,jasmin.smula,andreas.sprock}@inf.ethz.ch

**Abstract.** The advice complexity of an online problem describes the additional information both necessary and sufficient for online algorithms to compute solutions of a certain quality. In this model, an oracle inspects the input before it is processed by an online algorithm. Depending on the input string, the oracle prepares an advice bit string that may be accessed sequentially by the algorithm. The number of advice bits that are read to achieve some specific competitive ratio can then serve as a fine-grained complexity measure. The main contribution of this paper is to develop a new, powerful method to prove lower bounds on the number of advice bits necessary. To this end, we introduce the string guessing problem as a generic online problem and show a lower bound on the number of advice bits needed to obtain a small competitive ratio. We develop special reductions from string guessing to give a lower bound on the advice complexity of the online maximum clique problem and to improve the best known lower bound for the online set cover problem.

## 1 Introduction

Numerous computational problems work in so-called *online environments*, i. e., frameworks where the input arrives piecewise in successive time steps. An *online algorithm* has to answer every such piece by a part of the final output without knowing anything about the future requests (i. e., the rest of the input). In 1985, Sleator and Tarjan introduced the *competitive ratio* as a tool to measure the quality of such algorithms [19]. For an introduction to online computation and competitive analysis, we refer to the standard literature [3].

In this paper, we study the model of *computing with advice* to analyze how much information is necessary and sufficient to enable online algorithms to improve over purely deterministic strategies. The idea is to consider an oracle that sees the whole input in advance and writes binary information about this input onto an *advice tape* that may, at runtime, be accessed by the online algorithm. The idea of online computation with advice was introduced in [9]. Revised versions of this model were simultaneously introduced in [5, 13] and [10]. We follow the most general and exact model from [13] in this paper. The advice complexity was studied for various online problems in, e. g., [2, 4, 5, 6, 10, 11, 14, 15, 18].

**Definition 1 (Online Algorithm with Advice [5, 13]).** *Let $I = (x_1, \ldots, x_n)$ be an input of an online minimization problem. 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 $\phi$ is the content of the advice tape, i. e., an infinite binary sequence. For some output sequence $o$,* $\mathrm{cost}(o)$ *denotes the cost of $o$.* $\mathtt{A}$ *is* $c$-competitive with advice complexity $b(n)$ *if there exists some non-negative constant $\alpha$ such that, for every $n$ and every 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*

* This paper is partially funded by the SNF grant 200021–141089.

ISSN 1433-8092

*$b(n)$ bits of $\phi$ have been accessed during the computation of $\mathtt{A}^{\phi}(I)$. Here, $\mathtt{Opt}(I)$ denotes an optimal solution for $I$. If $\alpha = 0$, then $\mathtt{A}$ is called* strictly $c$-competitive. *$\mathtt{A}$ is* optimal *if it is strictly 1-competitive. The definition for maximization problems is analogous.*

The concept of advice complexity enables us to perform a much more fine-grained analysis of the hardness of online problems than using the classical competitive analysis. We are especially interested in *lower bounds* on the advice complexity. Such lower bounds do not only tell us something about the information content [13] of online problems, but they also carry over to a randomized setting where they imply lower bounds on the number of random decisions needed to compute a good solution [14]. However, as with the majority of computing models, lower bounds on the advice complexity are hard to prove. Thus, it is desirable to have some generic proof methods to establish lower bounds. In this paper, we take a first step towards this goal by introducing a generic online problem and showing how to transfer lower bounds on its advice complexity to lower bounds for other online problems.

### 1.1 Our Contribution

In this paper, we study the *string guessing problem* with respect to its advice complexity (Section 2). This problem is shown to be generic with respect to proving lower bounds on the advice complexity. Here, a string of length $n$ over an alphabet of size $q$ has to be guessed. More specifically, we define two versions of the problem. In the first case, the algorithm gets immediate feedback about which decisions would have been correct up to the current time step. In the second case, this feedback is not supplied. First, we prove a lower bound on the advice necessary to achieve some specific number of correct guesses for both versions. We show that the extra information of knowing the history does not help a lot for this class of problems. Additionally, we analyze the size of the advice depending on both $n$ and $q$.

Employing this result, we use the string guessing problem as a technique to prove lower bounds for other well-studied online problems. It seems to be a promising approach to use string guessing this way to show the hardness of further online problems.

Our first application, stated in Section 3, deals with an online version of the maximum clique problem where the vertices of the underlying graph arrive consecutively. In every time step, the online algorithm has to decide whether the current vertex is part of the solution or not. We give a lower bound on the number of advice bits necessary that is linear in the number of vertices. Second, in Section 4, we deal with an online version of the set cover problem introduced in [1]. We show how to use the results on the string guessing problem to give a lower bound that closes an exponential gap between the lower and upper bounds given in [15].

## 2 The String Guessing Problem

In many online problems, the question arises whether knowing the history, i. e., the parts of an optimal solution that correspond to the input known at a specific time step, has an effect on the additional information necessary to achieve a certain competitive ratio. We compare these two scenarios on a very generic online problem, namely the string guessing problem. In the first variant, the algorithm has to guess a character from some fixed alphabet, then, in the next step, it is told what would have been the correct answer and is asked for the next character. In the second variant, the algorithm also has to guess character by character, but it gets no feedback about whether its answer was correct or not, until the very end of the request sequence. In both cases, the length $n$ of the string is given as the first request and the algorithm then has to guess $n$ characters step by step.

In what follows, the *Hamming distance* between two strings of length $n$ over an alphabet of arbitrary size denotes the number of positions at which these two strings differ. Let us begin by defining the two variants of the string guessing problem formally.

**Definition 2 (String Guessing with Known History).** *The* string guessing problem with known history over an alphabet $\Sigma$ of size $q \geq 2$ *(*$q$-SGKH*) is the following online problem. The input $I = (n, d_1, d_2, \ldots, d_n)$ consists of a natural number $n$ and the characters $d_1, d_2, \ldots, d_n$, $d_i \in \Sigma$, that are revealed one by one. The online algorithm $\mathtt{A}$ computes the output sequence $\mathtt{A}(I) = y_1 y_2 \ldots y_n$, where $y_i = f(n, d_1, \ldots, d_{i-1}) \in \Sigma$, for some computable function $f$. The cost of a solution $\mathtt{A}(I)$ is the number of wrongly guessed characters, i. e., the Hamming distance* $\mathrm{Ham}(d, \mathtt{A}(I))$ *between $\mathtt{A}(I)$ and $d = d_1 d_2 \ldots d_n$.*

**Definition 3 (String Guessing with Unknown History).** *The* string guessing problem with unknown history over an alphabet $\Sigma$ of size $q \geq 2$ *(*$q$-SGUH*) is the following online problem. The input $I = (n, ?_2, ?_3, \ldots, ?_n, d)$, for $d = d_1, \ldots, d_n \in \Sigma^n$, consists of the input size $n$ in the first request and $n-1$ subsequent requests "?" carrying no extra information. In each of the first $n$ time steps, the online algorithm $\mathtt{A}$ is required to output one character from $\Sigma$, forming the output sequence $\mathtt{A}(I) = y_1 y_2 \ldots y_n$. In the last request, the string $d$ is revealed. The algorithm is not required to respond with any output in this time step. The cost of a solution $\mathtt{A}(I)$ is again the Hamming distance between $\mathtt{A}(I)$ and $d$.*

For simplicity, we sometimes speak about the input string $d = d_1 d_2 \ldots d_n$ when we mean the input sequence $I = (n, d_1, d_2, \ldots, d_n)$ or $I = (n, ?_2, ?_3, \ldots, ?_n, d)$ with $n = |d|$. Also, we write $\mathtt{A}(d)$ instead of $\mathtt{A}(I)$.

Since the cost of an optimal solution for any string guessing instance is always 0, it is not meaningful to consider the competitive ratio as a measure for these problems. We therefore restrict our analysis to the number of errors produced by an algorithm. Our goal is to minimize this number.

It is easy to see that, for every online algorithm without advice, there is an input string of length $n$ such that the algorithm produces $n$ errors. This holds for any alphabet of arbitrary size $q \geq 2$. To see this, consider an adversary $\mathtt{Adv}$ that, in each time step, produces an input character $\alpha_i$ differing from the deterministic output $y_i = f(n, \alpha_1, \ldots, \alpha_{i-1})$ of the algorithm. Clearly, no deterministic online algorithm gains anything by knowing the history.

Considering online algorithms with advice that produce optimal solutions, we easily see that each such algorithm needs to read at least $\lceil n \log_2 q \rceil$ advice bits to be optimal on any input of length $n$, even in the case of $q$-SGKH: Assume an optimal algorithm $\mathtt{A}$ reads $m < \lceil n \log_2 q \rceil$ advice bits. There are $q^n$ possible different input strings, but only $2^m \leq 2^{\lceil n \log_2 q \rceil - 1} < 2^{n \log_2 q} = q^n$ different advice strings. Thus, at least two different input strings $d = d_1 d_2 \ldots d_n$ and $d' = d'_1 d'_2 \ldots d'_n$ of length $n$ get the same advice. There is one position in the string where $d$ and $d'$ differ for the first time. The algorithm $\mathtt{A}$ makes a deterministic decision in the corresponding time step that is optimal for at most one of the two solutions and $\mathtt{Adv}$ can always choose the other one. A matching upper bound, even in the case of $q$-SGUH, can be achieved by simply enumerating all possible inputs and encoding the index of the concrete input using $\lceil n \log_2 q \rceil$ advice bits.

On the other hand, a constant amount of advice can already help to guess a linear number of characters correctly, even without considering the history.

**Observation 1** *There is an online algorithm for* $q$-SGUH *that guesses at least* $\lceil n/q \rceil$ *positions correctly on an input string of size* $n$ *using* $\lceil \log_2 q \rceil$ *advice bits.*

*Proof.* In every input string of length $n$ over an alphabet of size $q$, there is at least one character $z$ that occurs at least $\lceil n/q \rceil$ times, and it can be specified by the oracle using $\lceil \log_2 q \rceil$ bits. An online algorithm that outputs $z$ in every time step guesses at least $\lceil n/q \rceil$ positions correctly. □

In the remainder of this section, we estimate the number of advice bits necessary and sufficient to reach a specific cost.

## 2.1 Lower Bounds

First, we investigate a lower bound on the number of advice bits necessary to guarantee at most a specific number of wrong answers for $q$-SGUH. Consider an online algorithm using $b$ advice bits. This can be seen as a collection of $2^b$ different deterministic algorithms [15]. Since all possible inputs look the same on the first $n$ requests, the behavior of these algorithms can only depend on the advice.

For each of the $q^n$ possible inputs, the oracle can choose between $2^b$ different algorithms, each of which produces a fixed output string. The oracle has to construct a set of $2^b$ such strings, which we call center strings, in such a way that the maximum distance of any input string to the nearest of these center strings is minimized. This is exactly the task of constructing a so-called *covering code.* A covering code $K_q(n,r)$ for an alphabet $\Sigma$ of size $q$ of the strings of length $n$ with radius $r$ is defined as a set of codewords (elements of $\Sigma^n$) with the property that every string in $\Sigma^n$ has a distance smaller than or equal to $r$ to at least one codeword in $K_q(n,r)$. For an overview of covering codes, we recommend [7]. Thus, the minimum size of a covering code $K_q(n,r)$ gives us the number of different advice strings we need to make sure that the worst-case error over all inputs for $q$-SGUH is at most $r$.

To get a simple lower bound on the size of a covering code $K_q(n,r)$, we consider the Hamming balls of radius $r$ around the center strings. A Hamming ball of radius $r$ around a string $s$ in $\Sigma^n$ consists of all strings $t$ with $\mathrm{Ham}(s,t) \leq r$. Due to the symmetry of the hypercube, the size of a Hamming ball of radius $r$ around some string $s$ does not depend on $s$. Thus, we denote it by $\mathrm{Vol}_q(n,r)$. The number $b$ of advice bits needed to make sure that no error greater than $r$ occurs for any input string has to satisfy the condition

$$2^b \cdot \mathrm{Vol}_q(n,r) \geq q^n. \tag{1}$$

The volume of a Hamming ball is given by

$$\mathrm{Vol}_q(n,r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i \tag{2}$$

and can be estimated as follows.

**Lemma 1 (Guruswami et al. [12]).** *Let* $p \in \mathbb{R}$, $0 < p \leq 1 - 1/q$. *For sufficiently large* $n$, *we obtain* $\mathrm{Vol}_q(n,pn) \leq q^{H_q(p)n}$, *where* $H_q(p) = p\log_q(q-1) - p\log_q p - (1-p)\log_q(1-p)$ *is the* $q$*-ary entropy function.* □

An easy calculation immediately yields

$$\mathrm{Vol}_q(n, pn) \leq q^{H_q(p)n} = \left(\frac{q-1}{p}\right)^{pn} \left(\frac{1}{1-p}\right)^{(1-p)n}. \tag{3}$$

This observation leads to the following lower bound for $q$-SGUH.

**Theorem 1.** *Consider an input string of length $n$ for $q$-SGUH, for some $n \in \mathbb{N}$. The minimum number of advice bits for any online algorithm that can guarantee to be correct in more than $\alpha n$ characters, for $1/q \leq \alpha < 1$, is*

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha \log_q \alpha\right) n \log_2 q = (1 - H_q(1-\alpha))\, n \log_2 q.$$

*Proof.* Guessing at least $\alpha n$ characters correctly means there can be at most $(1-\alpha)n$ errors. We know from (1) and (2) that, in order to guarantee that the algorithm makes less than $r$ errors, we need at least $b$ advice bits such that

$$\frac{q^n}{2^b} \leq \sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

To give a lower bound on $b$, we define $\alpha' = 1 - \alpha$, substitute $r$ by $\alpha' n$ and, together with (3), we get

$$\frac{q^n}{2^b} \leq \sum_{i=0}^{\alpha' n} \binom{n}{i} (q-1)^i \leq \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n}.$$

After taking the logarithm to base $q$ on both sides, we get

$$\begin{aligned}
& n - \log_q 2^b \leq \alpha' n \log_q\left(\frac{(q-1)n}{\alpha' n}\right) + (n - \alpha' n)\log_q\left(\frac{n}{n-\alpha' n}\right) \\
\iff\quad & -b\log_q 2 \leq -\alpha' n \log_q(\alpha' n) + \alpha' n \log_q(q-1) + \alpha' n \log_q(n - \alpha' n) - n \\
& \qquad - n\log_q(n-\alpha' n) + n \log_q n \\
\iff\quad & b \log_q 2 \geq \alpha' n(\log_q(\alpha' n) - \log_q(q-1) - \log_q(n-\alpha' n)) \\
& \qquad + n(1 + \log_q(n-\alpha' n) - \log_q n) \\
\iff\quad & b \geq \big(1 + \alpha' \log_q(\alpha' n) + (1-\alpha')\log_q(n-\alpha' n) - \log_q n \\
& \qquad -\alpha' \log_q(q-1)\big)\, n \log_2 q.
\end{aligned}$$

We now resubstitute $\alpha'$ by $1-\alpha$ and finally obtain

$$\begin{aligned}
b &\geq (1 + (1-\alpha)\log_q((1-\alpha)n) + (1-(1-\alpha))\log_q(n-(1-\alpha)n) - \log_q n \\
&\quad - (1-\alpha)\log_q(q-1))n\log_2 q \\
&\geq (1 + (1-\alpha)\log_q(1-\alpha) + (1-\alpha)\log_q n + \alpha\log_q(\alpha n) - \log_q n \\
&\quad - (1-\alpha)\log_q(q-1))n\log_2 q \\
&\geq \big(1 + (1-\alpha)\log_q(1-\alpha) + \alpha\log_q\alpha - (1-\alpha)\log_q(q-1)\big)\, n\log_2 q \\
&\geq \left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log_2 q.
\end{aligned}$$

Thus, we have established a lower bound on $b$ to guarantee at least $\alpha n$ correct characters or, in other words, a maximal number of $\alpha' n$ errors. □

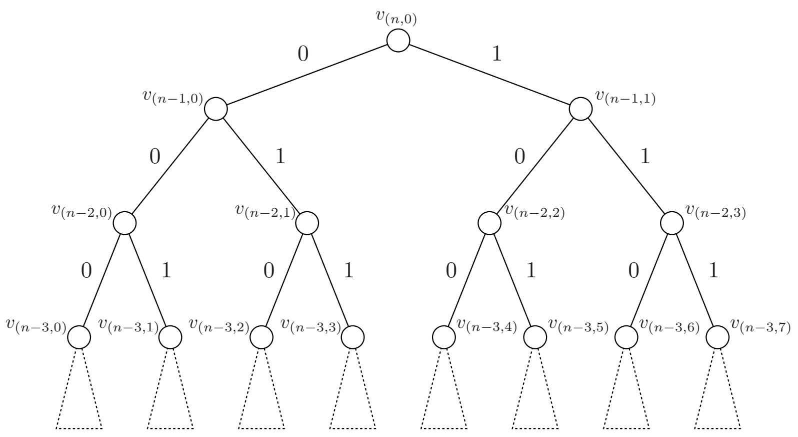

**Fig. 1.** Binary tree $T_n$ representing all input instances of size $n$ for $q = 2$.

The above argument heavily relies on the fact that, in case of $q$-SGUH, the output of a deterministic algorithm is unambiguously determined by the given advice. In the case of $q$-SGKH, this is no longer true. A deterministic algorithm might base its output on the history and thus might output different strings while reading the same advice. In the following we show that, despite this complication, the same lower bound as in Theorem 1 also holds for $q$-SGKH.

For the analysis, we use the $q$-ary tree $T_n$ of depth $n$ as a representation of the set $\Sigma^n$ of all input strings of length $n$ over the alphabet $\Sigma$ (see Figure 1). For $0 \leq i \leq q^n - 1$, the leaf $v_{(0,i)}$ represents the $i$th string in lexicographic order in $\Sigma^n$ and every inner vertex $v_{(h,i)}$ represents all $2^h$ strings of the leaves of the subtree rooted at $v_{(h,i)}$.

Let `A` be an online algorithm for $q$-SGKH that uses at most $b$ advice bits for any input instance of length $n$. Due to the pigeonhole principle, at least one advice string is used for at least $\lceil q^n/2^b \rceil$ different input instances. For a given advice string $s$ of length $b$, we now take a closer look at the set $\mathcal{I}_s$ of input strings for which `A` gets the advice string $s$. The algorithm `A` is not able to distinguish between any two strings in $\mathcal{I}_s$ at the beginning of the computation. However, this situation can change during the computation since `A` gets the additional information of what would have been the correct output in every time step.

For the analysis, we investigate the maximal cardinality of $\mathcal{I}_s$ such that `A` can guarantee that the maximal number of errors is $r$. We can view every computation of an online algorithm as a path in $T_n$ from the root down to a leaf. In every time step, the algorithm decides which subtree to enter. In the following step, it is revealed which direction would have been correct. If instances in more than one subtree of some vertex are represented by the given advice, the algorithm cannot know which subtree is correct.

For any vertex $v$ in $T_n$, let $F(v)$ denote the maximal number of errors the adversary `Adv` can enforce in the partial input string inside the subtree rooted at $v$, in addition to the errors already made on the way from the root to $v$. Moreover, let $\Phi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function such that $\Phi(h, r)$ measures how many strings in $\mathcal{I}_s$ can at most be represented by a vertex at depth $h$ such that the enforceable number of errors is at most $r$. We are interested in the value $\Phi(n, r)$ which gives us the desired lower bound. The function $\Phi(h, r)$ can be computed as stated by the following lemma.

**Lemma 2.** *For $0 \leq h \leq n$ and $0 \leq r \leq h$,*

$$\Phi(h,r) = \sum_{i=0}^{r} \binom{h}{i} (q-1)^i = \mathrm{Vol}_q(h,r).$$

*Proof.* The function $F$ can be computed recursively as follows. Let $v \in V$ be a vertex in $T_n$ such that the subtree rooted at $v$ contains at least one vertex that corresponds to a string from $\mathcal{I}_s$ and $\max\{F(u) \mid u \text{ is a child of } v\} = m$. Then,

$$F(v) = \begin{cases} m+1 & \text{if there are at least two children } u, w \text{ of } v \text{ with } F(u) = F(w) = m, \\ m & \text{else.} \end{cases} \tag{4}$$

To prove (4), we distinguish two cases. In the first case, there are two or more subtrees with the same maximal value of $F$. In the second case, there is exactly one subtree with a maximal number of errors.

**Case 1.** There are at least two children $u$ and $w$ of $v$ with $F(u) = F(w) = m$. Thus, it does not matter which subtree the algorithm chooses, because `Adv` will choose another subtree with a maximal number of errors and thus enforce one error in the current time step as well as the $m$ errors in the corresponding subtree.

**Case 2.** There is exactly one child $u$ of $v$ with $F(u) = m$ and, for all other children $w$ of $v$, $F(w) < m$. The algorithm should choose the subtree rooted at $u$ and accept $m$ errors. Otherwise, `Adv` would choose the subtree rooted at $u$ and thus enforce one error in the current time step and overall $m+1$ errors in the subtree rooted at $v$.

This proves (4). We now show that the function $\Phi(h,r)$ satisfies the recurrence relation

$$\Phi(h,0) = 1, \tag{5}$$

$$\Phi(h,h) = q^h, \text{and} \tag{6}$$

$$\Phi(h,r) = \Phi(h-1,r) + (q-1) \cdot \Phi(h-1,r-1), \text{ for } 0 < r < h. \tag{7}$$

To prove (5), assume there are two input strings represented by two leaves in $T_h$. These two leaves have a lowest common ancestor $v_{(g,j)}$, with $1 \leq g \leq h$. When the algorithm reaches $v_{(g,j)}$, it has to choose one of the $q$ successors. It does not matter which subtree the deterministic algorithm takes, `Adv` can always choose another one, i. e., another possible input string, and hence enforce one error. Thus, $\Phi(h,0) = 1$.

It is obvious that, if there are $h$ errors allowed, it does not matter what the algorithm does at depth $h$ because, in the worst case, the algorithm makes one error per step and thus at most $h$ errors in total. In other words, a subtree at depth $h$ with $h$ errors allowed can represent $q^h$ strings, i. e., $\Phi(h,h) = q^h$, proving (6).

Additionally, we know from (4) that, for a vertex $v$ at depth $h$ with a at most $r$ enforceable errors, the maximal number of errors in all $q$ subtrees of $v$ cannot be larger than $r$. Furthermore, we know that no two subtrees can contain $r$ errors. To maximize the number of errors in the subtree rooted at $v$, one child is assigned $r$ errors and all others $r-1$ errors. The maximal number of instances represented by a tree of depth $h$ when $r$ errors are allowed is thus $\Phi(h,r) = \Phi(h-1,r) + (q-1) \cdot \Phi(h-1,r-1)$, which proves (7).

Using this recurrence, we are now able to prove the claim of the lemma by induction on $h$. We already know that $\Phi(1,0) = 1 = \sum_{i=0}^{0} \binom{1}{i}(q-1)^i$ and $\Phi(1,1) = q = \sum_{i=0}^{1} \binom{1}{i}(q-1)^i$. Now we prove the statement for $h > 1$ and $0 \leq r \leq h$.

As induction hypothesis, assume that $\Phi(h,r) = \sum_{i=0}^{r} \binom{h}{i}(q-1)^i$. Note that $\Phi(h+1,r) = \Phi(h,r) + (q-1)\cdot\Phi(h,r-1)$ holds due to (7) and recall that $\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$. We get

$$\begin{aligned}
\Phi(h+1,r) &= \Phi(h,r) + (q-1)\cdot\Phi(h,r-1)\\
&= \sum_{i=0}^{r}\binom{h}{i}(q-1)^i + (q-1)\sum_{i=0}^{r-1}\binom{h}{i}(q-1)^i \qquad \text{(by the induction hypothesis)}\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h}{i}(q-1)^i + \sum_{i=0}^{r-1}\binom{h}{i}(q-1)^{i+1}\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h}{i}(q-1)^i + \sum_{i=1}^{r}\binom{h}{i-1}(q-1)^{i}\\
&= \binom{h}{0}(q-1)^0 + \sum_{i=1}^{r}\left(\binom{h}{i} + \binom{h}{i-1}\right)(q-1)^i\\
&= \binom{h+1}{0}(q-1)^0 + \sum_{i=1}^{r}\binom{h+1}{i}(q-1)^i\\
&= \sum_{i=0}^{r}\binom{h+1}{i}(q-1)^i.
\end{aligned}$$

It follows that

$$\Phi(h+1,h+1) = \sum_{i=0}^{h+1}\binom{h+1}{i}(q-1)^i = q^{h+1},$$

where the last equation holds due to the binomial theorem.

With this, the claim follows for all values of $\Phi(h+1,r)$, for $0 \le r \le h+1$. □

From Lemma 2 for $h = n$ together with Theorem 1, we immediately get the same lower bound on the advice complexity for $q$-SGKH, which we formulate in the following theorem.

**Theorem 2.** *Consider an input string of length $n$ for $q$-SGKH, for some $n \in \mathbb{N}$. The minimum number of advice bits for any online algorithm that can guarantee to be correct in more than $\alpha n$ characters, for $1/q \le \alpha < 1$, is*

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log_2 q = (1 - H_q(1-\alpha))\, n\log_2 q. \qquad \square$$

Let us give the following corollary for the *bit string guessing problem* (i. e., for $q = 2$).

**Corollary 1.** *Consider as input a bit string of length $n$ for 2-SGKH. Every deterministic algorithm that can guarantee to be correct in more than $\alpha n$ bits, for $1/2 \le \alpha < 1$, needs to read at least*

$$(1 + (1-\alpha)\log_2(1-\alpha) + \alpha\log_2\alpha)\, n$$

*many advice bits.* □

## 2.2 Upper Bounds

To give an upper bound on the advice complexity of $q$-SGUH on strings of length $n$ with at most $r$ errors, we analyze the minimal size of a covering code of length $n$ with radius $r$.

**Lemma 3 (Moser and Scheder [17]).** *Let $n \in \mathbb{N}^{>0}, q \in \mathbb{N}^{>1}, r \in \mathbb{N}$. On any alphabet $\Sigma$ of size $q$, there is a covering code of length $n$ with radius $r$ of size at most*

$$\left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n,r)} \right\rceil . \qquad \square$$

To estimate an upper bound on the advice to guarantee a certain number of correct characters, we need a lower bound on the volume of the Hamming ball of a given radius $r$.

**Lemma 4.** *Let $p \in \mathbb{R}$, $0 \leq p \leq 1 - 1/q$, such that $pn \in \mathbb{N}$. For sufficiently large $n$,*

$$\mathrm{Vol}_q(n, pn) \geq q^{H_q(p)\cdot n - \frac{1}{2}\log_q(2n)}.$$

*Proof.* We know from [16] that

$$\binom{n}{pn} \geq \frac{1}{\sqrt{8np(1-p)}} \cdot 2^{H_2(p)\cdot n}.$$

It follows that

$$\mathrm{Vol}_q(n,pn) = \sum_{i=0}^{pn} \binom{n}{i}(q-1)^i \geq \binom{n}{pn} \cdot (q-1)^{pn} \geq \frac{1}{\sqrt{8np(1-p)}} \cdot 2^{H_2(p)\cdot n} \cdot (q-1)^{pn}.$$

Together with the simple fact that $2^{H_2(p)\cdot n} \cdot (q-1)^{pn} = q^{H_q(p)\cdot n}$, we get

$$\mathrm{Vol}_q(n,pn) \geq \frac{q^{H_q(p)\cdot n}}{\sqrt{8np(1-p)}} \geq \frac{q^{H_q(p)\cdot n}}{\sqrt{2n}} = q^{H_q(p)\cdot n - \frac{1}{2}\log_q(2n)}$$

which finishes the proof. $\square$

Now we are ready to prove an upper bound on the number of advice bits sufficient to guarantee more than $\alpha n$ correctly guessed characters.

**Theorem 3.** *Consider an input of length $n$ for $q$-SGUH, for some $n \in \mathbb{N}$. There is an online algorithm that is correct in more than $\alpha n$ characters, for $1/q \leq \alpha < 1$, and needs at most*

$$\lceil (1 - H_q(1-\alpha))\, n \log_2 q + 3\log_2 n/2 + \log_2(\ln q) + 1/2 \rceil$$

*many advice bits.*

*Proof.* Guessing at least $\alpha n$ characters correctly means there can be at most $\alpha' n$ errors, for $\alpha' = 1 - \alpha$. To guarantee that there are at most $\alpha' n$ errors, we need to cover the strings of $\Sigma^n$ with Hamming balls of radius at most $\alpha' n$. We know from Lemma 3 that there is such a covering with at most

$$\left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n, \alpha' n)} \right\rceil$$

balls.

Such a covering leads to an algorithm that can guarantee that there are at most $\alpha' n$ errors and that uses $b$ advice bits such that $b$ is the smallest integer satisfying

$$2^b \geq \left\lceil \frac{n \cdot \ln q \cdot q^n}{\mathrm{Vol}_q(n, \alpha' n)} \right\rceil . \tag{8}$$

From Lemma 4, we know that

$$\mathrm{Vol}_q(n, \alpha' n) \geq q^{H_q(\alpha')\cdot n - \frac{1}{2}\log_q(2n)} \overset{(3)}{=} \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n} \cdot \frac{1}{\sqrt{2n}}.$$

Thus, it is sufficient for $b$ to satisfy

$$\frac{n \cdot \ln q \cdot q^n}{2^b} \leq \left(\frac{q-1}{\alpha'}\right)^{\alpha' n} \left(\frac{1}{1-\alpha'}\right)^{(1-\alpha')n} \cdot \frac{1}{\sqrt{2n}}.$$

After taking the logarithm to base $q$ on both sides, we get

$$\begin{aligned}\log_q n + n + \log_q(\ln q) - \log_q\left(2^b\right) \leq{}& \alpha' n \log_q\left(\frac{(q-1)n}{\alpha' n}\right) \\ &+ (n - \alpha' n)\log_q\left(\frac{n}{n-\alpha' n}\right) - \frac{1}{2}\log_q(2n)\end{aligned}$$

which is equivalent to

$$\begin{aligned}-b\log_q 2 \leq{}& -\alpha' n \log_q(\alpha' n) + \alpha' n \log_q(q-1) + \alpha' n \log_q(n - \alpha' n) - n \\ &- n\log_q(n-\alpha' n) + n \log_q n - \frac{1}{2}\log_q(2n) - \log_q(\ln q) - \log_q n.\end{aligned}$$

Dividing by $-\log_q 2$ yields

$$\begin{aligned}b \geq{}& (1 + \alpha' \log_q(\alpha' n) + (1-\alpha')\log_q(n - \alpha' n) - \log_q n - \alpha' \log_q(q-1))\, n \log_2 q \\ &+ \frac{1}{2}\log_2(2n) + \log_2(\ln q) + \log_2 n \\ ={}& \big(1 + \alpha' \log_q \alpha' + (1-\alpha')\log_q(1-\alpha') - \alpha' \log_q(q-1)\big)\, n\log_2 q + \frac{1}{2}\log_2(n) \\ &+ \log_2(\ln q) + \frac{1}{2} + \log_2 n \\ ={}& \big(1 + \alpha' \log_q \alpha' + (1-\alpha')\log_q(1-\alpha') - \alpha' \log_q(q-1)\big)\, n\log_2 q \\ &+ \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2} \\ ={}& \left(1 + \alpha' \log_q\left(\frac{\alpha'}{q-1}\right) + (1-\alpha')\log_q(1-\alpha')\right) n \log_2 q + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2}.\end{aligned}$$

We now resubstitute $\alpha'$ by $1 - \alpha$ and finally get

$$b \geq \left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha \log_q \alpha\right) n \log_2 q + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2}.$$

Since we wanted to find the minimum value for $b$ such that (8) is satisfied, we choose

$$b = \left\lceil \left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q \alpha\right) n \log_2 q + \frac{3}{2}\log_2 n + \log_2(\ln q) + \frac{1}{2} \right\rceil .$$

Hence, we now have an upper bound on the number $b$ of advice bits necessary for an algorithm to guarantee more than $\alpha n$ correctly guessed characters. □

For the special case of bit strings, we get the following result.

**Corollary 2.** *Consider as input a bit string of length $n$ for* 2-SGUH*. There is an online algorithm reading at most*

$$\left\lceil (1 + (1-\alpha)\log_2(1-\alpha) + \alpha\log_2\alpha)\, n + \frac{3}{2}\log_2 n + \frac{1}{2} + \log_2(\ln 2) \right\rceil$$

*advice bits and that is correct in more than $\alpha n$ bits, for* $1/2 \leq \alpha < 1$*.* □

## 3 The Online Maximum Clique Problem

In this section, we analyze the *online maximum clique* problem (MaxClique, see [8]). In MaxClique, in every time step, a vertex is given together with all edges to vertices that were already revealed in previous steps, and an online algorithm A has to decide whether the newly revealed vertex becomes part of the solution or not.

For giving a reasonable cost function, we briefly give some considerations. First, assume there is a maximum clique of size $n$ in the input graph $G$ and the algorithm finds a clique of size $n-1$. Then, intuitively, the cost of the solution should be $n-1$ irrespective of how many vertices of the found clique are also part of the largest clique in $G$. On the other hand, assume the algorithm selects a vertex that is not connected to any vertex revealed afterwards. Unless this is the only vertex A takes, A does not output a clique (i. e., its solution is not feasible). To avoid this, it should be allowed to give an output where, different to the situation in [8], not all selected vertices are part of a clique. Even if A gives an output in which many vertices form a large or even a maximum clique, but one additional vertex is selected, the output is no clique, but very close to a relatively good or even optimal solution. Thus, this solution should have almost optimal cost. Then again, we should clearly prevent the algorithm from simply selecting all vertices that are given.

Therefore, we consider, for an output $\mathtt{A}(I)$, the maximum clique $C_{\mathtt{A}(I)}$ in the graph $G_{\mathtt{A}(I)}$ restricted to the selected vertices $\mathtt{A}(I)$. Then, the solution becomes better the larger the maximum clique in $G_{\mathtt{A}(I)}$ is, and it becomes worse as more vertices are selected that are not part of $C_{\mathtt{A}(I)}$. All in all, we propose the cost function given in the following definition.

**Definition 4.** *The* online maximum clique problem (MaxClique) *is the following online problem. The input is a graph $G = (V, E)$ and the goal is to find a clique $C \subseteq V$ in $G$ of maximum size. In each time step $i$, one vertex $v_i \in V$ is revealed together with all edges $\{\{v_i, v_j\} \in E \mid j < i\}$, and the online algorithm A has to decide whether $v_i \in C$ or not. Let $\mathtt{A}(I)$ be the set of vertices selected by A and let $C_{\mathtt{A}(I)}$ be a maximum clique in the graph $G_{\mathtt{A}(I)}$. The cost function is defined by* $\mathrm{cost}(\mathtt{A}(I)) = \left|C_{\mathtt{A}(I)}\right|^2 / \left|\mathtt{A}(I)\right|$.

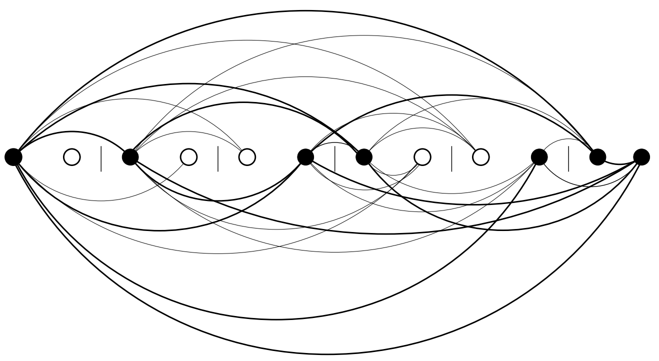

**Fig. 2.** Example of the graph $G_{00101}$.

Clearly, for the optimal solution $\texttt{Opt}(I)$ of a graph with a maximum clique $C_{\text{opt}}$, we have

$$\text{cost}(\texttt{Opt}(I)) = \frac{|C_{\text{opt}}|}{|\texttt{Opt}(I)|} \cdot |C_{\text{opt}}| = |C_{\text{opt}}|.$$

Thus, the competitive ratio $c$ of $\texttt{A}$ on $I$ can be computed as

$$c = \frac{\text{cost}(\texttt{Opt}(I))}{\text{cost}(\texttt{A}(I))} = \frac{|\texttt{A}(I)|}{|C_{\texttt{A}(I)}|} \cdot \frac{|C_{\text{opt}}|}{|C_{\texttt{A}(I)}|}.$$

In other words, the quality of the algorithm is given by the product of the two ratios $|\texttt{A}(I)|/|C_{\texttt{A}(I)}|$ and $|C_{\text{opt}}|/|C_{\texttt{A}(I)}|$. The first ratio measures how many useless vertices the algorithm has taken and the second ratio measures how many correct vertices the algorithm did not take.

In order to give a lower bound on the advice complexity of MaxClique, we use our results for 2-SGKH. To this end, we investigate the following subclass of instances, where every instance corresponds to a particular bit string. Let $s = s_1 s_2 \ldots s_{n'}$ be a bit string of length $n'$, for some $n' \in \mathbb{N}$. We construct an input instance $I_s$ for MaxClique corresponding to $s$ as follows. Consider the graph $G_{I_s} = (V(I_s), E(I_s))$ with $n = 2n' + 2$ vertices. Let

$$V(I_s) = \{v_{(1,0)}, v_{(1,1)}, v_{(2,0)}, v_{(2,1)}, \ldots, v_{(n'+1,0)}, v_{(n'+1,1)}\}$$

and let $V'(I_s) = \{v_{(i,s_i)} \mid 1 \leq i \leq n'\}$ be the set of the $n'$ vertices that correspond to the string $s$. Moreover,

$$\begin{aligned} E(I_s) = \{\{v_{(i,s_i)}, v_{(j,k)}\} \mid 1 \leq i < j \leq n', k \in \{0,1\}\} \\ \cup \{\{v, v_{(n'+1,0)}\}, \{v, v_{(n'+1,1)}\} \mid v \in V'(I_s)\} \cup \{\{v_{(n'+1,0)}, v_{(n'+1,1)}\}\}. \end{aligned}$$

Clearly, the vertices from $V'(I_s)$ plus the vertices $v_{(n'+1,0)}$ and $v_{(n'+1,1)}$ form a unique optimal solution for $I_s$ of size $n' + 2$. Although the vertices $v_{(i,0)}$ and $v_{(i,1)}$ are revealed separately (and after each vertex, the algorithm has to respond immediately), for the analysis, we combine them to a pair. After the first pair is revealed, the vertices of the second pair $(v_{(2,0)}, v_{(2,1)})$ are given and so on. An example for the string $s = 00101$ of length 5 is given in Figure 2.

Assume that $\texttt{A}$ knows that one vertex of each pair is part of the optimal solution. Then, we can see MaxClique as guessing one vertex per pair. Similar to guessing a string $s$, also when trying to find the correct vertex in a pair $(v_{(i,0)}, v_{(i,1)})$ in $G_{I_s}$, the correct decision can only depend on the input known so far, the history, and the given advice. However, in general, an algorithm has four options for every pair that is revealed, which are to take the first vertex, the second one, both, or none. As a next step, we show that, for any online algorithm with advice, it is the best strategy to take both vertices of any pair for which no advice is used. In this way, we derive an upper bound on the cost of any online algorithm for MaxClique that uses at most $b$ advice bits.

**Lemma 5.** *Let $s$ be an instance for 2-SGKH of length $n'$ and let $\mathtt{B}$ be the best online algorithm for 2-SGKH that reads $b$ advice bits. Let the number of bits that $\mathtt{B}$ guesses correctly be at most $\alpha n'$ where $0 \leq \alpha \leq 1$. Then, there is no online algorithm $\mathtt{A}$ for a corresponding MaxClique instance $I_s$ that reads $b$ advice bits with*

$$\mathrm{cost}(\mathtt{A}(I_s)) > \frac{(\alpha n + 2 + (1-\alpha)n')^2}{\alpha n' + 2 + 2(1-\alpha)n'}.$$

*Furthermore, an online algorithm $\mathtt{A}^*$ that correctly guesses the same pairs as $\mathtt{A}$ and takes both vertices for all remaining pairs satisfies $\mathrm{cost}(\mathtt{A}^*(I_s)) \geq \mathrm{cost}(\mathtt{A}(I_s))$.*

*Proof.* First we prove by a reduction from the string guessing problem that no algorithm for MaxClique can correctly guess more than $\alpha n'$ pairs when using at most $b$ advice bits. Consider any algorithm $\mathtt{A}$ for MaxClique such that $\tilde{\alpha} n'$ is the number of pairs $(v_{(i,0)}, v_{(i,1)})$ of vertices in $G_{I_s}$ that $\mathtt{A}$ guessed correctly. For the sake of a contradiction, suppose $\tilde{\alpha} > \alpha$ and consider the following reduction to solve 2-SGKH with $\tilde{\alpha} n'$ correctly guessed bits. Every time step in 2-SGKH can be transformed into two time steps of MaxClique by the above transformation. We then create an online algorithm $\mathtt{A}'$ for 2-SGKH as follows. According to the output of $\mathtt{A}$ in the two time steps that are associated with one pair, $\mathtt{A}'$ gives the output 0 if $\mathtt{A}$ takes the first vertex and 1 otherwise. Thus, $\mathtt{A}'$ is an online algorithm with advice for 2-SGKH that guesses more than $\alpha n'$ bits correctly while using $b$ advice bits and that is hence strictly better than $\mathtt{B}$, which is a contradiction to our assumption.

It follows that $\tilde{\alpha} \leq \alpha$. We may thus assume that $\mathtt{A}$ guesses exactly $\alpha n'$ pairs correctly for MaxClique, which is, by the above reasoning, the best $\mathtt{A}$ can do. Additionally, we may assume that $\mathtt{A}$ also knows where these $\alpha n'$ pairs lie in the instance $I_s$. For the rest of the $(1-\alpha)n'$ requests, suppose that $\mathtt{A}$ takes, for a fraction of $\beta$, both vertices of the corresponding pair, for a fraction of $\gamma$ the wrong one, and, for the remainder, no vertex at all. Thus, $\mathtt{A}$ outputs a solution of size $\alpha n' + 2 + 2(1-\alpha)\beta n' + (1-\alpha)\gamma n'$ while there is a clique $C_{\mathtt{A}(I_s)}$ in $G_{\mathtt{A}(I_s)}$ of size $\alpha n' + 2 + (1-\alpha)\beta n'$ yielding

$$\mathrm{cost}(\mathtt{A}(I_s)) = \frac{(\alpha n' + 2 + (1-\alpha)\beta n')^2}{\alpha n' + 2 + (1-\alpha)(2\beta + \gamma)n'}.$$

We immediately observe that this term does not depend on the number of pairs for which $\mathtt{A}$ chooses no vertex and that it decreases with increasing $\gamma$. We therefore set $\gamma$ to 0 and verify that the remaining term increases with $\beta$. To this end, let us substitute $X = \alpha n' + 2$ and $Y = (1-\alpha)n'$ and consider the function

$$f(\beta) = \frac{(\alpha n' + 2 + (1-\alpha)\beta n')^2}{\alpha n' + 2 + (1-\alpha)2\beta n'} = \frac{(X + Y\beta)^2}{X + 2Y\beta}.$$

Since

$$f'(\beta) = \frac{2Y(X + Y\beta)(X + 2Y\beta) - 2Y(X + Y\beta)^2}{(X + 2Y\beta)^2}$$

is positive for all values of $\alpha$ and $\beta$ between 0 and 1, we may set $\beta$ to 1. In other words, the best algorithm $\mathtt{A}^*$ that makes the right decisions on exactly the same set of pairs as $\mathtt{A}$ takes both vertices for all remaining pairs. □

In order to give a lower bound on the advice complexity, we analyze an online algorithm with advice that gets a sufficiently large number of advice bits to know $\alpha n$ pairs and, following Lemma 5, takes both vertices for all unknown positions. Using our results from Section 2 we can prove the following theorem.

**Theorem 4.** *Any* $(c - \varepsilon)$*-competitive online algorithm* $\mathtt{A}$ *for* MaxClique *needs at least*

$$(1 + (c - 1)\log_2(c - 1) + (2 - c)\log_2(2 - c))\,\frac{n - 2}{2} = (1 - H_2(c - 1))\frac{n - 2}{2}$$

*advice bits, for any* $1 < c \le 1.5$ *and* $\varepsilon > 0$.

*Proof.* Let $n' = (n-2)/2$. As above, assume that $\mathtt{A}$ reads a sufficiently large number of advice bits to correctly guess $\alpha n'$ pairs. In order to give a lower bound, we again assume that $\mathtt{A}$ also knows where these $\alpha n'$ pairs lie in the instance and that, according to Lemma 5, $\mathtt{A}$ takes both vertices for all pairs where the corresponding bit is unknown. Thus,

$$\mathrm{cost}(\mathtt{A}(I)) = \frac{(\alpha n' + 2 + (1 - \alpha)n')^2}{\alpha n' + 2 + 2(1 - \alpha)n'} = \frac{n'^2 + 2n' + 4}{2n' - \alpha n' + 2} = \frac{n' + 2 + \frac{4}{n'}}{2 - \alpha + \frac{2}{n'}}.$$

For the competitive ratio, we therefore get

$$c = \frac{\mathrm{cost}(\mathtt{Opt}(I))}{\mathrm{cost}(\mathtt{A}(I))} = \frac{(n' + 2)(2 - \alpha + \frac{2}{n'})}{n' + 2 + \frac{4}{n'}} > \frac{n'(2 - \alpha)}{n' + 2 + \frac{4}{n'}} = \frac{(2 - \alpha)}{1 + \frac{2}{n'} + \frac{4}{n'^2}}.$$

For any $\alpha$ and any $\varepsilon > 0$, we have

$$c \ge \frac{(2 - \alpha)}{1 + \frac{2}{n'} + \frac{4}{n'^2}} \ge (2 - \alpha - \varepsilon),$$

for all sufficiently large $n'$. In other words, $\mathtt{A}$ has to guess at least $\alpha n'$ characters correctly to reach a competitive ratio of $2 - \alpha - \varepsilon$. Using Corollary 1, we get a lower bound on the number of advice bits necessary to reach a competitive ratio of $c$ of

$$\begin{aligned}
&(1 + (1 - (2 - c))\log_2(1 - (2 - c)) + (2 - c)\log_2(2 - c))\,n' \\
&= (1 + (c - 1)\log_2(c - 1) + (2 - c)\log_2(2 - c))\,n' \\
&= (1 + (c - 1)\log_2(c - 1) + (2 - c)\log_2(2 - c))\,\frac{n - 2}{2}
\end{aligned}$$

as we claimed. □

Note that, without advice, an online algorithm for MaxClique can reach a competitive ratio of $(n' + 2)/((n' + 2)^2/(2n' + 2)) = (2n' + 2)/(n' + 2) \approx 2$ by just taking every vertex.

## 4 The Online Set Cover Problem

In this section, we study the advice complexity of SetCover. The (unweighted) online set cover problem, introduced in [1], is defined as follows.

**Definition 5 (Online Set Cover Problem).** *Given a* ground set $X = \{1, 2, \ldots, n\}$ *of size* $n$*, a set of* requests $X' \subseteq X$*, and a set family* $\mathcal{S} \subseteq \mathcal{P}(X)$ *of size* $m$*, a feasible solution for the online set cover problem (*SetCover*) is any subset* $\{S_1, \ldots, S_k\}$ *of* $\mathcal{S}$ *such that* $\bigcup_{i=1}^{k} S_i \supseteq X'$*. We may assume, without loss of generality, that no set in* $\mathcal{S}$ *is the subset of another set in* $\mathcal{S}$*. The aim is to minimize* $k$*, i. e., to use as few sets as possible. The set* $X$ *and the family* $\mathcal{S}$ *are known beforehand, but 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* $S_i \in \mathcal{S}$ *such that* $j \in S_i$*.*

We now use our results from Section 2 to give a lower bound on the advice necessary to achieve a specific competitive ratio that improves over the best known lower bounds in both $|X|$ and $|\mathcal{S}|$. More specifically, in [15], a lower bound on achieving a competitive ratio of $c$ was shown that is merely logarithmic in $m$. The following results yield an exponential improvement.

**Theorem 5.** *Assume that there is an algorithm* A *that solves* SetCover *with* $b$ *advice bits making at most* $r$ *errors, i. e., choosing at most* $r$ *sets more than an optimal algorithm. Then there also is an algorithm* B *that solves the string guessing problem with known history with* $b$ *advice bits and at most* $r$ *errors.*

*Proof.* First, we show how an instance $I_B$ for $q$-SGKH over an alphabet $\Sigma$ of size $q$ can be transformed into an instance $I_A$ for SetCover. Let the considered instance for $q$-SGKH be $I_B = (k, d_1, \ldots, d_k)$. Hence, the input string $d = d_1 \ldots d_k$ has length $k$. We give the set $X$ and the family $\mathcal{S}$, which are known to the SetCover algorithm, depending on $k$ and $\Sigma$: $X = \{X_t \mid t \text{ is a string over } \Sigma \text{ of length at most } k\}$. Hence, $X$ contains $\sum_{i=0}^{k} q^i = q^{k+1} - 1$ elements. Furthermore, $\mathcal{S} = \{\mathrm{tr}(s) \mid s = s_1 \ldots s_k \in \Sigma^k\}$, where $\mathrm{tr}(s) = \{X_t \mid t \text{ is a prefix of } s\} = \{X_\lambda, X_{s_1}, X_{s_1 s_2}, \ldots, X_{s_1 \ldots s_k}\}$ is the *transformation* of the string $s$ and $\lambda$ denotes the empty string. Each set $\mathrm{tr}(s)$ contains $k+1$ elements, and $\mathcal{S}$ consists of $q^k$ sets.

It remains to show that, in each time step $j$, we can transform the $j$th request of the $q$-SGKH instance into a request of the SetCover instance, and can also transform the output of a SetCover algorithm A into an output of a $q$-SGKH algorithm B such that the following condition is satisfied: If A uses $b$ advice bits and makes at most $r$ errors, then B also uses at most $b$ advice bits and makes at most $r$ errors.

The request for B in time step $2 \leq j \leq n+1$ consists of the character $d_{j-1}$. Hence, by time step $j$, algorithm B is aware of the first $j-1$ characters $d_1 \ldots d_{j-1}$ of the correct string. Then, the element that is requested to be covered by the SetCover algorithm A in time step $j$ will be $X_{d_1 \ldots d_{j-1}}$. Note that by this, A is implicitly informed about which character would have been the right choice in time step $j-1$. The element $X_{d_1 \ldots d_{j-1}}$ might already be covered, namely if A already chose a set $S_i = \mathrm{tr}(s)$ for some string $s$ with prefix $d_1 \ldots d_{j-1}$ in one of the previous time steps. If it is not covered yet, A has to choose a set containing $X_{d_1 \ldots d_{j-1}}$ now. Since every set $S_i$ corresponds to a certain string $s \in \Sigma^k$, the choice of a set containing element $X_{d_1 \ldots d_{j-1}}$ corresponds to guessing all the remaining characters of the string $d$, in particular the character $d_j$. If, in time step $j$, A chooses a set $S_i$ containing the element $X_{d_1 \ldots d_{j-1} y_j}$, the string guessing algorithm B will output the character $y_j$ as its guess for the $j$th character of $d$.

If A has made an error in time step $j-1$, this means that it has picked a set containing $X_{d_1 \ldots d_{j-2} y_{j-1}}$ for some $y_{j-1} \neq d_{j-1}$. This error will be noticed by A in time step $j$, when it gets the request $X_{d_1 \ldots d_{j-1}}$. It now has to pick one set to cover this element. If A did not make an error in time step $j-1$, it has picked a set containing $X_{d_1 \ldots d_{j-2} d_{j-1}}$. In time step $j$, the request sent to A is exactly this element $X_{d_1 \ldots d_{j-2} d_{j-1}}$. Because it is already covered, no additional set has to be chosen.

Now let us consider the number of errors of the string guessing algorithm B. If the output of A in time step $j-1$ is a set containing $X_{d_1 \ldots d_{j-2} y_{j-1}}$, B guesses the character $d_{j-1}$ to be $y_{j-1}$ in time step $j-1$. If and only if $X_{d_1 \ldots d_{j-2} y_{j-1}}$ is the request for A in the next time step, it holds that $d_{j-1} = y_{j-1}$. So, B guesses the $(j-1)$th character correctly if and only if the SetCover algorithm A did not make an error in time step $j-1$ and therefore does not have to pick another set in time step $j$.

All in all, we have shown that the string guessing algorithm B makes an error in time step $j$ if and only if the SetCover algorithm A makes an error in time step $j$. □

The reduction above helps us to establish a lower bound on the advice complexity of SetCover for higher competitive ratios. First, we show that it is equally hard to guess a percentage of $\alpha$ characters over one string of length $rk$ as to guess the same percentage over $r$ strings of length $k$ over an alphabet of the same size. We start with formally defining the problem of guessing $r$ strings of size $k$.

**Definition 6 ((*q*,*r*,*k*)-Multiple String Guessing with Known History).** *The $(q,r,k)$-multiple string guessing problem with known history over an alphabet $\Sigma$ of size $q \geq 2$ ($(q,r,k)$-MultSGKH for short) is to solve $r$ instances $I_1, \ldots, I_r$ of $q$-SGKH of length $k$ over an alphabet of size $q$. The input is $I = I_1 \circ \cdots \circ I_r$, where $I_i = (k, d_{(i,1)}, \ldots, d_{(i,k)})$ is a $q$-SGKH instance. The cost of a solution $\mathtt{A}(I)$ is the sum of the costs of the $r$ $q$-SGKH instances, i. e., $\operatorname{cost}(\mathtt{A}(I)) = \sum_{i=1}^{r} \operatorname{cost}(\mathtt{A}(I_i))$.*

The following lemma states that as many advice bits are necessary for correctly guessing $\alpha rk$ characters of a string of length $rk$ over an alphabet of size $q$ as for correctly guessing $\alpha rk$ characters of a $(q,r,k)$-MultSGKH instance.

**Lemma 6.** *Assume that there is an algorithm A solving $(q,r,k)$-MultSGKH with $b$ advice bits and making $\rho$ errors. Then there also is an algorithm B for $q$-SGKH on strings of length $rk$ using $b$ advice bits and making the same number of errors.*

*Proof.* Consider a $q$-SGKH instance of length $rk$. The first $k$ requests of this instance get mapped to the $k$ characters of the first string of a $(q,r,k)$-MultSGKH instance, and so on. Then, the decisions of an algorithm A for $(q,r,k)$-MultSGKH can directly be used by an algorithm B for $q$-SGKH on the corresponding positions. It is obvious that B makes an error if and only if A makes an error. □

**Theorem 6.** *For any $c \in \mathbb{R}^{\geq 1}$ and any $k \in \mathbb{N}^{>c-1}$, every online algorithm with advice for SetCover that is $c$-competitive needs to read at least*

$$\left(1 + \left(\frac{c-1}{k}\right) \log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right) \log_q\left(\frac{k-c+1}{k}\right)\right) \frac{k \cdot \log_2 q}{q^k} \cdot m \quad \text{or}$$

$$\left(1 + \left(\frac{c-1}{k}\right)\log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right)\log_q\left(\frac{k-c+1}{k}\right)\right)\frac{k\cdot(q-1)\cdot\log_2 q}{q^{k+1}-1}\cdot n$$

*advice bits, where $n = |X|$ and $m = |\mathcal{S}|$, for any $q \in \mathbb{N}^{\geq 2}$.*

*Proof.* We show how to transform any $(q, r, k)$-MultSGKH instance into a SetCover instance such that, if there is an algorithm $\mathtt{A}$ solving SetCover with $b$ advice bits and making $\rho$ errors, then there also is an algorithm $\mathtt{B}$ for $(q, r, k)$-MultSGKH using the same amount of advice and making the same number of errors.

Consider a set $\{s_1, \ldots, s_r\}$ of $r$ strings of length $k$ each over an alphabet of size $q$ and the corresponding instance $I = (I_{s_1}, \ldots, I_{s_r})$ of $(q, r, k)$-MultSGKH. We use the construction from the proof of Theorem 5 to construct $r$ SetCover instances $(X_j, \mathcal{S}_j)$ from $I_{s_j}$ such that the sets $X_j$ (and thus also the set families $\mathcal{S}_j$) are pairwise disjoint. For each $I_{s_j}$, we also construct a sequence $I'_{s_j}$ of requests for SetCover using the transformation from the proof of Theorem 5. Then we join these subinstances to get a SetCover instance $(X, \mathcal{S})$ by setting $X = \bigcup_{i=1}^{r} X_i$ and $\mathcal{S} = \bigcup_{i=1}^{r} \mathcal{S}_i$. The order of the requests in the SetCover instance follows an arbitrary order of the $I_{s_j}$, say $I_{s_1}, I_{s_2}, \ldots, I_{s_r}$.

The constructed SetCover instance has an optimal solution of size $r$ since every subinstance has a solution of size 1 and all subinstances are disjoint. The size of the ground set is $\frac{q^{k+1}-1}{q-1} \cdot r = n$ and the size of the set family is $q^k \cdot r = m$. We know from Theorem 5 that, for each algorithm for SetCover that reads $b$ advice bits and makes $\rho$ errors, there is an algorithm $\mathtt{B}$ for $q$-SGKH using the same advice and making the same number of errors. Consider an algorithm for the constructed SetCover instance. The algorithm $\mathtt{A}$ defines $r$ algorithms $\mathtt{A}_1, \ldots, \mathtt{A}_r$ for instances of length $k$. Each of these algorithms corresponds to one particular subinstance $(X_j, \mathcal{S}_j)$ of SetCover. Since these subinstances are disjoint, for any algorithm $\mathtt{A}_j$ that makes $\rho_j$ errors while using $b_j$ advice bits, there is an algorithm $\mathtt{B}_j$ using $b_j$ advice bits that makes the same number of errors for the string $s_j$ of the given instance of $(q, r, k)$-MultSGKH. Thus, in the sum, the number of errors $\mathtt{A}$ makes is the same as some algorithm $\mathtt{B}$ makes for the whole instance of $(q, r, k)$-MultSGKH. Next, employing Lemma 6, there is an algorithm $\mathtt{C}$ that makes $\rho$ errors while reading $b$ bits of advice for any instance of $q$-SGKH.

The competitive ratio $c$ that is reached by $\mathtt{A}$ is thus

$$c = \frac{r + (1-\alpha)\cdot rk}{r} = 1 + (1-\alpha)\cdot k,$$

hence $\alpha = 1 - \frac{c-1}{k}$.

Therefore, we can directly apply Theorem 2 for $\alpha = 1 - (c-1)/k$ yielding that at least

$$\left(1 + \left(\frac{c-1}{k}\right)\log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right)\log_q\left(\frac{k-c+1}{k}\right)\right)\cdot rk\log_2 q$$

advice bits are necessary to be $c$-competitive. To measure in $|\mathcal{S}| = m$ and $|X| = n$, we calculate $r = \frac{k}{q^k}\cdot m = \frac{k\cdot(q-1)}{q^{k+1}-1}\cdot n$, and finally get

$$\left(1 + \left(\frac{c-1}{k}\right)\log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right)\log_q\left(\frac{k-c+1}{k}\right)\right)\frac{k\cdot\log_2 q}{q^k}\cdot m \quad \text{or}$$

$$\left(1 + \left(\frac{c-1}{k}\right)\log_q\left(\frac{c-1}{k(q-1)}\right) + \left(\frac{k-c+1}{k}\right)\log_q\left(\frac{k-c+1}{k}\right)\right)\frac{k\cdot(q-1)\cdot\log_2 q}{q^{k+1}-1}\cdot n. \qquad \square$$

**Acknowledgments**

The authors would like to thank Richard Královič for enlightening discussions.

## References

1. N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.
2. P. Bianchi, H.-J. Böckenhauer, J. Hromkovič, and L. Keller. Online coloring of bipartite graphs with and without advice. In *Proc. of COCOON 2012*, LNCS 7434, pp. 519–530. Springer, 2012.
3. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
4. H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič. On the advice complexity of the $k$-server problem. In *Proc. of ICALP 2011*, LNCS 6755, pp. 207–218. Springer, 2011.
5. H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In *Proc. of ISAAC 2009*, LNCS 5878, pages 331–340. Springer, 2011.
6. H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. On the advice complexity of the knapsack problem. In *Proc. of LATIN 2012*, LNCS 7256, pp. 61–72. Springer, 2012.
7. G. Cohnen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes.* Elsevier, 1997.
8. M. Demange, X. Paradon, and V. Th. Paschos. On-Line Maximum-Order Induced Hereditary Subgraph Problems. In *Proc. of SOFSEM 2000*, LNCS 1963, pp. 327–335. Springer, 2000.
9. S. Dobrev, R. Královič, and D. Pardubská. How much information about the future is needed? In *Proc. of SOFSEM 2008*, LNCS 4910, pp. 247–258. Springer, 2008.
10. Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.
11. M. Forišek, L. Keller, and M. Steinová. Advice complexity of online coloring for paths. In *Proc. of LATA 2012*, LNCS 7183, pp. 228–239. Springer, 2012.
12. V. Guruswami, A. Rudra, and M. Sudan. *Essential Coding Theory.* Draft available at `http://www.cse.buffalo.edu/~atri/courses/coding-theory/book/`, 2012.
13. J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In *Proc. of MFCS 2010*, LNCS 6281, pages 24–36. Springer, 2010.
14. D. Komm and R. Královič. Advice complexity and barely random algorithms. *RAIRO ITA* 45(2):249–267, 2011.
15. D. Komm, R. Královič, and T. Mömke. On the advice complexity of the set cover problem. In *Proc. of CSR 2012*, LNCS 7353, pages 241–252. Springer, 2012.
16. F. J. MacWilliams, N. J. A. Sloane. *The Theory of Error-Correcting Codes*, Second Edition, North-Holland Publishing Company, 1978.
17. R. Moser, D. Scheder. A full derandomization of Schöning's $k$-SAT algorithm. In *Proc. of STOC 2011*, pp. 245–252. ACM, 2011.
18. M. Renault and A. Rosén. On online algorithms with advice for the $k$-server problem. In *Proc. of WAOA 2012*, *LNCS* 7164, pp. 198–210. Springer, 2012.
19. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.