# The online knapsack problem: Advice and randomization

# The Online Knapsack Problem: Advice and Randomization[☆,☆☆]

Hans-Joachim Böckenhauer[a], Dennis Komm[a], Richard Královič[a], Peter Rossmanith[b]

[a]*Department of Computer Science, ETH Zurich, Switzerland*
[b]*Department of Computer Science, RWTH Aachen, Germany*

**Abstract**

We study the advice complexity and the random bit complexity of the online knapsack problem. Given a knapsack of unit capacity, and $n$ items that arrive in successive time steps, an online algorithm has to decide for every item whether it gets packed into the knapsack or not. The goal is to maximize the value of the items in the knapsack without exceeding its capacity. In the model of advice complexity of online problems, one asks how many bits of advice about the unknown parts of the input are both necessary and sufficient to achieve a specific competitive ratio. It is well-known that even the unweighted online knapsack problem does not admit any competitive deterministic online algorithm.

For this problem, we show that a single bit of advice helps a deterministic online algorithm to become 2-competitive, but that $\Omega(\log n)$ advice bits are necessary to further improve the deterministic competitive ratio. This is the first time that such a phase transition for the number of advice bits has been observed for any problem. Additionally, we show that, surprisingly, instead of an advice bit, a single random bit allows for a competitive ratio of 2, and any further amount of randomness does not improve this. Moreover, we prove that, in a resource augmentation model, i. e., when allowing the online algorithm to overpack the knapsack by some small amount, a constant number of advice bits suffices to achieve a near-optimal competitive ratio.

We also study the weighted version of the problem proving that, with $\mathcal{O}(\log n)$ bits of advice, we can get arbitrarily close to an optimal solution and, using asymptotically fewer bits, we are not competitive. Furthermore, we show that an arbitrary number of random bits does not permit a constant competitive ratio.

*Keywords:* Online computation, advice complexity, randomization, resource augmentation, knapsack problem

## 1. Introduction

Online problems are an important class of computing problems where the input is not known to the algorithm in advance, but is revealed stepwise, and where, in each step, a piece of output has to be produced irrevocably. The standard way to analyze the quality of an online algorithm is via the so-called *competitive analysis*. Here, the quality of the solution as produced by the online algorithm is compared to the quality of an offline algorithm that knows the complete input in advance. An introduction to the theory and applications of competitive analysis is given in [1, 5, 12].

Comparing an algorithm having no knowledge about the forthcoming parts of the input with an algorithm having full knowledge of the future might only give a rough estimate of the real quality of an algorithm facing an online situation. To enable a more fine-grained analysis of the hardness of online problems, the *advice complexity* of online problems has been introduced recently [3, 7, 10, 18]. The idea behind this concept is to measure the amount of information about the forthcoming parts

[☆]This work was partially supported by ETH grant TH 18 07-3 and SNF grants 200020-120073 and 200021-141089.
[☆☆]An extended abstract of this paper was presented at LATIN 2012 [4].

*Email addresses:* hjb@inf.ethz.ch (Hans-Joachim Böckenhauer), dennis.komm@inf.ethz.ch (Dennis Komm), richard.kralovic@inf.ethz.ch (Richard Královič), rossmani@cs.rwth-aachen.de (Peter Rossmanith)

of the input an online algorithm needs to be optimal or to achieve a certain competitive ratio. More precisely, in this model, the online algorithm has access to some tape containing advice bits produced by an oracle knowing the complete input, and its advice complexity is the number of bits it reads from this advice tape, i. e., the amount of information about the yet unknown input parts it needs to know for its computation.

In this paper, we deal with an online version of the well-known knapsack problem. Here, an input consists of a set of items with specified weights and values, and a knapsack capacity. The goal is to choose a set of items with maximum value such that the total sum of their weights does not exceed the knapsack's capacity. The knapsack problem is a very well-studied hard optimization problem, for an introduction, see [17, 25]. In the online version of the knapsack problem, the items arrive one by one and the algorithm has to decide for each item whether it will pack it into the knapsack or not. These decisions must not be withdrawn at a later stage, i. e., no items can be removed from the knapsack. It is easy to see that no deterministic online algorithm can achieve any bounded competitive ratio [28]. Thus, the existing literature on the online knapsack problem mainly considers restricted variants of the problem [14, 20, 34] or an average-case analysis of randomized algorithms [28].

Our results are as follows. For the unweighted version of the problem, we prove that, with a single advice bit, a competitive ratio of 2 is achievable. Moreover, for an instance of $n$ items, any number $b$, $2 < b < \log(n-1)$, of advice bits cannot improve the competitive ratio. However, for every constant $\varepsilon > 0$, a competitive ratio of $1+\varepsilon$ is achievable using $\mathcal{O}(\log n)$ advice bits. For computing an optimal solution, a linear number of advice bits is necessary. At first glance, these results fit well into the picture as given by the advice complexity results for other problems like paging, job shop scheduling, or disjoint path allocation [3]: Linear advice is needed for optimality, logarithmic advice for beating the best randomized algorithm, and very few bits suffice to beat a deterministic algorithm. But having a closer look, one sees that the situation is pretty much different for the knapsack problem compared to the other above-mentioned problems: This problem is the first one for which a sharp phase transition in the number of advice bits can be shown in the following sense. Even $\log(n-2)$ advice bits are exactly as helpful as one bit, but $\mathcal{O}(\log n)$ bits already allow for an almost optimal solution.

A second line of research in this paper considers the random bit complexity of randomized online algorithms (without advice) for the knapsack problem. For the unweighted case, it turns out that, surprisingly, a single random bit is as powerful as an advice bit, i. e., a single random bit can be used to achieve an expected competitive ratio of 2. Moreover, we prove that an arbitrary amount of additional randomness does not help at all, no randomized algorithm can achieve an expected competitive ratio better than $2-\varepsilon$, for any $\varepsilon > 0$.

We analyze the behavior of online algorithms with advice that are allowed to overpack the knapsack by some small constant amount of $\delta$. In contrast to the original model, we show that, in this case, a constant number of advice bits is already sufficient to achieve a near-optimal competitive ratio when dealing with the unweighted knapsack problem.

In the second part of the paper, we study the weighted version of the problem; obviously, all lower bounds carry over immediately from the unweighted one. We show that, with less than $\log n$ advice bits, no online algorithm is competitive and that we can be arbitrarily close to the optimum when using $\mathcal{O}(\log n)$ advice bits. Conversely, randomized online algorithms are not competitive independent of the number of random bits used.

## *1.1. Related Work*

The decision version of the offline knapsack problem is one of Karp's famous 21 $\mathcal{NP}$-hard problems [23] and thus one of the first computing problems that were considered solvable, but intractable. Regarding the optimization variant, the knapsack problem admits a *pseudo-polynomial-time algorithm* using dynamic programming. Ibarra and Kim used this approach to design a *fully polynomial-time approximation scheme* [19]. An online version of the knapsack problem was first analyzed by Marchetti-Spaccamela and Vercellis [28]; they gave the following result.

**Theorem 1 (Marchetti-Spaccamela and Vercellis [28]).** *No deterministic online algorithm for* SimpleKnapsack *(and thus* Knapsack*) without advice is competitive.* □

Due to this negative result, relaxed formulations were investigated. Iwama and Taketomi introduced the online removable knapsack problem; this variant allows to discard items that are already packed retrospectively [20]. A two-dimensional version of the problem was introduced by Han et al. [13].

The classical tool to analyze online algorithms, competitive analysis, was introduced by Sleator and Tarjan in the mid 80's [31]. Dobrev et al. were the first to equip online algorithms with an oracle that helps them to compute a high-quality solution [7]. Their model was then revised and applied to a number of different online problems; for a detailed introduction to the advice complexity of online problems, see [3, 18]. More results on the advice complexity of specific problems are found in [2, 10, 27], the relationship between advice complexity and randomized algorithms is discussed in [2, 26].

It was, however, pointed out earlier that competitive analysis, i. e., comparing online algorithms to an optimal offline algorithm, may be inaccurate to analyze the performance of algorithms that work under uncertainty, see, e. g., [1]. The textbook example is the comparison between the two strategies *least recently used* and *first-in-first-out* for the paging problem. The first one outperforms the latter in practice [33] while both are equally bad when being analyzed by competitive analysis; as a matter of fact, *no* deterministic strategy is better than $K$-competitive where $K$ denotes the computer's cache size [31]. Similar unsatisfying results were obtained for numerous other problems and thus a large number of alternative measurements were introduced to better capture what works well in the real world; for an overview, we refer to [8]. To this end, for the online knapsack problem, Han and Makino studied the removable knapsack problem while additionally allowing the online algorithm to cut the items at most $k$ times [14]; they showed a tight bound of $(k+1)/k$ on the competitive ratio. Let us point out two further concepts that are important for our following investigations.

The first one is known as *resource augmentation* and it is widely applicable. Here, the online algorithm is allowed to use more resources than the optimal offline algorithm it is compared to. Resource augmentation was introduced by Kalyanasundaram and Pruhs [22] and was since then applied to a large number of online problems. A straightforward way to allow an online algorithm to use more resources than the offline algorithm for the knapsack problem is to permit *overpacking* of the knapsack, which we mentioned above. This idea was used for the online (removable) knapsack problem in [14, 21]. In this paper, we combine this technique with the concept of computing with advice. For further applications of resource augmentation, we refer to, e. g., [6, 22, 29].

The second concept, widely used in the area of scheduling, is to study so-called *semi-online* problems. These are frameworks in which some specific property about the input is known in advance; for instance, we might be given the sum of all the processing times of all jobs that we want to schedule before we get the first piece of the input [9, 11, 15, 16, 24, 30]. Note that the approach of using advice is far more general, as the advice string may encode any information and is not restricted to some specific parameter.

### *1.2. Organization of this Paper*

In Section 2, we formally introduce online algorithms with advice and the online knapsack problem. We study the unweighted version of the problem in Section 3. Subsection 3.1 is devoted to the analysis of deterministic online algorithms with advice. In Subsection 3.2, we investigate the random bit complexity of the problem, and in Subsection 3.3, we deal with the advice complexity in the resource augmentation model. In Section 4, we consider the general, weighted knapsack problem. We conclude the paper with some remarks on future work in Section 5.

## 2. Preliminaries

In this section, we formally define the notions used in the following. All logarithms in this paper are taken to be binary, unless stated otherwise.

**Definition 1 (Online Maximization Problem).** An online maximization problem consists of a set $\mathcal{I}$ of inputs and a *gain function.* Every input $I \in \mathcal{I}$ is a sequence of *requests* $I = (x_1, \ldots, x_n)$. Furthermore, a set of feasible outputs (or solutions) is associated with every $I$; every output is a sequence of *answers* $O = (y_1, \ldots, y_n)$. The gain function assigns a positive real value $\mathrm{gain}(I, O)$ to every input $I$ and any feasible output $O$. If the input is clear from the context, we omit $I$ and denote the gain of $O$ as $\mathrm{gain}(O)$. For every input $I$, we call any output $O$ that is feasible for $I$ and has largest possible gain an *optimal solution of* $I$, denoted by $\texttt{Opt}(I)$, where $\texttt{Opt}$ is an optimal offline algorithm. If $I$ is clear from the context, we also denote the optimal solution by $\mathcal{O}pt = \texttt{Opt}(I)$.

We now formally define online algorithms with advice for online maximization problems and their competitive ratios.

**Definition 2 (Online Algorithm with Advice).** Consider an input $I$ of an online maximization problem. An *online algorithm* $\texttt{A}$ *with advice* computes the output sequence $\mathcal{A}^{\phi} = \texttt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ 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[1] binary sequence. We denote the gain of the computed output by $\mathrm{gain}(\texttt{A}^{\phi}(I))$. The algorithm $\texttt{A}$ is *c-competitive with advice complexity* $s(n)$ if there exists a constant $\alpha$ such that, for every $n$ and for each $I$ of length at most $n$, there exists some $\phi$ such that $\mathrm{gain}(\texttt{A}^{\phi}(I)) \geq \frac{1}{c} \cdot \mathrm{gain}(\texttt{Opt}(I)) - \alpha$ and at most the first $s(n)$ bits of $\phi$ have been accessed during the computation of $\texttt{A}^{\phi}(I)$. If $\texttt{A}$ is $c$-competitive for $\alpha = 0$, we call it *strictly c-competitive.*

A detailed introduction into the theory of advice complexity is found in [18].

**Definition 3 (Online Knapsack Problem).** The *online knapsack problem*, KNAPSACK for short, is the following maximization problem. The input consists of a sequence of $n$ items that are tuples of weights and values, i. e., $S = \{s_1, \ldots, s_n\}$, $s_i = (w_i, v_i)$, where $0 < w_i \leq 1$ and $v_i > 0$ for $i \in \{1, \ldots, n\}$. A feasible solution is any set of indices $S' \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S'} w_i \leq 1$; the goal is to maximize $\sum_{i \in S'} v_i$. The items are given in an online fashion. For each item, an online algorithm $\texttt{A}$ must specify whether it is part of the solution as soon as it is offered. In the *simple* version of KNAPSACK, denoted by SIMPLEKNAPSACK, each item has a value smaller than 1 that is equal to its weight.

Since the value of an optimal solution for any instance of SIMPLEKNAPSACK is bounded by the constant capacity 1 of the knapsack, we only consider strict competitiveness in this paper. For simplicity, we subsequently abbreviate the term "strictly competitive" by "competitive".

## 3. The Unweighted Case

Let us begin by considering an online algorithm $\texttt{G}$ that realizes a straightforward greedy approach. This means that $\texttt{G}$ takes any item while there is space left for it in the knapsack. Of course, this strategy also fails in general (as Theorem 1 implies), but for a subset of the instances it works quite well.

**Observation 1.** *Let $I$ denote any instance of* SIMPLEKNAPSACK *where every item has a weight of at most $\beta$. Then $\texttt{G}$ achieves a gain of at least $1 - \beta$ or it is optimal.*

Indeed, if the sum of all weights is less than one, $\texttt{G}$ is optimal. However, if this is not the case, the space that is not covered by $\texttt{G}$ cannot be larger than $\beta$.

[1]The online algorithm may read an arbitrary, but finite prefix of the sequence; note that, if the whole advice sequence were finite, additional information could be encoded into the *length* of the advice as in [7].

*3.1. Online Algorithms with Advice*

To enable online algorithms to achieve better results, we now equip these algorithms with an advice tape as in Definition 2. At first, we study the *information content* of the problem, i. e., the number of advice bits both sufficient and necessary to produce an optimal output. Obviously, there is a linear upper bound.

**Theorem 2.** *There exists an optimal online algorithm for* SimpleKnapsack *that uses $n$ bits of advice.*

Proof. For each of the $n$ items, one bit of advice tells the algorithm whether this item is part of an arbitrary but fixed optimal solution or not. □

It might surprise that this bound is indeed tight as the next theorem shows.

**Theorem 3.** *Any online algorithm with advice for* SimpleKnapsack *needs at least $n-1$ advice bits to be optimal.*

Proof. For any $n$, consider the set of all inputs of the form $1/2, 1/4, \ldots, 1/2^{n-1}, t(B)$, where $t(B)$ is defined as

$$t(B) := 1 - \sum_{i=1}^{n-1} B_i 2^{-i},$$

for some vector $B \in \{0,1\}^{n-1}$. Consider the first $n-1$ items of these input instances. Any two different subsets of these items have a different sum. From this, it directly follows that, for any distinct value of $B$, there exists a *unique* optimal solution with gain 1. In other words: If $t(B)$ is revealed, there was only one correct choice for the algorithm.

If any online algorithm uses strictly less than $n-1$ bits, it cannot (by the pigeonhole principle) distinguish between all $2^{n-1}$ different inputs. Hence, it will output the same subset of the first $n-1$ items for two different input instances and thus produces a sub-optimal solution for at least one of them. □

Next, let `Aone` be an online algorithm reading one bit of advice. This bit indicates whether there exists an item within the input that has a weight of $> 1/2$. If this bit is zero, `Aone` acts greedily, if it is one, `Aone` takes nothing until an item with weight $> 1/2$ appears (and anything else afterwards).

**Theorem 4.** *The online algorithm* `Aone` *for* SimpleKnapsack *is 2-competitive.*

Proof. Suppose, there is no item with weight $> 1/2$. In this case, the claim directly follows from Observation 1. However, if there exists an item with weight $> 1/2$, the proof of the claim is trivial. □

This result seems counterintuitive. With merely one bit of advice and a straightforward approach, we jump from an unbounded output quality to 2-competitiveness. However, any further increase of the number of advice bits does not help until a logarithmic number is reached. Algorithm `Aone` is therefore the best we can hope for when dealing with any constant number of advice bits.

**Theorem 5.** *Let $b < \lfloor \log(n-1) \rfloor$ and let $\varepsilon > 0$. No online algorithm for* SimpleKnapsack *using $b$ bits of advice is better than $(2-\varepsilon)$-competitive.*

Proof. Let $\delta := \varepsilon/(4-2\varepsilon)$ and let `A` be any online algorithm with advice that reads $b$ advice bits. Consider the class $\mathcal{I}$ of inputs $I_j$, for $1 \le j \le n-1$, of the form

$$\frac{1}{2}+\delta, \frac{1}{2}+\delta^2, \ldots, \frac{1}{2}+\delta^j, \frac{1}{2}-\delta^j, \frac{1}{2}+\delta, \ldots, \frac{1}{2}+\delta,$$

where the item $\frac{1}{2}+\delta$ appears $n-j-1$ times at the end of the instance, for $j \in \{1,\dots,n-1\}$. Obviously, since $|\mathcal{I}| > 2^b$, there are more inputs than strategies to choose from and, thus, there are two different inputs for one advice string. In order to be optimal, $\mathtt{A}$ needs to take the $j$th and $(j+1)$th item for the instance $I_j$ and, hence, this choice is unique for every input from $\mathcal{I}$. For any other choice of items on the instance $I_j$, $\mathtt{A}$ achieves a gain of at most $\frac{1}{2}+\delta$, leading to a competitive ratio of

$$\frac{1}{\frac{1}{2}+\delta} = 2-\varepsilon$$

as we claimed. □

The competitive ratio that is achievable with respect to the number of used advice bits now makes a second jump as stated by the following theorem.

**Theorem 6.** *Let $\varepsilon > 0$. There exists an online algorithm* $\texttt{Slog}$ *with advice for* SimpleKnapsack *that achieves a competitive ratio of $1+\varepsilon$ reading*

$$\left\lceil \frac{3\varepsilon+3}{\varepsilon} \right\rceil \cdot \lceil \log n \rceil + 2 \cdot \left\lceil \log\left( \left\lceil \frac{3\varepsilon+3}{\varepsilon} \right\rceil + 1 \right) \right\rceil + 2 \cdot \lceil \log \lceil \log n \rceil \rceil + 1$$

*bits of advice.*

Proof. Let $\delta := \varepsilon/(3\varepsilon+3)$. Suppose there does not exist any item within the input with weight larger than $\delta$ which can be indicated using one bit at the beginning of the advice tape. Then, $\texttt{Slog}$ may safely take sets greedily which leads to a competitive ratio of

$$\frac{1}{1-\delta} = 1 + \frac{\delta}{1-\delta} = 1 + \frac{\varepsilon}{3+2\varepsilon} \le 1+\varepsilon.$$

Now assume the contrary, i. e., there is some item of weight $> \delta$. The oracle inspects an arbitrary, but fixed optimal solution which consists of two disjoint sets of items $S_1$ and $S_2$, where $S_1$ denotes the set of *heavy* items of weight $> \delta$ and $S_2$ contains all *light* items of weight $\le \delta$. Let $i = |S_1|$, $j = |S_2|$, and let $s_1$ [$s_2$] be the sum of all weights of the items in $S_1$ [$S_2$]. The indices of all heavy items are written onto the advice tape using $i \cdot \lceil \log n \rceil$ bits (also, we need to communicate $i$ which can be done using another $\lceil \log \lceil 1/\delta \rceil \rceil$ bits). Since the sum of all weights of any solution does not exceed 1, we clearly have $i \le 1/\delta$, i. e., $i$ is constant with respect to $n$. For being able to decode the advice string, additionally the length $\lceil \log n \rceil$ of such an index has to be included in the advice in some self-delimiting form using $2\lceil \log \lceil \log n \rceil \rceil$ bits.[2]

Moreover, let the oracle encode a number $k$ on the advice tape, where $k$ is such that $k\delta \le s_2 < (k+1)\delta$. Since $\texttt{Slog}$ knows $\varepsilon$ and therefore $\delta$, it computes $k\delta$ and thus obtains a lower bound on $s_2$, i. e., the part of the considered optimal solution that is due to the light items. Every such light item is taken as long as their sum is below $k\delta$. It is immediate that $k \le 1/\delta$, due to $s_2 \le 1$. According to Observation 1, $\texttt{Slog}$ packs at least as many items from $S_2$ such that their sum is not smaller than $k\delta - \delta \ge s_2 - 2\delta$. Observe that, if there do not exist any light items (i. e., $S_2$ is empty), $\texttt{Slog}$ is clearly optimal, because it packs all heavy items into the knapsack. Thus, we may assume that there exists at least one light item and the optimal solution takes it. Furthermore, if, under this assumption, the optimal solution would be smaller than $1-\delta$, it follows that it takes all small items. This can be communicated to $\texttt{Slog}$ with setting $k := \lceil 1/\delta \rceil$, resulting in an optimal algorithm. We therefore assume the contrary, i. e., that $\text{gain}(\mathit{Opt}) \ge 1-\delta$. Consequently, we get a competitive ratio of

$$\frac{s_1+s_2}{s_1+s_2-2\delta} \le \frac{1}{1-3\delta} = 1 + \frac{3\delta}{1-3\delta} = 1+\varepsilon.$$

[2]For an example on how to construct such self-delimiting encodings, see, for example, the proof of Theorem 5 in [2].

Since $k$ is an integer from the range $0 \ldots \lceil 1/\delta \rceil$, it can be encoded using $\lceil \log(\lceil 1/\delta \rceil + 1) \rceil$ bits. The total number of advice bits used by the algorithm is

$$1 + i \cdot \lceil \log n \rceil + 2 \cdot \left\lceil \log\left( \left\lceil \frac{1}{\delta} \right\rceil + 1 \right) \right\rceil + 2 \cdot \lceil \log \lceil \log n \rceil \rceil$$

and the claim follows. □

*3.2. Randomized Online Algorithms*

In this section, we study the random bit complexity of the problem. At first, suppose we use the same algorithm as in Theorem 4, but guess the advice bit. Obviously, this algorithm, which we call Rone, is 2-competitive with probability 1/2 and not competitive with the same probability, i. e., 4-competitive in expectation. This bound is tight as the next theorem shows.

**Theorem 7.** *The randomized online algorithm* Rone *for* SimpleKnapsack *cannot be better than* 4*-competitive in expectation.*

Proof. Let $\varepsilon < 1/6$. Consider three items with weights $1/2 - \varepsilon, 3\varepsilon, 1/2 - \varepsilon$. A greedy approach takes the first two items and therefore obtains a gain of $1/2 + 2\varepsilon$, whereas the algorithm that waits for an item of weight $\geq 1/2$ gains nothing. Thus, Rone is $c$-competitive only for

$$c \geq \frac{1 - 2\varepsilon}{\frac{1}{2}\left(\frac{1}{2} + 2\varepsilon\right) + \frac{1}{2} \cdot 0} = 4 \cdot \frac{1 - 2\varepsilon}{1 + 4\varepsilon}.$$

As $\varepsilon$ can be arbitrarily small, Rone cannot be better than 4-competitive. □

It seems somehow intuitively clear that randomization (the average over good and bad) is twice as bad as using advice (*always* good). However, while this is right for this specific strategy, we now show the following: Remarkably, randomization and advice are equally powerful for SimpleKnapsack when dealing with a small amount of either random or advice bits.

**Theorem 8.** *There exists a randomized online algorithm* Rone′ *for* SimpleKnapsack *that is* 2*-competitive in expectation and that uses exactly one random bit.*

Proof. Consider the following deterministic online algorithms $A_1$ and $A_2$; $A_1$ is the straightforward greedy algorithm for SimpleKnapsack. $A_2$ locally simulates $A_1$ and does not take any item until it realizes that an item just offered would not fit into $A_1$'s solution anymore. $A_2$ then acts greedily starting from here. If the input consists of items that, in the sum, have a weight less than the knapsack's capacity, $A_1$ is obviously optimal, while $A_2$ might have gain zero. If, however, this is not the case, the gain of $A_1$ plus the gain of $A_2$ is at least 1.

Let Rone′ choose between $A_1$ and $A_2$ uniformly at random. Obviously, one random bit suffices to do that. We then immediately get that the expected gain of Rone′ is at least $(1/2) \cdot \text{gain}(\mathcal{O}pt)$, and the competitive ratio of Rone′ is thus at most 2. □

Note that the lower bound of 2 on the competitive ratio from algorithms with advice (see Theorem 5) carries over immediately for the randomized case. Thus, the bound of Theorem 8 is tight. The above results imply that randomization and advice are equally powerful when we consider a sub-logarithmic number of bits.

As we have seen before (see Theorem 6), logarithmic advice helps a lot. On the other hand, we now show that this is not the case for randomization.

**Theorem 9.** *No randomized online algorithm for* SimpleKnapsack *can be better than* 2*-competitive (independent of the number of random bits).*

PROOF. Consider the following class of inputs. At first, an item of weight $\varepsilon > 0$ is offered. After that, either nothing else is offered or an additional item of weight 1.

Now consider any algorithm R that decides to use the first item with non-zero probability $p$ (otherwise, its gain is obviously zero). If R takes the item, of course, it cannot use the second one if it is offered. On the other hand, if R does not take the first item (with probability $1 - p$), it does not have any gain if there is no second item. Suppose the second item is offered. Algorithm R then has competitive ratio

$$\frac{1}{p \cdot \varepsilon + (1-p) \cdot 1},$$

and, if the second item is not offered, R has competitive ratio $\varepsilon/(p \cdot \varepsilon)$. By equalizing the ratios, we get

$$\frac{1}{(\varepsilon - 1) \cdot p + 1} = \frac{1}{p} \quad \Longleftrightarrow \quad p = \frac{1}{2 - \varepsilon}$$

and, thus R is no better than $(2 - \varepsilon)$-competitive. □

Let us summarize: With one random bit, we can achieve a (tight) bound of 2. However, any additional random bit does not help at all.

*3.3. Resource Augmentation*

In this subsection, we allow the online algorithms considered to use more powerful resources than the optimal offline algorithm they are compared against. More precisely, an online algorithm is permitted to overpack the knapsack by some $\delta$, $0 < \delta < 1$, whereas the optimal solution is merely allowed to fill it up to 1. It is known that this approach allows for a $1/\delta$-competitive algorithm and this bound is tight [21]. Now we consider online algorithms with advice and show that overpacking allows to save a huge amount of advice bits.

**Theorem 10.** *Let $1/4 > \delta > 0$. There exists an online algorithm* AUG *for* SIMPLEKNAPSACK *that achieves a competitive ratio of $1 + 3\delta/(1 - 4\delta)$ in the $\delta$-resource-augmented model, using at most*

$$\left\lceil 2 \log \left\lceil \frac{1}{\delta} \right\rceil + \frac{1}{\delta} \cdot \log \left\lceil \frac{1}{\delta^2} \right\rceil \right\rceil + 1$$

*advice bits.*

PROOF. Consider any instance $I$ and let $\mathcal{O}pt = \text{OPT}(I)$ denote an optimal solution computed by an optimal offline algorithm OPT. Suppose gain(OPT) $\leq 1/2$. In this case, there is obviously no item with weight $> 1/2$ and a simple greedy strategy enables AUG to be optimal. We fix the first advice bit to indicate whether $\mathcal{O}pt$ has a weight of $1/2$ or smaller and, in the further analysis, assume the contrary.

To this end, let $\mathcal{O}pt = \{x_1, \ldots, x_k\} \dot\cup \{y_1, \ldots, y_m\}$ denote an optimal solution computed by an algorithm OPT where the items $x_i$ have weights $\geq \delta$ and the items $y_j$ have weights $< \delta$. Obviously, we have $k \leq 1/\delta$. AUG knows $\delta$ and is designed such that it reads all the approximate weights (computed via an integer division by $\delta^2$) of all *heavy* items and the approximate fraction of the knapsack that is filled using *light* ones from the advice tape.

First, we show how the heavy items are encoded. To this end, let

$$\overline{x}_i := j \text{ such that } j \cdot \delta^2 \leq x_i < (j+1) \cdot \delta^2,$$

for every heavy item $x_i$. All $\overline{x}_i$s are sequentially written onto the advice tape and read by AUG right after the first item is offered. Thus, if AUG is offered any item $x'$, it checks whether the corresponding $\overline{x}_i$ is part of the advice, that is, if there exists $\overline{x}_i$ such that $\delta^2 \cdot \overline{x}_i \leq x' < \delta^2 \cdot (\overline{x}_i + 1)$. If so, $x'$ is taken into the knapsack as an item $x'_i$ corresponding to $x_i$; else it is neglected. In the former case, there exists $x_i$ that is part of $\mathcal{O}pt$ and $x_i - \delta^2 < x'_i < x_i + \delta^2$. Clearly, there are at

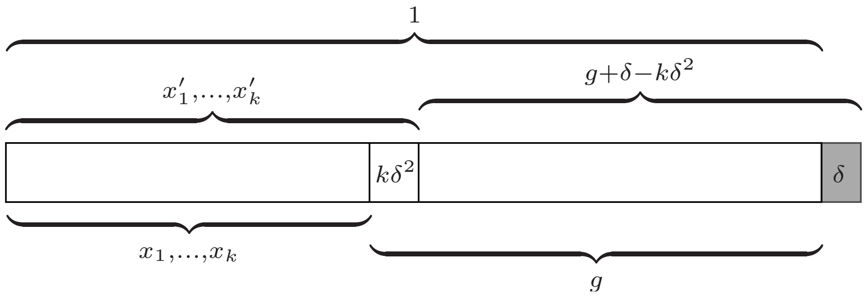

Figure 1: The gains of both $\mathcal{O}pt$ and AUG.

most $k$ different $\overline{x}_i$s (as many as there are corresponding heavy items) and each is at most of weight $1/\delta^2$; hence, to communicate all of these values, we need no more than

$$k \cdot \log\left\lceil \frac{1}{\delta^2} \right\rceil \le \left\lceil \frac{1}{\delta} \cdot \log\left\lceil \frac{1}{\delta^2} \right\rceil \right\rceil$$

advice bits, which is obviously constant with respect to $n$. However, to be able to decode the advice, AUG needs to know $k$ beforehand.[3] The value of $k$ can be written onto the advice tape in a self-delimiting form using another $2\lceil \log k \rceil \le 2\lceil \log 1/\delta \rceil$ additional bits. The number of bits needed to encode the items $\overline{x}_i$ can be calculated by AUG without any further knowledge.

Note that we have

$$-\delta + \sum_{i=1}^{k} x_i \le \sum_{i=1}^{k} x'_i \le k \cdot \delta^2 + \sum_{i=1}^{k} x_i \le \delta + \sum_{i=1}^{k} x_i,$$

which means that, for every heavy item, AUG chooses an item such that the algorithm uses at most $\delta^2$ more space within the knapsack. Since there are $k \le 1/\delta$ heavy items, the sum of the chosen heavy items uses at most $\delta$ more space than the heavy items in $\mathcal{O}pt$.

We now distinguish two cases regarding the weight of an optimal solution.

**Case 1.** Suppose that $\text{gain}(\mathcal{O}pt) < 1 - \delta$. This directly implies that all light items are part of the optimal solution because, otherwise, $\mathcal{O}pt$ would not be optimal. Algorithm AUG uses at most $\delta$ more space for the heavy items as $\mathcal{O}pt$, and since AUG is allowed to overpack the knapsack by $\delta$, it takes all light items as well. On the other hand, the sum of weights of the heavy items chosen by AUG is at least $-\delta + \sum_{i=1}^{k} x_i$, hence AUG has gain at least $\text{gain}(\mathcal{O}pt) - \delta$. Thus, the competitive ratio obtained by AUG can be bounded by

$$\frac{\text{gain}(\mathcal{O}pt)}{\text{gain}(\mathcal{O}pt) - \delta} \le 1 + \frac{\delta}{\text{gain}(\mathcal{O}pt) - \delta} \le 1 + \frac{2\delta}{1 - 2\delta},$$

where the last inequality follows from the fact that $\text{gain}(\mathcal{O}pt) \ge 1/2$.

**Case 2.** Suppose that $1 - \delta \le \text{gain}(\mathcal{O}pt) \le 1$. Let us now consider the light items. To this end, let $g := 1 - \sum_{i=1}^{k} x_i$ denote the space $\mathcal{O}pt$ is left with after packing all heavy items into the knapsack (see Figure 1); AUG does not know $g$ but calculates the approximate value

$$g' := 1 - \sum_{i=1}^{k} (\overline{x}_i + 1)\delta^2.$$

It follows that $g - \delta \le g' \le g$. Note that both AUG and $\mathcal{O}pt$ have all light items available. Now, for $z \in \{g, g'\}$, consider the instance $I(z)$ shrunk to a knapsack capacity of $z$ and light

[3]Recall that the length of the advice is not known to AUG, but AUG reads the advice from an infinite tape.

items only and let $\mathcal{O}pt(z)$ denote the corresponding optimal solution for this instance. Since `AUG` acts greedily on $I(g')$, by Observation 1 it follows that `AUG` is either optimal or has a gain of at least

$$g' - \delta \geq g - 2\delta \geq \text{gain}(\mathcal{O}pt(g)) - 2\delta.$$

On the other hand, `Opt` obtains a gain of exactly $\text{gain}(\mathcal{O}pt(g))$ on $I(g)$. Thus,

$$\begin{aligned}\frac{\text{gain}(\mathcal{O}pt)}{\text{gain}(\texttt{AUG}(I))} &\leq \frac{\text{gain}(\mathcal{O}pt)}{\sum_{i=1}^{k} x'_i + g' - \delta} \leq \frac{\text{gain}(\mathcal{O}pt)}{\sum_{i=1}^{k} x'_i + g - 2\delta} \leq \frac{\text{gain}(\mathcal{O}pt)}{\sum_{i=1}^{k} x_i + g - 3\delta} \\ &\leq \frac{\text{gain}(\mathcal{O}pt)}{\sum_{i=1}^{k} x_i + \text{gain}(\mathcal{O}pt(g)) - 3\delta} = \frac{\text{gain}(\mathcal{O}pt)}{\text{gain}(\mathcal{O}pt) - 3\delta} \leq 1 + \frac{3\delta}{1-4\delta}\,,\end{aligned}$$

which concludes the proof. □

The above results show that resource augmentation allows for a much more efficient way to use the advice. Without the ability to overpack the knapsack, we need a number of advice bits that grows logarithmically with $n$ to achieve a competitive ratio that is close to 1, whereas constant advice suffices in the $\delta$-resource-augmented model; more specifically, to get a competitive ratio of $1 + \varepsilon$, we choose $\delta = \varepsilon/(3 + 4\varepsilon)$ and apply Theorem 10.

Let us briefly discuss a randomized setting.

**Theorem 11.** *No randomized online algorithm for* SimpleKnapsack *can be better than* $(2 - \delta)$*-competitive (independent of the number of random bits) in the* $\delta$*-resource-augmented model.*

Proof. The proof is a straightforward adaptation of the proof of Theorem 9; the only difference is that the two items now have a weight of $\delta + \varepsilon$ and 1, respectively, for an arbitrarily small $\varepsilon > 0$. □

## 4. The Weighted Case

We now consider the general knapsack problem, Knapsack, from Definition 3, where every item has both a weight and a value.

### *4.1. Online Algorithms with Advice*

Our results only hold if we restrict ourselves to instances where the gains and weights can be represented within polynomial space. More formally, for any item $x$, let $w(x)$ be the weight of $x$, $g(x)$ be the gain of $x$ and $r(x) := g(x)/w(x)$ be the ratio of its gain and weight. We assume that, for every $x$, $g(x)$ and $w(x)$ are rational numbers, and their numerators and denominators are bounded by $2^{p(n)}$ for some fixed polynomial $p(n)$, where $n$ is the input size. First of all, we note that the lower bounds for SimpleKnapsack from the previous section carry over immediately since we are now dealing with a generalization of the above problem. Second, Theorem 2 obviously also applies for the general knapsack problem. However, for the weighted version, small advice is a lot less powerful.

**Theorem 12.** *No online algorithm for* Knapsack *using strictly less than* $\log n$ *bits of advice is competitive.*

Proof. Suppose that some arbitrary online algorithm `A` with advice reads $k < \log n$ advice bits which allows it to distinguish at most $2^k$ different inputs. We construct a set $\mathcal{I}$ of $n$ different instances as follows. Let $\alpha := 2^n$ and let $I_s$ be the instance determined by the $n$ items $(1, \alpha), (1, \alpha^2), \ldots, (1, \alpha^s), (1, 1), \ldots, (1, 1), (1, 1)$, for $s \in \{1, \ldots, n\}$ and $\mathcal{I} := \{I_s \mid 1 \leq s \leq n\}$. Obviously, since $|\mathcal{I}| > 2^k$, there are more inputs than strategies to choose from and, thus, there are two different inputs for one advice string. Let these two instances be $I_i$ and $I_j$ and assume $i > j$. The unique optimal solution for $I_i$ $[I_j]$ fills the knapsack with the $i$th [$j$th] item yielding a gain of $\alpha^i$ $[\alpha^j]$.

Clearly, if A does not choose the $j$th item when given the instance $I_j$, its gain is at least a factor of $\alpha$ away from gain($\mathcal{O}pt$). Since A cannot distinguish between $I_j$ and $I_i$ in the first $j$ time steps (and it is given the same fixed advice string) it also takes the $j$th item when given $I_i$. This results in a competitive ratio of at least $\alpha^i/\alpha^j \geq \alpha$ finishing the proof. □

In the following, we show how to solve the general knapsack problem almost optimally when using logarithmic advice. This implies that the bound from Theorem 12 is asymptotically tight.

**Theorem 13.** *Let $\varepsilon > 0$. There exists an online algorithm* WLOG *with advice for* KNAPSACK *that achieves a competitive ratio of $1+\varepsilon$ using at most $\mathcal{O}(\log n)$ bits of advice. Here, the $\mathcal{O}$ notation hides a multiplicative constant depending on $\varepsilon$ and on the degree $d$ of the polynomial $p(n)$.*

PROOF. On an intuitive level, the strategy of WLOG can be outlined as follows. The algorithm packs all the *expensive* items that are part of an optimal solution $\mathcal{O}pt$; other expensive items are discarded. Moreover, WLOG uses all items that have a large value/weight ratio as long as this ratio is larger than some lower bound. However, this lower bound cannot be communicated with absolute accuracy; items that are part of $\mathcal{O}pt$, but whose ratio is too small, are explicitly communicated to WLOG if their weight is larger than some specific threshold. Finally, WLOG calculates a bound on the space that it fills greedily by items that are not too heavy and that have a good value/weight ratio.

We now formalize this idea. To this end, let $\delta := (\sqrt{1+\varepsilon}-1)/(2\sqrt{1+\varepsilon}+1)$. Consider any optimal solution $\mathcal{O}pt$ and let

$$c' := (1+\delta)^{\lfloor \log_{1+\delta}(\text{gain}(\mathcal{O}pt))\rfloor},$$

i. e., $c'$ is an approximation of gain($\mathcal{O}pt$) such that

$$\frac{\text{gain}(\mathcal{O}pt)}{1+\delta} < c' \leq \text{gain}(\mathcal{O}pt).$$

Next, let $x_1, \ldots, x_k$ be all items in $\mathcal{O}pt$ with gain at least $\delta \cdot c'$. Since there are at most gain($\mathcal{O}pt$)/$(\delta \cdot c')$ such items, we immediately get $k \leq (1+\delta)/\delta$.

Let $\mathcal{S}_1$ be an (offline) solution constructed as follows. At first, all expensive items $x_1, \ldots, x_k$ are taken; then, the rest of the knapsack is filled using items that have gains less than $\delta \cdot c'$ greedily by the ratio of their gain and weight in descending order. Consider $\mathcal{S}_1$ plus the item $x$ that is the first one that did not fit into the knapsack in the greedy phase of $\mathcal{S}_1$'s construction. Clearly, $\mathcal{S}_1 \cup \{x\}$ has a larger gain than $\mathcal{O}pt$. Since $g(x) \leq \delta \cdot c' \leq \delta \cdot \text{gain}(\mathcal{O}pt)$, we get that

$$\text{gain}(\mathcal{S}_1) \geq (1-\delta)\text{gain}(\mathcal{O}pt).$$

Let $y_1, \ldots, y_l$ denote the items of $\mathcal{S}_1$ added in the greedy phase. Without loss of generality, assume that $r(y_1) \geq r(y_2) \geq \ldots \geq r(y_l)$ and let

$$r' := (1+\delta)^{\lceil \log_{1+\delta}(r(y_l))\rceil},$$

i. e., $r'$ is an approximation of $r(y_l)$ such that

$$r(y_l) \leq r' < r(y_l) \cdot (1+\delta).$$

Let $m$ be the largest number such that $r(y_m) \geq r'$, i. e., the items $y_1, \ldots, y_m$ have ratios of at least $r'$ and all other items $y_{m+1}, \ldots, y_l$ have ratios between $r'$ and $r'/(1+\delta)$. Let $v$ be the space not occupied by $x_1, \ldots, x_k, y_1, \ldots, y_m$ in $\mathcal{S}_1$, i. e.,

$$v := 1 - \sum_{i=1}^{k} w(x_i) - \sum_{i=1}^{m} w(y_i).$$

Intuitively speaking, if we consider the part of the solution $\mathcal{S}_1$ that consists of the items $y_i$, for $i > m$, we see that this is a solution of an "almost-unweighted" knapsack instance with knapsack capacity $v$. Therefore, we can approximate it by a solution for the unweighted knapsack problem without doing much harm. To this end, let

$$v' := (1+\delta)^{\lfloor \log_{1+\delta} v \rfloor},$$

i. e., $v'$ is an approximation of $v$ such that

$$v/(1+\delta) < v' \leq v.$$

Furthermore, let

$$\{z_1, \dots, z_j\} = S := \{y_i \mid y_i \in \{y_{m+1}, \dots, y_l\}, w(y_i) \geq \delta \cdot v'\},$$

i. e., $z_1, \dots, z_j$ are all items from $\mathcal{S}_1$ that have a ratio of roughly $r'$ and whose weights are at least a $\delta$-fraction of $v'$. Since $v' > v/(1+\delta)$, there are at most $(1+\delta)/\delta$ such items.

Let

$$u := v - \sum_{i=1}^{j} w(z_i),$$

i. e., the space not occupied by $x_1, \dots, x_k, y_1, \dots, y_m, z_1, \dots, z_j$, and let

$$u' := (1+\delta)^{\lfloor \log_{1+\delta} u \rfloor},$$

i. e., $u'$ is an approximation of $u$ such that

$$u/(1+\delta) < u' \leq u.$$

Again, we consider an (offline) solution $\mathcal{S}_2$ that is constructed as follows. At first, all items $x_1, \dots, x_k, y_1, \dots, y_m, z_1, \dots, z_j$ are taken. After that, we consider all remaining items of weight less than $\delta \cdot v'$ and a ratio of at least $r'/(1+\delta)$; each of these items is added greedily to $\mathcal{S}_2$ if it fits into a reserved space of size $u'$. We now show that

$$\text{gain}(\mathcal{S}_2) \geq \frac{1-2\delta}{(1+\delta)^2}\text{gain}(\mathcal{S}_1). \tag{1}$$

To this end, consider two cases. If the greedy construction of $\mathcal{S}_2$ takes all possible items, $\mathcal{S}_2$ contains all items included in $\mathcal{S}_1$, and (1) follows trivially. Therefore, we may assume the contrary. Since the items $y_{m+1}, \dots, y_l$ have a ratio of at most $r'$ and weights of at most 1, it follows that the value of each such item is at most $r'$ as well. Hence, the gain of $\mathcal{S}_1$ is at most

$$\sum_{i=1}^{k} g(x_i) + \sum_{i=1}^{m} g(y_i) + v \cdot r' \leq \sum_{i=1}^{k} g(x_i) + \sum_{i=1}^{m} g(y_i) + v' \cdot (1+\delta) \cdot r'.$$

On the other hand, the gain of $\mathcal{S}_2$ is at least

$$\sum_{i=1}^{k} g(x_i) + \sum_{i=1}^{m} g(y_i) + v' \cdot (1-2\delta) \cdot \frac{r'}{1+\delta},$$

because the greedy step took items of total weight of at least $u' - \delta \cdot v'$ which is, together with all $z_i$, at least

$$u' - \delta \cdot v' + v - u \geq (1-\delta) \cdot v' + u' - u \geq (1-2\delta) \cdot v'.$$

All these items have a ratio of at least $r'/(1+\delta)$. It follows that (1) holds.

Putting all together, we finally get

$$\text{gain}(\mathcal{S}_2) \geq \frac{1-2\delta}{(1+\delta)^2}\text{gain}(\mathcal{S}_1) \geq \frac{(1-2\delta)^2}{(1+\delta)^2}\text{gain}(\mathcal{O}pt) = \frac{\text{gain}(\mathcal{O}pt)}{1+\varepsilon}$$

as claimed.

Let us now look at the number of advice bits necessary to be communicated to WLOG. At first, the oracle needs to encode $n$ and $k$ which can be done using no more than $2\lceil\log\lceil\log n\rceil\rceil + 2\lceil\log n\rceil$ bits. Furthermore, since WLOG knows $\delta$, it suffices to read at most

$$\left\lceil \log\lfloor\log_{1+\delta}\left(2^{p(n)}\right)\rfloor\right\rceil \le \log\left(\frac{\log\left(2^{p(n)}\right)}{\log(1+\delta)}\right) + 1 \in \mathcal{O}\big(\log\big(n^d\big)\big)$$

advice bits to communicate $c'$, where $d$ is the degree of the polynomial $p(n)$. We immediately see that, to encode $r'$, $v'$, and $u'$, we also need no more than $\mathcal{O}\big(\log\big(n^d\big)\big)$ bits. The indices of the items $x_i$ can be specified using $k\lceil\log n\rceil \le (1+\delta)/\delta\log n + 1$ additional bits. Similarly, the indices of the items $z_i$ can be communicated using $j\lceil\log n\rceil \le (1+\delta)/\delta\log n + 1$ bits.

We conclude that at most $\mathcal{O}\big(\log\big(n^d\big)\big) = \mathcal{O}(\log n)$ bits are needed in total. Finally, the online algorithm WLOG works as follows to construct $\mathcal{S}_2$ using the advice as specified above.

```
Algorithm WLOG
1. for any item $x$ do
2.   if $x = x_i$ for some $i$, use;
3.   else if $g(x) \ge \delta \cdot c'$, discard;
4.        else if $r(x) \ge r'$, use;
5.             else if $x = z_i$ for some $i$, use;
6.                  else if $r(x) < r'/(1+\delta)$ or $w(x) \ge \delta \cdot v'$, discard;
7.                       else if total weight of items taken at line 7 $\le u'$, use;
8.                            else discard;
9. end
```

This finishes our proof. □

*4.2. Randomized Online Algorithms*

Next, we show that no randomized online algorithm for Knapsack, independent of the number of random bits it reads, is competitive.

**Theorem 14.** *No randomized online algorithm for* Knapsack *is competitive (independent of the number of random bits).*

Proof. For any $n \ge 2$, consider the sequence

$$(1, 2), (1, 2^2), \ldots, (1, 2^n)$$

of $n$ items; clearly, there is space for exactly one of them in the knapsack. We construct the class of instances $\mathcal{I}$ that consists of every prefix of this sequence, i. e., for each instance $I_k$, the first $k$ items get offered and the optimal solution takes the last one among them such that it has a gain of $2^k$.

Conversely, any deterministic online algorithm that has any gain on $I_n$ picks exactly one item it packs into the knapsack. Let $A_k$ denote the deterministic algorithm that decides to wait until the $k$th item is offered and to take it. As all possible inputs from $\mathcal{I}$ are prefixes of $I_n$, the behavior of any deterministic algorithm on any of these instances is determined by its behavior on $I_n$; hence, it is either equivalent to some algorithm $A_k$ or it has no gain on any instance from $\mathcal{I}$.

We analyze the performance of $A_k$ on the uniform distribution on $\mathcal{I}$ and then use Yao's principle [32] to transfer our results to any randomized algorithm that deals with a worst-case instance from $\mathcal{I}$; more precisely, following Theorem 8.3 in [5], the expected competitive ratio of some randomized online algorithm R can be bounded from below by

$$r := \min_k \left\{ \frac{1}{\mathbb{E}\left[\frac{A_k(I_j)}{\textsc{Opt}(I_j)}\right]} \right\},$$

where $\mathbb{E}[X]$ denotes the expectation of a random variable $X$ according to the uniform distribution on $\mathcal{I}$.

If the concrete input is $I_k$, the gain of $A_k$ and the optimal gain are both $2^k$, which happens with probability $1/n$; if the randomly chosen instance is $I_j$, $j < k$ (i. e., if $k$ is too large), $A_k$'s gain is 0 and if the instance is $I_i$, $k < i$ (i. e., if $k$ is too small), $A_k$'s gain is again $2^k$. Note that the optimal gain is $2^i$ in these cases. Summing up, we get

$$\mathbb{E}\left[\frac{A_k(I_j)}{\texttt{Opt}(I_j)}\right] = \frac{1}{n}\cdot\sum_{i=1}^{k-1}\frac{0}{2^i} + \frac{1}{n}\cdot\frac{2^k}{2^k} + \frac{1}{n}\cdot\sum_{i=k+1}^{n}\frac{2^k}{2^i} = \frac{1}{n}\cdot\sum_{i=k}^{n}2^{k-i},$$

for any fixed $k$, and thus

$$r = \min_k\left\{\frac{n}{\sum_{i=k}^{n}2^{k-i}}\right\} \geq \frac{n}{2},$$

which proves the claim. □

Note that in the above proof, there are, for any $n$, only $n$ deterministic strategies and $\log n$ bits are sufficient to choose one uniformly. However, the proof implies that it does not help at all to increase the number of random bits in any way, because we made no assumption about the distribution `R` uses.

### *4.3. Resource Augmentation*

Iwama and Zhang showed that resource augmentation does not help when considering deterministic algorithms for Knapsack [21]. Note that all items which we used in the proof of Theorem 14 have a weight of 1 each. It thus follows immediately that resource augmentation does not help at all for randomized online algorithms as long as $\delta < 1$.

**Theorem 15.** *No randomized online algorithm for* Knapsack *is competitive in the $\delta$-resource-augmented model (independent of the number of random bits).* □

## 5. Conclusion

We have analyzed the advice complexity and the random bit complexity of the online knapsack problem. For the unweighted case, the advice complexity exhibits a very interesting phase transition: Less than $\log(n-1)$ advice bits do not improve over a single bit of advice, but $\mathcal{O}(\log n)$ advice bits already allow for an almost optimal competitive ratio. A similar phenomenon can be observed for the random bit complexity. Here, a single random bit allows for a competitive ratio of 2 and no additional randomness can improve this result. We have also seen that, when allowing online algorithms to overpack the knapsack a little bit, a constant number of advice bits suffices to produce an output that is arbitrarily close to the optimum. Finally, we have shown that $\mathcal{O}(\log n)$ bits are also sufficient to get arbitrarily close to an optimal solution for the weighted online knapsack problem; here, the $\mathcal{O}$ notation hides a larger constant as for the unweighted case. For this problem, randomization does not enable the design of an online algorithm that achieves any constant competitive ratio, not even if overpacking is allowed.

For further research, it would be interesting to see how randomized online algorithms with advice behave on this problem, i. e., whether some of the $\mathcal{O}(\log n)$ advice bits in the proofs of Theorems 6 and 13 can be substituted by some amount of random bits.

## References

[1] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1):3–26, 2003.

[2] H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič. On the advice complexity of the $k$-server problem. In: *ICALP 2011*, LNCS 6755, pages 207–218. Springer, 2011.

[3] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In: *ISAAC 2009*, LNCS 5878, pages 331–340. Springer, 2009.

[4] H.-J. Böckenhauer, D. Komm, R. Královič, and P. Rossmanith. On the advice complexity of the knapsack problem. In: *LATIN 2012*, LNCS 7256, Springer, 2012.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[6] J. Csirik and G. J. Woeginger. Resource augmentation for online bounded space bin packing. *Journal of Algorithms*, 44(2):308–320, 2002.

[7] S. Dobrev, R. Královič, and D. Pardubská. Measuring the problem-relevant information in input. *Theoretical Informatics and Applications (RAIRO)*, 43(3):585–613, 2009.

[8] R. Dorrigiv. *Alternative Measures for the Analysis of Online Algorithms.* PhD thesis, University of Waterloo, 2011.

[9] T. Ebenlendr and J. Sgall. Semi-online preemptive scheduling: One algorithm for all variants. *Theory of Computing Systems*, 48(3):577–613, Springer, 2011.

[10] Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.

[11] L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Operations Research Letters*, 30(4):269–275, Elsevier Science Publishers, 2002.

[12] A. Fiat, G. J. Woeginger (eds.). *Online Algorithms – The State of the Art*, LNCS 1442. Springer, 1998.

[13] X. Han, K. Iwama, G. Zhang. Online removable square packing. *Theory of Computing Systems*, 43(1):38–55, 2008.

[14] X. Han and K. Makino. Online removable knapsack with limited cuts. *Theoretical Computer Science*, 411(44-46):3956-3964, 2010.

[15] Y. He and Y. Jiang. Optimal algorithms for semi-online preemptive scheduling problems on two uniform machines. *Acta Informatica*, 40(5):367–383, Springer, 2004.

[16] Y. He and G. Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62(3):179–187, Springer, 1999.

[17] J. Hromkovič. *Algorithmics for Hard Problems.* Springer, 2nd edition, 2004.

[18] J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In: *MFCS 2010*, LNCS 6281, pages 24–36. Springer, 2010.

[19] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[20] K. Iwama and S. Taketomi Removable online knapsack problems. In: *ICALP 2002*, LNCS 2480, pages 293–305. Springer, 2002.

[21] K. Iwama and G. Zhang. Online knapsack with resource augmentation. *Information Processing Letters*, 110(22):1016–1020, 2010.

[22] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[23] R. M. Karp. Reducibility among combinatorial problems. In: *Proc. of a Symposium on the Complexity of Computer Computations*, pages 85–103. 1972.

[24] H. Kellerer, V. Kotov, M. Grazia Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, Elsevier Science Publishers, 1997.

[25] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[26] D. Komm and R. Královič. Advice complexity and barely random algorithms. *Theoretical Informatics and Applications (RAIRO)*, 45(2):249–267, 2011.

[27] D. Komm, R. Královič, and T. Mömke. On the advice complexity of the set cover problem. In: *CSR 2012*, LNCS 7353, pages 241–252, Springer, 2012.

[28] A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.

[29] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[30] S. S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, Elsevier Science Publishers, 2000.

[31] D. D. Sleator and R E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, Association for Computing Machinery, 1985.

[32] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In: *FOCS 1977*, pages 222–227. IEEE Computer Society, 1977.

[33] N. E. Young. The $k$-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, Springer, 1994.

[34] Y. Zhou, D. Chakrabarty, and R. M. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In: *WINE 2008*, LNCS 5385, pages 566–576. Springer, 2008.