RAIRO-Theor. Inf. Appl. 49 (2015) 139–152
DOI: 10.1051/ita/2015003

Available online at:
www.rairo-ita.org

TOWARDS USING THE HISTORY IN ONLINE COMPUTATION WITH ADVICE*

Sacha Krug[1]

Abstract. Recently, advice complexity has been introduced as a new framework to analyze online algorithms. There, an online algorithm has access to an infinite binary advice tape during the computation. The contents of this tape were prepared beforehand by an omniscient oracle. One is interested in analyzing the number of accessed advice bits necessary and/or sufficient to achieve a certain solution quality. Among others, the bit guessing problem was analyzed in this framework. Here, an algorithm needs to guess a binary string bit by bit, either with or without getting immediate feedback after each bit. The bit guessing problem can be used to obtain lower bounds on the advice complexity of a variety of other online problems. In this paper, we analyze the difference between the two bit guessing variants. More precisely, we show that getting feedback after each request helps save advice bits when we allow errors to be made. This is by no means obvious – for optimality, both problem versions need the same amount of advice, and without advice, knowing the history does not help at all.

Mathematics Subject Classification. 68Q01, 68W27.

1. Introduction

Ever since the inception of computer science, online computation has played an important role. Many real-life problems do not follow the classic input – computation – output pattern; instead, the input arrives piecewise, and an algorithm

Keywords and phrases. Online computation, bit guessing, advice complexity.

* *This work was partially supported by SNF Grant 200021-141089.*

[1] Department of Computer Science, ETH, 8092 Zürich, Switzerland.
sacha.krug@inf.ethz.ch

Article published by EDP Sciences

is required to provide a part of its final output in every time step, without the possibility to change it later on. Usually, the performance of such an *online algorithm* is measured by comparing its output with that of an optimal offline algorithm for the same problem. This is the so-called *competitive analysis* as introduced by Sleator and Tarjan [19]. For a general overview of the topic, we recommend the textbook by Borodin and El-Yaniv [8].

Unfortunately, simply comparing online and offline algorithms turns out to be too much of a black-and-white approach. An online algorithm knows *nothing* about future parts of the input, whereas an offline algorithm knows *everything*. What about an algorithm that has *partial* knowledge of the future? To model such an algorithm, the framework of *online computation with advice* has been introduced a few years ago [6, 10, 11]. Here, an online algorithm has access to an infinite binary *advice tape*, from which it can read information during the computation. The advice tape is prepared before the computation by an all-knowing oracle and may contain any information that is helpful to the algorithm during the computation. In particular, the contents of the advice tape may encode the entire input instance at hand. Often, however, the algorithm can achieve a certain output quality with much less information. Therefore, one is interested in obtaining upper and lower bounds on the *advice complexity* of online problems, *i.e.*, on the number of accessed *advice bits* necessary and sufficient to produce an optimal or near-optimal output. This framework has been successfully applied to various online problems, *e.g.*, k-server [5, 13, 17], knapsack [7], graph coloring [3, 12, 18], or disjoint path allocation [2].

Böckenhauer *et al.* [4] established the *bit guessing problem* as a generic tool to prove lower bounds on the advice complexity of online problems. The main goal in this problem is, unsurprisingly, to guess a binary string bit by bit. There are two variants of it: either an algorithm gets immediate feedback on whether its last output bit was correct (called *bit guessing with known history,* short *BGKH*), or it gets no feedback until the very end, when the entire string is revealed at once (called *bit guessing with unknown history,* short *BGUH*). The authors showed that the relatively straightforward lower bound for BGUH resulting from the sphere-covering bound also holds for BGKH, thereby establishing a common lower bound for both problems. They did not, however, show that getting immediate feedback actually helps in saving advice or, in other words, that the advice complexity of BGKH is lower than the one of BGUH.

The goal of this paper is to do just that, *i.e.*, to show that knowing the history indeed helps. This is by no means obvious. For instance, when there is no advice, knowing the history does not help at all. And even if there is advice, an optimal algorithm still needs to read the same amount of advice for both problems. But if we allow the algorithm to make errors, it can work together with the oracle to reduce the necessary number of advice bits (compared to BGUH).

This paper only considers algorithms that are allowed to make at most one error. We are, however, convinced that our approach can be generalized to a larger number of allowed errors.

2. Preliminaries and unknown history

The framework of online computation with advice is captured in the following definition.

Definition 2.1. Consider an input sequence $I = (x_1, x_2, \ldots, x_n)$. An *online algorithm* $\mathtt{A}$ *with advice* computes the output sequence $\mathtt{A}^{\phi}(I) = (y_1, y_2, \ldots, y_n)$ such that y_i is computed from $\phi, x_1, x_2, \ldots, x_i$, where ϕ is the content of the advice tape, *i.e.*, an infinite bit string. The algorithm $\mathtt{A}$ is *c-competitive with advice complexity* $s(n)$ if there is a non-negative constant α such that, for every $n \in \mathbb{N}$ and every input sequence I of length at most n, there is some ϕ such that $\mathrm{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \mathrm{cost}(\mathcal{O}pt(I)) + \alpha$ and at most the first $s(n)$ bits of ϕ have been accessed during the computation of $\mathtt{A}$ on I.

The two problems we are dealing with are the following ones.

Definition 2.2 (Böckenhauer *et al.* [4])**.** The *bit guessing problem with unknown history* (BGUH) is the following online minimization problem. The input $I = (n, ?, ?, \ldots, ?, d)$ consists of the *input length* n in the first request, $n-1$ subsequent requests "?" carrying no extra information, and the correct bit string $d = d_1 d_2 \ldots d_n \in \{0,1\}^n$. In each of the first n time steps, an online algorithm $\mathtt{A}$ has to output one bit, forming the output sequence $\mathtt{A}(I) = y_1 y_2 \ldots y_n \in \{0,1\}^n$. The algorithm does not have to produce an output in the last time step, when the string d is revealed. The cost of a solution $\mathtt{A}(I)$ is the Hamming distance $\mathrm{Ham}(d, \mathtt{A}(I))$ between d and $\mathtt{A}(I)$, *i.e.*, the number of positions at which d and $\mathtt{A}(I)$ differ.

Definition 2.3 (Böckenhauer *et al.* [4])**.** The *bit guessing problem with known history* (BGKH) is the following online minimization problem. The input $I = (n, d_1, d_2, \ldots, d_n)$ consists of the *input length* n and the bits $d_1, d_2, \ldots, d_n \in \{0,1\}$ that are revealed one by one. In each of the first n time steps, an online algorithm $\mathtt{A}$ has to output one bit, forming the output sequence $\mathtt{A}(I) = y_1 y_2 \ldots y_n \in \{0,1\}^n$. The algorithm does not have to produce an output in the last time step. The cost of a solution $\mathtt{A}(I)$ is again the Hamming distance between $d_1 d_2 \ldots d_n$ and $\mathtt{A}(I)$.

Moreover, we will sometimes refer to the following two concepts.

Definition 2.4. A *Hamming ball* of radius r around a bit string s of length n is the set of all bit strings of length n with Hamming distance at most r from s. More formally, the Hamming ball of any $s \in \{0,1\}^n$ is $\{u \in \{0,1\}^n | \, \mathrm{Ham}(u,s) \leq r\}$.

Definition 2.5. A *covering code* of radius r and dimension n is a subset of bit strings of length n such that the union of the Hamming balls of radius r around these strings contain all bit strings of length n. More formally, it is any set S such that $\bigcup_{s \in S} \{u \in \{0,1\}^n | \, \mathrm{Ham}(u,s) \leq r\} = \{0,1\}^n$.

Let us first briefly consider BGUH. Clearly, for optimality, n advice bits are necessary and sufficient. We consider the case where one error, *i.e.*, one wrongly guessed bit, is allowed. e know that an online algorithm reading 2^b advice bits is equivalent to a set of 2^b deterministic online algorithms [15]. With BGUH, a deterministic online algorithm receives only n as input. Thus, for fixed n, it outputs one fixed bit string while the computation is in progress, independent of the actual bit string $d_1 d_2 \dots d_n$. Every such fixed output bit string, which we call *center string* from now on, corresponds to an advice string. We want to minimize the number of center strings such that the algorithm never outputs more than one wrong bit. From Definition 2.4, we can see that a Hamming ball of radius 1 around some center string s has size $\sum_{i=0}^{1} \binom{n}{i}$. To construct a covering code of radius 1 and dimension n, we therefore need at least $\left\lceil 2^n / (\sum_{i=0}^{1} \binom{n}{i}) \right\rceil$ center strings in total. This lower bound is called the *sphere-covering bound.* It is not known to be tight in general. Various upper bounds have been established over the years for specific values of n; MacWilliams and Sloane [16] provide a good overview.

3. Known history

Now we consider BGKH. As already mentioned, Böckenhauer *et al.* [4] showed that the sphere-covering bound is also a lower bound on the advice complexity in this case. The goal of this section is to establish an upper bound that is at most one bit higher than the sphere-covering bound if the algorithm is allowed to make at most one error. First, however, we improve the lower bound by one bit for infinitely many n. For these n, the upper bound is thus tight.

3.1. Lower bound

We establish a lower bound of $z_n := 2 \left\lceil 2^{n-1}/(n+1) \right\rceil$ for infinitely many $n \geq 3$. Note that z_n is the smallest even number greater than or equal to $\lceil 2^n/(n+1) \rceil$, the sphere-covering bound for $r = 1$. This holds because z_n is even and we have $\lceil 2^n/(n+1) \rceil \leq z_n \leq \lceil 2^n/(n+1) \rceil + 1$, which follows from the simple fact that $\lceil 2x \rceil \leq 2 \lceil x \rceil \leq \lceil 2x \rceil + 1$.

The goal of this section is to prove that every algorithm for BGKH that outputs at most one wrong bit needs at least z_n advice strings on inputs of length n, for infinitely many $n \geq 3$ with odd $\lceil 2^n/(n+1) \rceil$ (if $\lceil 2^n/(n+1) \rceil$ is an even number, it equals z_n and the result trivially holds). That is, our goal is to show the following.

Theorem 3.1. *Every algorithm for BGKH that guesses at most one bit wrong needs at least z_n advice strings on inputs of length n, for infinitely many $n \geq 3$ with odd $\lceil 2^n/(n+1) \rceil$.*

We assume $\lceil 2^n/(n+1) \rceil$ to be odd for the rest of this section.

Assume that there is an algorithm that outputs at most one wrong bit using $\lceil 2^n/(n+1) \rceil$ advice strings. We know from Lemma 2 in [4] that every advice string

corresponds to a set of $n+1$ strings (actually, the lemma talks about sets of *at most* $n+1$ strings, but we can always fill smaller sets with dummy strings). We observe the following simple fact about these sets.

Lemma 3.2. *There are only two types of sets: The family $\mathcal{X}$ of sets that contain one string starting with* 0 *and n strings starting with* 1*, and the family $\mathcal{Y}$ of sets satisfying the* vice versa *condition.*

Proof. We prove that (i) each set corresponding to an advice string contains at most one string starting with 0 or at most one string starting with 1 and (ii) each set corresponding to an advice string contains at least one string starting with 0 and at least one string starting with 1.

For (i), assume for contradiction that some set contains two strings that start with 0 and two that start with 1. Then, the adversary can enforce an error at the first position. After this, at least two possible instances I and I' are left, but no algorithm can distinguish them, as both the advice and the history are identical until I and I' differ. Thus, any algorithm makes at least two errors, and we have a contradiction.

For (ii), assume for contradiction that some set S contains $n+1$ strings that all start with the same bit. Consider the situation that there is some non-empty bit string α such that two strings in S start with $\alpha 0$ and two other strings with $\alpha 1$. Then, the adversary can enforce an error directly after α. As in the previous proof, at least two possible instances I and I' are left, but no algorithm can distinguish them, as both the advice and the history are identical until I and I' differ. Thus, any algorithm makes at least two errors, and we have a contradiction.

Hence, we only need to show that, if some set contains $n+1$ strings that all start with the same bit b, then there is such an α. Assume for contradiction that there is no such α. Then, at most one or at least n strings must have 1 at the second position, as otherwise two strings start with $b0$ and two strings start with $b1$, *i.e.*, $\alpha = b$. Without loss of generality, at most one string has a 1 at the second position, *i.e.*, at least n strings start with $b0$. Among these n strings, at most one has a 1 at the third position, *i.e.*, at least $n-1$ strings start with $b00$. If we apply this argument inductively, we can show that at least three strings start with a common prefix of length $n-1$, *i.e.*, two of them are identical. This is, however, not possible, since all $n+1$ strings in S are different. □

Since $n \geq 3$, every set contains at least four strings, so $\mathcal{X}$ and $\mathcal{Y}$ are disjoint. Let $x := |\mathcal{X}|$ and $y := |\mathcal{Y}|$. Hence, $x + y = \lceil 2^n/(n+1) \rceil$. Since this term is odd, $x > y$ without loss of generality. More precisely, we assume $x = y + 1$, *i.e.*, $x = (\lceil 2^n/(n+1) \rceil + 1)/2$ (the argument generalizes to greater x, because with smaller y, the number of covered strings that start with 0 decreases). Exactly $x + ny$ covered strings start with 0. Clearly, *all* strings starting with 0 have to be covered, *i.e.*,

$$x + ny \geq 2^{n-1} \tag{3.1}$$

must hold. Our goal now is to show that

$$x + ny < 2^{n-1}, \quad \textit{i.e.},$$
$$\frac{\left\lceil \frac{2^n}{n+1} \right\rceil + 1}{2} + n\frac{\left\lceil \frac{2^n}{n+1} \right\rceil - 1}{2} = \frac{n+1}{2}\left\lceil \frac{2^n}{n+1} \right\rceil - \frac{n-1}{2} < 2^{n-1} \tag{3.2}$$

holds for infinitely many n with odd $\lceil 2^n/(n+1) \rceil$, since this immediately implies that not all strings can be covered and thus z_n advice strings are necessary.

Since $\lceil 2^n/(n+1) \rceil$ is odd, we know that $2^n/(n+1) \notin \mathbb{N}$ and thus $\lceil 2^n/(n+1) \rceil = \lfloor 2^n/(n+1) \rfloor + 1$. We obtain

$$\begin{aligned}\frac{n+1}{2}\left\lceil \frac{2^n}{n+1} \right\rceil - \frac{n-1}{2} &= \frac{n+1}{2}\left(\left\lfloor \frac{2^n}{n+1} \right\rfloor + 1\right) - \frac{n-1}{2} \\ &= \frac{n+1}{2}\left\lfloor \frac{2^n}{n+1} \right\rfloor + 1 \\ &< 2^{n-1} + 1.\end{aligned}$$

If $\lfloor 2^n/(n+1) \rfloor$ is divisible by 4, then the term $(n+1)/2 \cdot \lfloor 2^n/(n+1) \rfloor + 1$ is odd and (3.2) indeed holds, because every odd integer smaller than $2^{n-1}+1$ is also smaller than 2^{n-1}. From now on, we thus assume that $\lfloor 2^n/(n+1) \rfloor = 4k+2$, for some $k \in \mathbb{N}$.

We already know that

$$\frac{n+1}{2}\left\lfloor \frac{2^n}{n+1} \right\rfloor + 1 < 2^{n-1} + 1.$$

Because $\lfloor 2^n/(n+1) \rfloor$ is even, the left side is an integer, and to prove (3.2), we only need to show that

$$\frac{n+1}{2}\left\lfloor \frac{2^n}{n+1} \right\rfloor + 1 \neq 2^{n-1}.$$

Assume that the two terms are equal, *i.e.*,

$$(n+1)\frac{\left\lfloor \frac{2^n}{n+1} \right\rfloor}{2} = (n+1)(2k+1) = 2^{n-1} - 1.$$

Note that the right-hand side is the $(n-1)$th Mersenne number M_{n-1}. If, for some $n \in \mathbb{N}$, $n+1$ is no divisor of M_{n-1}, we have obtained a contradiction and thus shown that our claimed lower bound z_n holds for this n. One easily sees that this is true for all $n = 3 \times 2^{2m} - 1$, where $m \in \mathbb{N}$, since $n+1 = 3 \cdot 2^{2m}$ does not divide $2^{n-1} - 1$, simply because the former is even and the latter odd.

It remains to show that $\lceil 2^n/(n+1) \rceil$ is an odd number for all these n. To see this, however, we simply observe that

$$\left\lceil \frac{2^n}{n+1} \right\rceil = \left\lceil \frac{2^{3\times 2^{2m}-1}}{3 \times 2^{2m}} \right\rceil = \left\lceil \frac{2^{3\times 2^{2m}-2m-1}}{3} \right\rceil$$

is of the form $\lceil 2^{2l+1}/3 \rceil$, and all these terms are odd ([1], pp. 315–316).

Thus, we have improved the lower bound on the number of necessary advice bits to guarantee an output with at most one error to z_n, for infinitely many n with odd $\lceil 2^n/(n+1) \rceil$. This concludes the proof of Theorem 3.1. □

3.2. Upper bound

Now we show that the lower bound established in the previous section is tight.

Theorem 3.3. *There is an algorithm* A *that solves BGKH on bit strings of length n using at most z_n advice strings and guessing at most one bit wrong.*

We first explain the overall idea of the proof. In a nutshell, we now show how to systematically construct a covering code for the binary hypercube of dimension n. As we have already seen, this approach also works for BGUH. The crucial observation, however, is that with BGKH, we have much more freedom in constructing the set, since the algorithm can, at any time step, incorporate the feedback received so far. More precisely, we do not have the requirement that all elements of a Hamming ball around some string s have Hamming distance at most 1 from s, but rather that, once A has output a wrong bit, the remaining bit string is uniquely determined (and, therefore, A outputs no more wrong bits). We can use this increased freedom to our advantage by constructing a covering code that has, on average, less "intersections" than one for BGUH, meaning that the average number of Hamming balls per bit string is lower. In a sense, less of the Hamming balls is "wasted".

Let T be the complete binary tree with 2^n leaves. We fix an order on the leaves such that they represent, from left to right, the bit strings of length n in lexicographical order. Due to this one-to-one correspondence between the leaves of T and the bit strings of length n, we use the terms leaf and string interchangeably. We can even generalize this one-to-one correspondence to *all* vertices of T as follows. Any inner vertex v of T corresponds to the common prefix of all leaves in the subtree rooted at v. Therefore, any vertex at depth d corresponds to a bit string of length d, for $0 \leq d \leq n$.

Note that any possible output of any algorithm A for bit guessing (with known or unknown history) on any input of length n is some leaf of T, *i.e.*, the set of all outputs of A is a subset of the leaves of T.

With BGUH, once a center string s is fixed, all other n strings in the Hamming ball are also fixed; they are all the strings with Hamming distance 1 to s. That is, as soon as we pick some leaf s in T as a center string, the other n covered leaves are fixed.

With BGKH, however, we have more freedom. For some fixed center string s, we can construct a set S as follows. Consider the path P from s to the root R of T. For every vertex $v \neq s$ in P, select *some* string from the subtree $T_{\overline{s},v}$ of v not containing s (the possibility to choose such a string in $T_{\overline{s},v}$ is crucial. With BGUH, it is the unique string in $T_{\overline{s},v}$ that has Hamming distance 1 to s).

On any fixed input instance $(n, d_1, \dots, d_n)$, A now proceeds as follows. First, it reads advice to determine a center string $s = s_1 s_2 \dots s_n$ (we will later compute how much advice is necessary for this). Then, A outputs the bits of s one by one. In T, this corresponds to traversing the path from R to s, where each visited vertex corresponds to the string that A has output so far. Let t be the time step in which A makes the first mistake. In other words, $s_1 s_2 \dots s_{t-1} = d_1 d_2 \dots d_{t-1}$, but $s_t \neq d_t$. Considering again T, this implies that $d_1 d_2 \dots d_n$ lies in $T_{\overline{s},v}$, for the vertex v corresponding to $s_1 s_2 \dots s_{t-1}$. By construction, however, we know that S contains exactly one string s' in $T_{\overline{s},v}$. Thus, A simply outputs the corresponding bits of s' from time step t on. This results in at most one error in total, since $s_1 s_2 \dots s_{t-1} = d_1 d_2 \dots d_{t-1}$, $s_t \neq d_t$, and $s'_{t+1} s'_{t+2} \dots s'_n = d_{t+1} d_{t+2} \dots d_n$.

All that is left to explain is how to select the center strings (and how many) and how to construct the corresponding sets.

3.2.1. Selection of the center strings

We want to analyze the number of center strings necessary to cover all leaves of T. To this end, we define a function $V\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $V(n,h)$ tells us how many strings can be covered with h center strings in a tree of depth n with 2^n leaves.

If there is no center string, then no leaf can be covered. Hence,

$$V(n,0) = 0, \quad \text{for } n \in \mathbb{N}.$$

If there is one leaf and at least one center string, then the leaf can be covered. Hence,

$$V(0,h) = 1, \quad \text{for } h \geq 1.$$

Let now T_l and T_r denote the left and the right subtree of R, respectively. For any number h of center strings, we can choose them arbitrarily among all the leaves of T, *i.e.*, $h-i$ center strings in T_l and i center strings in T_r, for some i with $0 \leq i \leq h$. Then $V(n-1, h-i)$ leaves in T_l and $V(n-1,i)$ leaves in T_r are covered. Additionally, each center string in T_l allows us to cover an additional leaf in T_r and vice versa. This follows from the construction above: For any center string s in T_l, we can add an arbitrary vertex from $T_{\overline{s},v}$ to S, for every vertex v on the path from s to R. In particular, we can add an arbitrary vertex from $T_{\overline{s},R} = T_r$ to S.

This is clearly only possible if there are still uncovered leaves in T_r. Otherwise, all 2^{n-1} leaves in T_r are already covered. Thus, $\min\{V(n-1, h-i) + i, 2^{n-1}\}$ vertices in T_l and $\min\{V(n-1,i) + h - i, 2^{n-1}\}$ vertices in T_r are covered in total. Hence,

$$\begin{aligned} V(n,h) = \max_{0 \leq i \leq h} \big\{ & \min\big\{V(n-1, h-i) + i, 2^{n-1}\big\} \\ & + \min\big\{V(n-1,i) + h - i, 2^{n-1}\big\}\big\}. \end{aligned} \tag{3.3}$$

Lemma 3.4. *For every $n \in \mathbb{N}$, we have $V(n, z_n) = 2^n$.*

TABLE 1. Values of $V(n,h)$ for $0 \le n \le 7$, $0 \le h \le 15$.

n \ h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	0	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4
3	0	4	8	8	8	8	8	8	8	8	8	8	8	8	8	8
4	0	5	10	14	16	16	16	16	16	16	16	16	16	16	16	16
5	0	6	12	18	24	29	32	32	32	32	32	32	32	32	32	32
6	0	7	14	21	28	35	42	49	56	61	64	64	64	64	64	64
7	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120

Proof. Trivially, $V(n, z_n) \le 2^n$. For the other direction, we first show that, for every $n \in \mathbb{N}$, we have

$$V\left(n, \frac{z_{n+1}}{2}\right) \ge 2^n - \frac{z_{n+1}}{2}. \tag{3.4}$$

Together with the observation that

$$\begin{aligned} V(n,h) \ge \min &\left\{ V\left(n-1, \left\lceil \frac{h}{2} \right\rceil\right) + \left\lfloor \frac{h}{2} \right\rfloor, 2^{n-1} \right\} \\ &+ \min \left\{ V\left(n-1, \left\lfloor \frac{h}{2} \right\rfloor\right) + \left\lceil \frac{h}{2} \right\rceil, 2^{n-1} \right\}, \end{aligned} \tag{3.5}$$

this immediately yields the lemma for $h := z_n$.

Table 1 shows that (3.4) holds for $n \le 6$. For $n \ge 7$, we show by induction that

$$V(n,h) \ge (n+1)h, \tag{3.6}$$

for every $h \le z_{n+1}/2$. This immediately implies

$$\begin{aligned} V\left(n, \frac{z_{n+1}}{2}\right) &\ge (n+1)\frac{z_{n+1}}{2} = (n+2)\frac{z_{n+1}}{2} - \frac{z_{n+1}}{2} \\ &= (n+2)\left\lceil \frac{2^n}{n+2} \right\rceil - \frac{z_{n+1}}{2} \ge 2^n - \frac{z_{n+1}}{2}. \end{aligned}$$

The induction base $n = 7$ follows from (1). To prove (3.6) for $n \ge 8$, we use (3.5). Observe that

$$\left\lfloor \frac{h}{2} \right\rfloor \le \left\lceil \frac{h}{2} \right\rceil \le \left\lceil \frac{z_{n+1}}{4} \right\rceil = \left\lceil \frac{\left\lceil \frac{2^n}{n+2} \right\rceil}{2} \right\rceil = \left\lceil \frac{2^{n-1}}{n+2} \right\rceil \le \left\lceil \frac{2^{n-1}}{n+1} \right\rceil = \frac{z_n}{2},$$

where the second equality holds because $\lceil \lceil x \rceil / 2 \rceil = \lceil x/2 \rceil$, for every $x \in \mathbb{R}^+$. Remember moreover that the induction hypothesis tells us that $V(n-1, h') \ge nh'$, for every $h' \le z_n/2$. Therefore, we can replace the respective terms in (3.5) and obtain

$$V(n,h) \ge \min \left\{ n\left\lceil \frac{h}{2} \right\rceil + \left\lfloor \frac{h}{2} \right\rfloor, 2^{n-1} \right\} + \min \left\{ n\left\lfloor \frac{h}{2} \right\rfloor + \left\lceil \frac{h}{2} \right\rceil, 2^{n-1} \right\}. \tag{3.7}$$

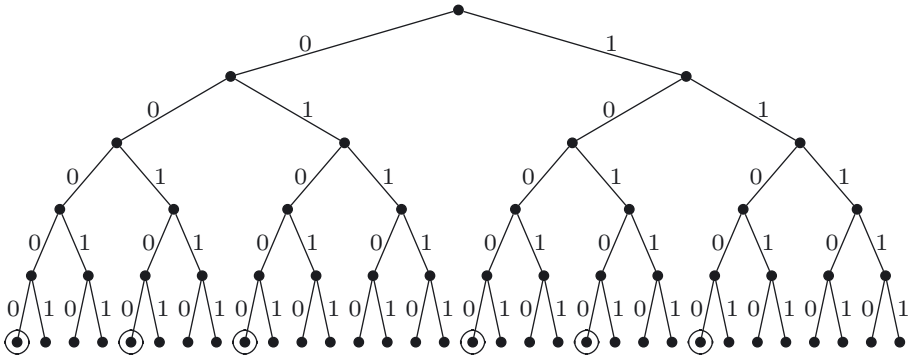

FIGURE 1. The selected center strings for $n = 5$.

Note that

$$\begin{aligned} n\left\lfloor\frac{h}{2}\right\rfloor + \left\lceil\frac{h}{2}\right\rceil &\le n\left\lceil\frac{h}{2}\right\rceil + \left\lfloor\frac{h}{2}\right\rfloor \le n\left\lceil\frac{z_{n+1}}{4}\right\rceil + \left\lfloor\frac{z_{n+1}}{4}\right\rfloor \\ &\le n\left\lceil\frac{\left\lceil\frac{2^n}{n+2}\right\rceil}{2}\right\rceil + \frac{\left\lceil\frac{2^n}{n+2}\right\rceil}{2} \le (n+1)\left\lceil\frac{2^{n-1}}{n+2}\right\rceil \\ &\le \frac{n+1}{n+2}\cdot 2^{n-1} + n + 1 = 2^{n-1} - \frac{2^{n-1}}{n+2} + n + 1 \\ &\le 2^{n-1}, \end{aligned}$$

where the last inequality holds because $2^{n-1}/(n+2) \ge n+1$ for $n \ge 8$. Thus, we can replace the respective first terms in (3.7) and obtain

$$V(n,h) \ge n\left\lceil\frac{h}{2}\right\rceil + \left\lfloor\frac{h}{2}\right\rfloor + n\left\lfloor\frac{h}{2}\right\rfloor + \left\lceil\frac{h}{2}\right\rceil = (n+1)h. \qquad \square$$

We gained two important insights from this lemma and its proof. First, z_n center strings are sufficient to cover all strings in $\{0,1\}^n$. Second, an optimal way to select them among the leaves of T_l and T_r is to do so "as balanced as possible".

We now show how to do this. Let $\mathrm{Bin}_n(i)$ denote the binary representation of a natural number i using n bits, and $\mathrm{Bin}_n(i)^R$ its reverse. We choose as center strings the set $\{\mathrm{Bin}_n(i)^R | 0 \le i < z_n\}$. One can easily see that exactly every second leaf is in the left subtree (and this holds recursively inside T_l and T_r). For instance, for $n = 5$, we choose the set $\{\mathrm{Bin}_5(i)^R | 0 \le i < 6\} = \{00000, 10000, 01000, 11000, 00100, 10100\}$ (see Fig. 1).

Lemma 3.5. *The set of center strings above is selected such that, for every inner vertex v in T, the number of center strings in the left and in the right subtree of v differ at most by* 1.

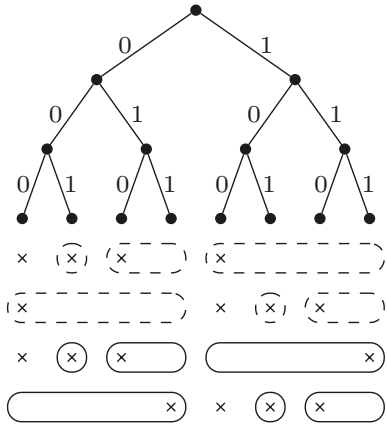

FIGURE 2. A failing (dashed) and a successful allocation strategy. The rounded rectangles indicate the clusters, and the crosses show which leaf was selected from this cluster (the crosses without rectangles indicate the center strings).

Proof. We can identify every subtree of T at depth d, for $0 \leq d \leq n$, with the prefix of length d common to all strings in this subtree. Thus, we only need to show that every prefix occurs at most once more than any other of the same length in the selected center strings. Equivalently, we need to show that every suffix occurs at most once more than any other of the same length in $\{\mathrm{Bin}_n(i) | 0 \leq i < z_n\}$. This follows, however, immediately from the fact that all suffixes of some fixed length occur periodically when enumerating integers. □

3.2.2. Construction of the hamming sets

Given a center string s, we now want to investigate the best way to construct a set S from it. Remember that we do this by going up the path from s to R and select, for every vertex $v \neq s$, some string in $T_{\overline{s},v}$. The question is: Which vertex should we choose from $T_{\overline{s},v}$?

Clearly, if there is only one center string, this does not matter. However, as soon as there are several center strings, we should avoid covering a leaf with two sets, as we then potentially waste one position. For instance, for $n = 3$, if the two center strings are 000 and 100, an algorithm that always selects the leftmost vertex in every subtree $T_{\overline{s},v}$ fails, because it ends up with the two sets $\{000, 001, 010, 100\}$ and $\{000, 100, 101, 110\}$, which do not cover all strings, as opposed to the sets $\{000, 001, 010, 111\}$ and $\{011, 100, 101, 110\}$ (see Fig. 2).

Nevertheless, a simple greedy strategy works. For some fixed center string s, let $C_{s,i}$ denote the set of all bit strings in the unique tree $T_{\overline{s},v}$ of size 2^i, for $0 \leq i < n$.

We call the sets $C_{s,i}$ *clusters*. For every $0 \leq i < n$, the set corresponding to s thus contains one vertex from cluster $C_{s,i}$.

The algorithm constructs all sets in parallel as follows. In time step i, it processes all clusters of size 2^i and selects in each cluster the leftmost leaf, *i.e.*, the lexicographically first bit string, that has not yet been covered.

Lemma 3.6. *The greedy algorithm described above implements a surjective mapping from the clusters to the leaves.*

Proof. If the number of clusters and center strings together is greater than the number of leaves in the tree, then any surjective mapping covers at least one string several times. In what follows, however, we assume, without loss of generality, that every string is covered exactly once.

We show how to transform any surjective mapping into the greedy mapping that the algorithm described above ends up with using only surjectivity-preserving transformations. For the mapping to be greedy, observe that one of the following two statements has to hold for every cluster C of size c:

- Cluster C is not assigned a string at all. Then, all its strings are assigned to clusters of size at most c.
- Cluster C is assigned a string s. Then, all vertices in C left of s are assigned to clusters of size at most c.

If, for some cluster C, neither of these two statements holds, the mapping is not greedy. We show how to deal with such clusters.

(1) A cluster C is assigned no string, but some string s in C is assigned to a larger cluster C'. Then, we can just re-assign s from C' to C, *i.e.*, C' is no longer assigned a string. We can do this iteratively for increasing cluster sizes such that, in the end, no cluster is "blocked" by some larger cluster.
(2) A cluster C is assigned some string s, but another string s' in C that is lexicographically before s is assigned to a larger cluster C'. Then we just "flip" the two strings, *i.e.*, assign s' to C and s to C' (if s can be flipped with several strings, pick the leftmost one, *i.e.*, the lexicographically first one). As above, we can do this iteratively for increasing cluster sizes such that, in the end, every cluster is assigned a string as far to the left as necessary. □

4. Conclusion

We have established an upper bound of $z_n = 2\left\lceil 2^{n-1}/(n+1)\right\rceil$ on the advice complexity of BGKH on instances of length n when one error is allowed. For all n with even $\lceil 2^n/(n+1)\rceil$, this upper bound is tight. We showed that it is also tight for infinitely many n with odd $\lceil 2^n/(n+1)\rceil$. For all other n, the gap between the upper and the lower bound is at most one bit. From Table 2, one can see that, already for small values, z_n is almost always smaller than the lower bound

TABLE 2. Lower bounds on the advice complexity of BGUH and upper bounds z_n on the advice complexity of BGKH with one allowed error, for $1 \leq n \leq 20$. The numbers for BGUH are taken from Cohen *et al.* [9] and Kéri [14].

n	1	2	3	4	5	6	7	8	9	10
LB BGUH	1	2	2	4	7	12	16	32	62	107
UB BGKH	2	2	2	4	6	10	16	30	52	94
n	11	12	13	14	15	16	17	18	19	20
LB BGUH	180	342	598	1172	2048	4096	7419	14 564	26 309	52 618
UB BGKH	172	316	586	1094	2048	3856	7282	13 798	26 216	49 934

for unknown history. Indeed, for infinitely many n, the tight bound for BGKH is lower than the lower bound for BGUH ([9], Sect. 6.6 and 6.8).

REFERENCES

[1] J. Arndt, Matters Compuational: Ideas, Algorithms, Source Code. Springer (2011).

[2] K. Barhum, H.-J. Böckenhauer, M. Forišek, H. Gebauer, J. Hromkovič, S. Krug, J. Smula and B. Steffen, On the power of advice and randomization for the disjoint path allocation problem. In *Proc. of SOFSEM*. Vol. 8327 of *Lect. Note Comput. Sci.* Springer (2014) 89–101.

[3] M.P. Bianchi, H.-J. Böckenhauer, J. Hromkovič and L. Keller, Online coloring of bipartite graphs with and without advice. In *Proc. of COCOON*. Vol. 7434 of *Lect. Note Comput. Sci.* Springer (2012) 519–530.

[4] H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula and A. Sprock, The string guessing problem as a method to prove lower bounds on the advice complexity. In *Proc. of COCOON*. Vol. 7936 of *Lect. Note Comput. Sci.* Springer (2013) 493–504.

[5] H.-J. Böckenhauer, D. Komm, R. Královič and R. Královič, On the advice complexity of the k-server problem. In *Proc. of ICALP*. Vol. 6755 of *Lect. Note Comput. Sci.* Springer (2011) 207–218.

[6] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič and T. Mömke, On the advice complexity of online problems. In *Proc. of ISAAC*. Vol. 5878 of *Lect. Note Comput. Sci.* Springer (2009) 331–340.

[7] H.-J. Böckenhauer, D. Komm, R. Královič and P. Rossmanith, On the advice complexity of the knapsack problem. In *Proc. of LATIN*. Vol. 7256 of *Lect. Note Comput. Sci.* Springer (2012) 61–72.

[8] A. Borodin and R. El-Yaniv, Online Computation and Competitive Analysis. Cambridge University Press (1998).

[9] G. Cohen, I. Honkala, S. Litsyn and A. Lobstein, Covering Codes. Elsevier Science Publishers Ltd. (1997).

[10] S. Dobrev, R. Královič and D. Pardubská, How much information about the future is needed? In *Proc. of SOFSEM*. Vol. 4910 of *Lect. Note Comput. Sci.* Springer (2008) 247–258.

[11] Y. Emek, P. Fraigniaud, A. Korman and A. Rosén, Online computation with advice. *Theor. Comput. Sci.* **412** (2011) 2642–2656.

[12] M. Forišek, L. Keller and M. Steinová, Advice complexity of online coloring for paths. In *Proc. of LATA* (2012) 228–239.

[13] S. Gupta, S. Kamali and A. López-Ortiz, On advice complexity of the k-server problem under sparse metrics. In *Proc. of SIROCCO*. Vol. 8179 of *Lect. Note Comput. Sci.* Springer (2013) 55–67.

[14] G. Kéri, Tables for Bounds on Covering Codes. http://www.sztaki.hu/~keri/codes/. Accessed: 2014-05-30.
[15] D. Komm, R. Královič and T. Mömke, On the advice complexity of the set cover problem. In *Proc. of CSR*. Vol. 7353 of *Lect. Note Comput. Sci.* Springer (2012) 241–252.
[16] F.J. MacWilliams and N.J.A. Sloane, The Theory of Error-Correcting Codes, 2nd edition. North-Holland (1978).
[17] M.P. Renault and A. Rosén, On online algorithms with advice for the k-server problem. In *Proc. of WAOA* (2011) 198–210.
[18] S. Seibert, A. Sprock and W. Unger, Advice complexity of the online coloring problem. In *Proc. of CIAC*. Vol. 7878 of *Lect. Note Comput. Sci.* Springer (2013) 345–357.
[19] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules. *Commun. ACM* **28** (1985) 202–208.

Communicated by B. Doerr.
Received September 30, 2014. Accepted March 12, 2015.