Online Bin Packing with Advice*

Joan Boyar[1], Shahin Kamali[2], Kim S. Larsen[1], and Alejandro López-Ortiz[2]

1 **University of Southern Denmark, Denmark**
   `{joan,kslarsen}@imada.sdu.dk`
2 **University of Waterloo, Canada**
   `{s3kamali,alopez-o}@uwaterloo.ca`

──── **Abstract** ────

We consider the online bin packing problem under the advice complexity model where the "online constraint" is relaxed and an algorithm receives partial information about the future requests. We provide tight upper and lower bounds for the amount of advice an algorithm needs to achieve an optimal packing. We also introduce an algorithm that, when provided with $\log n + o(\log n)$ bits of advice, achieves a competitive ratio of $3/2$ for the general problem. This algorithm is simple and is expected to find real-world applications. We introduce another algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$. Finally, we provide a lower bound argument that implies that advice of linear size is required for an algorithm to achieve a competitive ratio better than $9/8$.
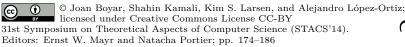
1 Introduction

In the classical one-dimensional bin packing problem the goal is to pack a given sequence of *items* into a minimum number of *bins* with fixed and equal capacities. For convenience, it is assumed that items sizes are in the range $(0, 1]$ and the capacities of bins are 1. In the *online* version of the problem, the items are revealed one by one, and an algorithm must pack each item without any knowledge about future items. The decisions of an online algorithm are irrevocable, i.e., it is not possible to move an item from one bin to another after it is *packed* in a bin.

The online bin packing problem has many applications in practice, from loading trucks subject to weight limitations to creating file backups in removable media [10]. Heuristics that have been proposed for the problem include Next-Fit (NF), First-Fit (FF), Best-Fit (BF), and the Harmonic-based class of algorithms. NF maintains a single *open* bin and places an item in that bin; in the case the item does not fit, it *closes* the bin and opens a new one. FF keeps a list of bins in the order they are opened, packs an item in the first bin that has enough space, and opens a new bin if necessary. BF performs similarly to FF, except that the bins are ordered in increasing order of their remaining capacity. Harmonic-based algorithms are based on the idea of packing items of similar sizes together in a bin. For Harmonic$_K$, an
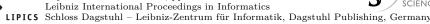
---

item has type i ($1 \le i \le K-1$) if it is in the range $(\frac{1}{i+1}, \frac{1}{i}]$, and type K if it is in the range $(0, \frac{1}{K}]$. The algorithm applies the NF strategy for items of each type separately.

As for other online problems, the standard method for comparing bin packing algorithms is competitive analysis. Under competitive analysis, the performance of an algorithm $\mathbb{A}$ is compared to that of OPT, which is the optimal offline algorithm. More precisely, the competitive ratio of an algorithm $\mathbb{A}$ is the asymptotically maximum ratio of the cost of $\mathbb{A}$ to that of OPT for serving the same sequence σ. FF and BF have the same competitive ratio of 1.7, while the best Harmonic-based algorithm has a competitive ratio of at most 1.58889 [22]. It is also known that no online algorithm can have a competitive ratio better than 1.54037 [3].

The total lack of information about the future is unrealistic in many real-world scenarios [13]. A natural approach for addressing this issue is to relax the problem by providing extra information about the input sequence. For the online bin packing problem, such relaxations have been studied in the contexts of *lookahead*, in which the online algorithm can look at the items arriving in the near future [16], and *closed bin packing*, in which the length of the request sequence is known to the online algorithm [1]. In both cases, the average performance of the online algorithm improves, compared to the online algorithms with no information about the future.

The advice complexity model for online algorithms is a more general framework under which the "no knowledge assumption" behind online algorithms is relaxed, and the algorithm receives some bits of *advice* about the future requests. The advice can be any information about the input sequence and is generated by an offline oracle which has unbounded computational power. Provided with the appropriate advice, the online algorithms are expected to achieve improved competitive ratios. The advice model has received significant attention since its introduction [8, 17, 13, 7, 18, 20, 9, 4, 11, 15, 19, 6, 5, 21].

In this paper, we study the advice complexity of the online bin packing problem. Our interest in studying the problem under this setting is mostly theoretical. Nevertheless, in many practical scenarios, it can be justified to allow a fast offline oracle to take a "quick look" at the input sequence and send some advice to the online algorithm. For example, it may be possible to take a quick look and count the number of items which are larger than 1/2 and smaller than 2/3 of the bin capacity. We show that this form of advice can be used to achieve an algorithm which outperforms all online algorithms.

1.1 Model

In the last few years, slightly different models of advice complexity have been proposed for online problems. All these models assume that there is an offline oracle with infinite computational power, which provides the online algorithm with some bits of advice. How these bits of advice are given to the algorithm is the source of difference between the models. In the first model, presented in [12], an online algorithm poses a series of questions which are answered by the offline oracle in *blocks of answers*. The total size of the answers, measured in the number of bits, defines the advice complexity. The problem with this model is that a lot of information can be encoded in the individual length of each block. To address this issue, another model is proposed in [13] which assumes that online algorithms receive a fixed number of bits of advice per request. We call this model the *advice-with-request model*. This model is studied for problems, such as metrical task systems and k-server, and the results tend to use at least a constant number of bits of advice per request [13, 20]. Nevertheless, there are many online problems for which a sublinear and even a constant number of bits of advice in total is sufficient to achieve good competitive ratios. However, under the advice-with-request

model, the possibility of sending a sublinear number of advice bits to the algorithm is not well defined. In [8, 7] another model of advice complexity is presented which assumes that the online algorithm has access to an *advice tape*, written by the offline oracle. At any time step, the algorithm may refer to the tape and read any number of advice bits. The advice complexity is the number of bits on the tape accessed by the algorithm. We refer to this model as *advice-on-tape model*. Since its introduction, the advice-on-tape model has been used to analyze the advice complexity of many online problems including paging [8, 17, 18], disjoint path allocation [8], job shop scheduling [8, 18], k-server [7, 20], knapsack [9], various coloring problems [4, 15, 5, 21], set cover [19, 6], maximum clique [6], and graph exploration [11].

Under the advice-on-tape model, we require a mechanism to infer how many bits of advice the algorithm should read at each time step. This could be implicitly derived during the execution of the algorithm or explicitly encoded in the advice string itself. For example, we may use a *self-delimited* encoding as used in [7], in which the value of a non-negative integer X is encoded by writing the value of $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil$ in unary (a string of 1's followed by a zero), the value of $\lceil \log(X+1) \rceil$ in binary [1], and the value of X in binary. These codes respectively require $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil + 1$, $\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil$, and $\lceil \log(X+1) \rceil$ bits. Thus, the self-delimited encoding of X requires

$$e(X) = \lceil \log(X+1) \rceil + 2\lceil \log(\lceil \log(X+1) \rceil + 1) \rceil + 1$$

bits. The existence of self-delimited encodings at the beginning of the tape usually adds a lower-order term to the number of advice bits required by an algorithm.

Regarding notation, we use $\mathbb{A}(\sigma)$ to denote the costs of $\mathbb{A}$ for packing a request sequence σ. When σ follows from the context, we simply use $\mathbb{A}$ to denote this cost. We use similar notation for all algorithms, including OPT.

We consider the bin packing problem under the advice-on-tape model, which is formally defined as follows, based on the definition of the advice model in [7]:

▶ **Definition 1.** In the *online bin packing problem with advice*, the input is a sequence of items $\sigma = \langle x_1, \ldots, x_n \rangle$, revealed to the algorithm in an online manner ($0 < x_i \leq 1$). The goal is to pack these items in the minimum number of bins of unit size. At time step t, an online algorithm should pack item x_t into a bin. The decision of the algorithm to select the target bin is a function of $\Phi, x_1, \ldots, x_{t-1}$, where Φ is the content of the advice tape. An algorithm $\mathbb{A}$ is *c-competitive with advice complexity $s(n)$* if there exists a constant c_0 such that, for all n and for all input sequences σ of length at most n, there exists some advice Φ such that $\mathbb{A}(\sigma) \leq c\,\text{OPT}(\sigma) + c_0$, and at most the first $s(n)$ bits of Φ have been accessed by the algorithm. If $c = 1$ and $c_0 = 0$, then $\mathbb{A}$ is *optimal*.

1.2 Contribution

We answer different questions about the advice complexity of the online bin packing problem. First, we study how many bits of advice are required to achieve an optimal solution. We consider two different settings of the problem. When there is no restriction on the number of distinct items or their sizes, we present the easy result that $n\lceil \log \text{OPT}(\sigma) \rceil$ bits of advice are sufficient to achieve an optimal solution, where $\text{OPT}(\sigma)$ is the number of bins in an optimal packing. We also prove that at least $(n - 2\,\text{OPT}(\sigma)) \log \text{OPT}(\sigma)$ bits of advice are required to achieve an optimal solution.

---

[1] In this paper we use $\log n$ to denote $\log_2(n)$.

When there are m distinct items in the sequence, we prove that at least $(m - 3) \log n - 2m \log m$ bits of advice are required to achieve an optimal solution. If m is a constant, there is a linear time online algorithm that receives $m \log n + o(\log n)$ bits of advice and achieves an optimal solution. We also show that, even if m is not a constant, there is a polynomial time online algorithm that receives $m\lceil \log(n + 1) \rceil + o(\log n)$ bits of advice and achieves a packing with $(1 + \varepsilon) \operatorname{OPT}(\sigma) + 1$ bins.

We also study a relevant question that asks how many bits of advice are required to perform strictly better than all online algorithms. We bound this by providing an algorithm which receives $\log n + o(\log n)$ bits of advice and achieves a competitive ratio of $3/2$. Recall that any online bin packing algorithm has a competitive ratio of at least 1.54037 [3]. Hence, our algorithm outperforms all online algorithms.

Moreover, we introduce an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$, for any fixed value of $\varepsilon > 0$. We also prove a lower bound that implies that a linear number of bits of advice are required to achieve a competitive ratio of $9/8 - \delta$ for any fixed value of $\delta > 0$.

Due to space restrictions, many proofs have been removed. They will appear in the long version of the paper.

2   Optimal Algorithms with Advice

In this section we study the amount of advice required to achieve an optimal solution. We first investigate the theoretical setting in which there is no restriction on the number of distinct items or on their sizes. We observe that there is a simple algorithm that receives $n\lceil \log \operatorname{OPT}(\sigma) \rceil$ bits of advice and achieves an optimal solution. Such an algorithm basically reads $\lceil \log \operatorname{OPT}(\sigma) \rceil$ bits for each item, encoding the index of the bin that includes the item in an optimal packing. We show that the upper bound given by this algorithm is tight up to lower order terms, when $n - 2 \operatorname{OPT}(\sigma) \in \Theta(n)$.

▶ **Theorem 2.** *To achieve an optimal packing for a sequence of size n and optimal cost $\operatorname{OPT}(\sigma)$, it is sufficient to receive $n\lceil \log \operatorname{OPT}(\sigma) \rceil$ bits of advice. Moreover, any deterministic online algorithm requires at least $(n - 2 \operatorname{OPT}(\sigma)) \log \operatorname{OPT}(\sigma)$ bits of advice to achieve an optimal packing.*

Next, we consider a more realistic scenario where there are $m \in o(n)$ distinct items and the values of these items are known to the algorithm. Assume that the advice tape specifies the number of items of each size. If we are not concerned about the running time of the online algorithm, there is enough information to obtain an optimal solution. If we are concerned, we can use known results for solving the offline problem [2, 14, 23] to obtain the following:

▶ **Theorem 3.** *Consider the online bin packing problem in which there are m distinct items. If m is a constant, there is a (linear time) optimal online algorithm that receives $m \log n + o(\log n)$ bits of advice. If m is not a constant, there is a (polynomial time) online algorithm that reads $m\lceil \log(n + 1) \rceil + o(\log n)$ bits of advice and achieves an almost optimal packing with at most $(1 + \varepsilon) \operatorname{OPT}(\sigma) + 1$ bins, for any small but constant value of ε.*

We show that the above upper bound is asymptotically tight.

▶ **Theorem 4.** *At least $(m - 3) \log n - 2m \log m$ bits of advice are required to achieve an optimal solution for the online bin packing problem on sequences of length n with m distinct items, each of size at least $\frac{1}{2m}$.*

3   An Algorithm with Sublinear Advice

In what follows we introduce an algorithm that receives $\log n + o(\log n)$ bits of advice and achieves a competitive ratio of $\frac{3}{2}$, for any instance of the online bin packing problem. An offline oracle can compute and write the advice on the tape in linear time, and the online algorithm runs as fast as First-Fit. Thus, the algorithm might be applied in practical scenarios in which it is allowed to have a "quick look" at the input sequence.

We call items *tiny*, *small*, *medium*, and *large* if their sizes lie in the intervals $(0, 1/3]$, $(1/3, 1/2]$, $(1/2, 2/3]$, and $(2/3, 1]$, respectively. The advice that the algorithm receives is the number of medium items, which we denote by α.

The algorithm reads the advice tape, obtains α, opens α bins, called *critical bins*, and reserves $2/3$ of the space in each of them. This reserved space will be used to pack a medium item in each of the critical bins, and these bins have a *virtual level* of size $2/3$ at the beginning. All other bins have virtual level zero when they are opened. The algorithm serves an item x in the following manner:

- If x is a large item, open a new bin for it. Set the virtual level to its size.
- If x is a medium item, put it in the reserved space of a critical bin B. Update the virtual level to the actual level. (B will not have any reserved space now.)
- If x is small or tiny, use the First Fit (FF) strategy to put it into any of the open bins, based on virtual levels (open a new bin if required). Add the size of the item to the virtual level.

Note that the critical bins appear first in the ordering maintained by the algorithm as they are opened before other bins.

▶ **Theorem 5.** *There is an online algorithm which receives* $\log n + o(\log n)$ *bits of advice and has cost* $3/2 \operatorname{OPT}(\sigma) + 3$ *for serving any sequence* σ *of size* n.

Proof. We prove that the algorithm described above has the desired property. The value of α is encoded in $X = \lceil \log(n + 1) \rceil$ bits of advice. In order to read this properly from the tape, the algorithm needs to know the value of X. This can be done by adding the self-delimited encoding of X in $e(X) = \lceil \log X \rceil + 2\lceil \log \log(X) \rceil + 2$ bits at the beginning of the tape. Consequently the number of advice bits used by the algorithm is $X + O(\log X)$, which is $\log n + o(\log n)$ as stated by the theorem.

Consider the final packing of the algorithm for serving a sequence σ. There are two cases. In the first case, there is a critical bin B so that no other item, except a medium item, is packed in it. Since all tiny items are smaller than $1/3$ and can fit in B, all the non-critical bins that are opened after B include small and large items only. More precisely, they include either a single large item or two small items (except the last one which might have a single small item). Let L, M, and S denote the number of large, medium, and small items. The cost of the algorithm is at most $L + M + S/2 + 1$. Now, if $S \leq M$, this would be at most $L + 3/2M + 1$. Since $L + M$ is a lower bound on the cost of OPT, the cost of the algorithm is at most $3/2 \operatorname{OPT}(\sigma) + 1$ and we are done. If $S > M$, OPT should open $L + M$ bins for large and medium items, and in the best case, it packs M small items together with medium ones. For the other $S - M$ bins, OPT has to open at least $(S - M)/2$ bins. Hence the cost of OPT is at least $L + M + (S - M)/2 = L + M/2 + S/2$, and we have $3/2 \operatorname{OPT}(\sigma) \geq 3L/2 + 3M/4 + 3S/4 > L + M + S/2$. Thus, the cost of the algorithm is at most $3/2 \operatorname{OPT}(\sigma) + 1$.

In the second case, we assume that all critical bins include another item in addition to the medium item. We claim that at the end of serving a sequence all bins, except possibly

two, have level at least $2/3$. First, we verify this for non-critical bins (bins without medium items). If a non-critical bin is opened by a large item, it clearly has level higher than $2/3$. All other non-critical bins only include items of size at most $1/2$. Hence, these bins, except possibly the last one, include at least two items. Among the non-critical bins that include two items, consider two bins b_i and b_j ($i < j$) that have levels smaller than $2/3$. Since b_j contains at least two items, at least one of them has size smaller than $1/3$. This item could fit in b_i by the FF property. We conclude that all non-critical bins, except possibly two, have level at least $2/3$. Now, suppose two critical bins b_i and b_j have levels smaller than $2/3$. Consider the first non-medium item x which is packed in b_j (in the second case, such an item exists). Since a medium item is packed in the bin, x should be either tiny or small. If x is small, then the level of b_j is at least $1/2 + 1/3$, which contradicts the level of b_j being smaller than $2/3$. Similarly, x cannot be a tiny item of size larger than $1/6$ (since $1/2 + 1/6 \geq 2/3$). Hence, x is a tiny item of size at most $1/6$. This implies that at the time the online algorithm packs x, bin b_i has a virtual level of at least $5/6$. The virtual level is at most $1/6$ larger than the actual level (the final level). Hence, the actual level of b_i is at least $5/6 - 1/6 = 2/3$. We conclude that at most one critical bin has level smaller than $2/3$. To summarize, at most three bins have level smaller than $2/3$. Hence, the cost of the algorithm is at most $3/2 \, \text{OPT}(\sigma) + 3$. ◀

4 An Algorithm with Linear Advice

In this section, we present an algorithm that receives $2n + o(n)$ bits of advice and achieves a competitive ratio of $4/3 + \varepsilon$ for any sequence of size n, and arbitrarily small (but constant) values of ε. Consider an algorithm that receives an *approximate size* for each sufficiently large item x encoded using k bits. The approximate size of x would be larger than its *actual size* by at most an additive term of $1/2^k$. The algorithm can optimally pack items by their approximate sizes and achieve an *approximate packing* which includes a reserved space of size $x + \varepsilon$ ($\varepsilon \leq 1/2^k$) for each item. Precisely, for each sufficiently large item x, the approximate packing includes a reserved space of size $x + \varepsilon$ ($\varepsilon \leq 1/2^k$) for x. This enables the algorithm to place x in the reserved space for it in the approximate packing. Smaller items are treated differently and the algorithm does not reserve any space for them. In the reminder of this section, we elaborate this idea to achieve a $4/3$-competitive algorithm.

Notice that the cost of an approximate packing can be as large as $\frac{3}{2}$ times the cost of OPT. To see that, consider a sequence which is a permutation of $\langle \frac{1}{2} + \varepsilon_1, \frac{1}{2} - \varepsilon_1, \frac{1}{2} + \varepsilon_2, \frac{1}{2} - \varepsilon_2, \ldots, \frac{1}{2} + \varepsilon_{n/2}, \frac{1}{2} - \varepsilon_{n/2} \rangle$, where $\varepsilon_i < 1/2^n (1 \leq i \leq n/2)$. Since OPT packs all bins tightly, an increase in the sizes of items by a constant (small) ε results in opening a new bin for each two bins OPT uses. Hence the cost of the optimal approximate packing can be as bad as $\frac{3}{2}$ OPT. This example suggests that using approximate packings is not good for the bins in which a small number of large items are tightly packed. To address this issue we divide the bins of OPT into two groups:

▶ **Definition 6.** Consider an optimal packing of a sequence σ. Given a small parameter $\varepsilon' < 1/60$, define *good bins* to be those where the total size of the items smaller than $1/4$ in the bin is at least $5\varepsilon'$. Define all other bins to be *bad bins*.

A part of the advice received for each item x indicates if x is packed by OPT in a good bin or in a bad bin. This enables us to treat items packed in these two groups separately.

▶ **Lemma 7.** *Consider sequences for which all bins in the optimal packing are good (as defined above). There is an online algorithm that receives $o(n)$ bits of advice and achieves a competitive ratio of $4/3$.*

Proof. Call an item *small* if it is smaller than or equal to $1/6$ and *large* otherwise. The advice bits define the approximate sizes of all large items with a precision of ε'. The amount of advice will be roughly $2^{1/\varepsilon'} \log n$ which is $o(n)$ for constant values of ε'. The online algorithm $\mathbb{A}$ can build the optimal approximate packing of large items. In such a packing, there is a reserved space of size at most $x + \varepsilon'$ for any large item of size x. The algorithm considers this packing as a partial packing and initializes the level of each bin to be the total sizes of approximated items in that bin. For packing an item x, if x is large, $\mathbb{A}$ packs it in the space reserved for it in the approximate packing. It also updates the level of the bin to reflect the actual size of x. If x is small, $\mathbb{A}$ simply applies the First-Fit strategy to pack x in a bin of the partial packing (and opens a new bin for it if necessary). We prove that $\mathbb{A}$ is $4/3$-competitive. In the final packing by $\mathbb{A}$, call a bin "red" if all items packed in it are small items and call it "blue" otherwise (the blue bins constitute the approximated packing at the beginning). There are two cases to consider.

In the first case, there is no red bin in the final packing of $\mathbb{A}$, i.e., all small items fit in the remaining space of the bins in the approximate packing of large items. Let σ' be a copy of the input sequence in which the sizes of large items are approximated, i.e., increased by at most ε'; also let X be the number of bins for the optimal packing of σ'. Since there is no red bin in the final packing of $\mathbb{A}$, the cost of $\mathbb{A}$ is equal to X. Consider the optimal packing of the actual input sequence σ. Since all bins are good, one can transfer a subset of items to provide an available space of size at least $5\varepsilon'$ in each bin. After such a transfer, we can increase the sizes of large items to their approximate sizes. Since there are at most 5 large items in each bin and also available space of size at least $5\varepsilon'$, the packing constructed this way is a valid packing for the sequence σ'. Since the size of the transferred items for each bin is at most $1/4$, the transferred items from each group of four bins can fit in one new bin. Consequently the number of bins in the new packing is at most $5/4 \, \text{OPT}(\sigma)$. We know that the final packing by $\mathbb{A}$ is the optimal packing for σ' (with cost X), and in particular not worse than the packing constructed above. Hence, the cost of $\mathbb{A}$ is not more than $5/4 \, \text{OPT}(\sigma)$.

In the second case, there is at least one red bin in the final packing of $\mathbb{A}$. We claim that all bins in the final packing of $\mathbb{A}$, except possibly the last, have levels larger than $3/4$. The claim obviously holds for the red bins since the levels of all these bins (excluding the last one) are larger than $5/6$. Moreover, since there is a bin which is opened by a small item, all blue bins have levels larger than $5/6$, i.e., the total size of packed items and reserved space for the large items is larger than $5/6$. Since there are at most 5 large items in each bin, the actual level of each bin in the final packing of $\mathbb{A}$ is at least $5/6 - 5\varepsilon'$, which is not smaller than $3/4$ for $\varepsilon' \leq 1/60$. So, all bins, except possibly one, have levels larger than $3/4$. Consequently, the algorithm is $4/3$-competitive. ◀

It remains to address how to deal with bad bins. The next three lemmas do this.

▶ **Lemma 8.** *Consider sequences for which all bins in the optimal packing include precisely two items. There is an algorithm that receives 1 bit of advice per request and achieves an optimal packing.*

▶ **Lemma 9.** *Consider a sequence σ for which all items have sizes larger than $1/4$ and for which each bin in* OPT*'s packing includes precisely three items. The cost of the Harmonic algorithm is at most $4/3 \, \text{OPT}(\sigma) + 3$ for serving such a sequence.*

▶ **Lemma 10.** *Consider a sequence σ for which all bins in the optimal packing are bad bins (as defined earlier). There is an algorithm that receives two bits of advice for each request, and opens at most $(4/3 + \frac{5\varepsilon'}{1 - 5\varepsilon'}) \, \text{OPT}(\sigma) + 3$ bins.*

Proof. By the definition of bad bins, for any bin in the optimal packing, all items are either smaller than $5\varepsilon'$ or larger than $1/4$. We call the former group of items *tiny* items and pack them separately using the FF strategy. We refer to other items as *normal items.* Consider an offline packing P which is the same as OPT's packing, except that all tiny items are removed from their bins and packed separately in new bins using the FF strategy. This implies that the cost of P is larger than $\mathrm{OPT}(\sigma)$ by a multiplicative factor of at most $1 + \frac{5\varepsilon'}{1-5\varepsilon'}$. Let Q be the optimal packing for normal items. Since all normal items are larger than $1/4$, each bin of Q contains at most three items. We say a bin of Q has type i ($i \in \{1, 2, 3\}$), if it contains i normal items. Similarly, we say an item x has type i if it is packed in a type i bin. All items in type 3 bins have sizes smaller than $1/2$ (otherwise one will have size at most $1/4$ which contradicts the assumption). Moreover, the sizes of the items in all type 1 bins (except possibly the last one) are larger than $1/2$ (otherwise a better packing is achieved by pairing two of them). With two bits of advice, we can detect the type of an item as follows: Let b denote the two bits of advice with item x. If b is "01" and $x > 1/2$, then x has type 1; if b is "01" and $x \leq 1/2$, then x has type 3; and if b is "10" or b is "11", then x has type 2. Note that the code "00" is not used at this point (this is used later on), and the use of "10" and "11" is still to be detailed.

Let X_i denote the number of bins of type i ($1 \leq i \leq 3$). Hence, the cost of Q is $X_1+X_2+X_3$, and consequently the cost of P is at least $X_1 + X_2 + X_3 + X'$, where X' is the number of bins filled by tiny items. Consider an algorithm $\mathbb{A}$ that performs as follows. If an item x has type 1, $\mathbb{A}$ simply opens a new bin for x. If x has type 2, $\mathbb{A}$ applies the strategy of Lemma 8 to place it in one of the bins maintained for items of type 2. Recall that the advice in this case is either "10" or "11", so the second bit provides the advice required by Lemma 8. If x has type 3, $\mathbb{A}$ applies the Harmonic strategy to pack the item in a set of bins maintained for type 3 items. By Lemma 9, the cost of $\mathbb{A}$ for these items is at most $4/3X_3+3$. Finally, $\mathbb{A}$ uses the FF strategy to pack tiny items in separate bins. Consequently, the cost of the algorithm is at most $X_1 + X_2 + 4/3X_3 + X' + 3 \leq (1 + \frac{5\varepsilon'}{1-5\varepsilon'})\,\mathrm{OPT}(\sigma) + X_3/3 + 3 \leq (4/3 + \frac{5\varepsilon'}{1-5\varepsilon'})\,\mathrm{OPT}(\sigma) + 3.$ ◀

Provided with the above results, we arrive at the following result:

▶ **Theorem 11.** *There is an online algorithm which receives two bits of advice per request, plus an additive lower order term, and achieves a competitive ratio of $4/3+\varepsilon$, for any positive value of ε.*

Proof. Define ε' to be $\frac{11\varepsilon}{60}$. For $\varepsilon < 1/11$, we have $\varepsilon' < 1/60$. Moreover, we have $\frac{5\varepsilon'}{1-5\varepsilon'} \leq \frac{5\varepsilon'}{1-1/12} = \frac{60\varepsilon'}{11} = \varepsilon$. In an optimal packing, divide bins into good and bad bins using Definition 6. Also, let Gd and Bd respectively denote the number of good and bad bins. Use advice bits to distinguish items which are packed in good and bad bins, and pack them in separate lists of bins. More precisely, let the two bits of advice for an item x be "00" if it is packed by OPT in a good bin, and apply Lemma 7 to pack these items in at most $4/3Gd$ bins. Similarly, apply Lemma 10 to pack items from bad bins in at most $(4/3 + \frac{5\varepsilon'}{1-5\varepsilon'})Bd + 3 \leq (4/3 + \varepsilon)Bd + 3$ bins, using bits of advice of the form "01", "10", or "11", as discussed in the proof of Lemma 10. Consequently, the cost of the algorithm will be at most $4/3Gd + (4/3 + \varepsilon)Bd + 3 \leq (4/3 + \varepsilon)\,\mathrm{OPT}(\sigma) + 3.$ ◀

5 A Lower Bound for Linear Advice

The *GMP problem* [13] and the *String Guessing Problem* [6] both contain a core special case of guessing a binary sequence. We use their results to show that an online algorithm needs a linear number of bits of advice to achieve a competitive ratio better than $9/8$ for bin packing.

▶ **Definition 12** ([13, 6]). The *Binary String Guessing Problem with known history (2-SGKH)* is the following online problem. The input $I = (n, \sigma = \langle x_1, x_2, \ldots, x_n \rangle)$ consists of n items that are either "0" or "1" and that are revealed one by one. For each item x_t, the online algorithm $\mathbb{A}$ must guess if it is a "0" or a "1". After the algorithm has made a guess, the value of x_t is revealed to the algorithm.

▶ **Lemma 13** ([6]). *On any input of length n, any deterministic algorithm for 2-SGKH that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha)n$ bits of advice.*

Since the number of bits needed to express the number of "0"s in the input is at most $\lceil \log(n + 1) \rceil \leq \log n + 1$, and this number can be given as advice by an oracle, if it is not given to the algorithm otherwise, we easily obtain the following lemma. Recall that the definition of e, the length of the encoding function, is given in Section 1.1.

▶ **Lemma 14.** *Consider instances of size n of the 2-SGKH problem in which the number of "0"s is given to the algorithm as part of the input. For these instances, any deterministic algorithm that is guaranteed to guess correctly on more than αn bits, for $1/2 \leq \alpha < 1$, needs to read at least $(1 + (1 - \alpha) \log(1 - \alpha) + \alpha \log \alpha)n - e(n)$ bits of advice.*

In order to relate the Binary String Guessing Problem to the online bin packing problem, we introduce another problem called the Binary Separation Problem.

▶ **Definition 15.** The *Binary Separation Problem* is the following online problem. The input $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n \rangle)$ consists of $n = n_1 + n_2$ positive values which are revealed one by one. There is a fixed partitioning of the set of items into a subset of n_1 *large* items and a subset of n_2 *small* items, so that all large items are larger than all small items. Upon receiving an item y_i, an online algorithm for the problem must guess if y belongs to the set of small or large items. After the algorithm has made a guess, it is revealed to the algorithm whether y_i actually belongs to class of small or large items.

We provide reductions from the modified Binary String Guessing Problem to the Binary Separation Problem, and from the Binary Separation Problem to the online bin packing problem. In order to reduce a problem P_1 to another problem P_2, given an instance of P_1 defined by a sequence σ_1 and a set of parameters η_1 (such as the length of σ_1 or the number of "0"s in it), we create an instance of P_2 which is defined by a sequence σ_2 and also a set of parameters η_2. In our reductions, we assume η_2 is derived from η_1, and since σ_1 is revealed in an online manner, σ_2 is created in an online manner by looking only at η_1 and the revealed items of σ_1.

▶ **Lemma 16.** *Assume that there is an online algorithm that solves the Binary Separation Problem on sequences of length n with $b(n)$ bits of advice, and makes at most $r(n)$ mistakes. Then there is also an algorithm that solves the Binary String Guessing Problem on sequences of length n, assuming the number of "0"s is given as a part of input, so that the algorithm receives $b(n)$ bits of advice and makes at most $r(n)$ errors.*

Proof. We assume that we have an algorithm BSA that solves the Binary Separation Problem under the conditions of the lemma statement. Using that algorithm, we define the number n_1 of large items to be the number of "0"s in the instance of the Binary String Guessing Problem. Then, we implement our algorithm BSGA for the Binary String Guessing Problem as outlined in Algorithm 1, which defines the reduction. This BSGA implementation, defined in Algorithm 1, functions as an adversary for BSA, e.g., in Line 4, BSGA gives BSA its next

---

Algorithm 1 Implementing Binary String Guessing via Binary Separation.

> The Binary Guessing algorithm knows the number of "0"s (n_1) and passes it as a parameter (the number of large items) to the Binary Separation algorithm

1: small = 0; large = 1
2: **repeat**
3:     mid = (large − small) / 2
4:     class_guess = SeparationAlgorithm.ClassifyThis(mid)
5:     **if** class_guess = "large" **then**
6:         bit_guess = 0
7:     **else**
8:         bit_guess = 1
9:     actual_bit = Guess(bit_guess) {The actual value is received after guessing (2-SGKH).}

10:     **if** actual_bit = 0 **then**
11:         large = mid {We let "large" be the correct decision.}
12:     **else**
13:         small = mid {We let "small" be the correct decision.}
14: **until** end of sequence

---

request. Notice that we ensure that the BSGA makes a correct guess if and only if BSA makes a correct guess. The advice tape is filled with bits of advice for this combined algorithm. The BSGA uses the BSA as a sub-routine, but all the questions are effectively coming from the BSA.

The set-up, reminiscent of binary search, is carried out as specified in the algorithm with the purpose of ensuring that when the BSA is informed of the actual class of the item it considered, no result can contradict information already obtained. Specifically, the next item for the BSA to consider is always in between the largest item which has previously been deemed "small" and the smallest item which has previously been deemed "large". The fact that we give the middle item from that interval is unimportant; any value chosen from the open interval would work.   ◀

Now, we prove that if we can solve a special case of the bin packing problem, we can also solve the Binary Separation Problem.

▶ **Lemma 17.** *Consider the bin packing problem on sequences of length $2n$ for which* OPT *opens n bins. Assume that there is an online algorithm $\mathbb{A}$ that solves the problem on these instances with $b(n)$ bits of advice and opens at most $n + r(n)/4$ bins. Then there is also an algorithm* BSA *that solves the Binary Separation Problem on sequences of length n with $b(n)$ bits of advice and makes at most $r(n)$ errors.*

Proof. In the reduction, we encode requests for the BSA as items for bin packing. Assume we are given an instance $I = (n_1, \sigma = \langle y_1, y_2, \ldots, y_n \rangle)$ of the Binary Separation problem, in which n_1 is the number of large items ($n_1 + n_2 = n$), and the values of y_ts are revealed in an online manner ($1 \leq t \leq n$). We create an instance of the bin packing problem which has length $2n$. Algorithm 2 shows the details of the reduction. The bin packing sequence starts with n_1 items of size $\frac{1}{2} + \varepsilon_{min}$ (in Algorithm 2, the variable "NumberOfLargeItems" is n_1 from the Binary Separation Problem). Any algorithm needs to open a bin for each of these n_1 items. We create the next n items in an online manner, so that we can use the result of their packing to guess the requests for the Binary Separation Problem. Let $\tau = y_t$

$(1 \le t \le n)$ be a requested item of the Binary Separation Problem; we ask the bin packing algorithm to pack an item whose size is an increasing function of τ, and slightly less than $\frac{1}{2}$. Depending on the decision of the bin packing algorithm for opening a new bin or placing the item in one of the existing bins, we decide the type of τ as being consecutively small or large. The last n_2 items of the bin packing instance are defined as complements of the items in the bin packing instance associated with small items in the binary separation instance (the complement of item x is $1 - x$). We do not need to give the last items complementing the small items in order to implement the algorithm, but we need them for the proof of the quality of the correspondence that we are proving.

Call an item in the bin packing sequence "large" if it is associated with large items in the Binary Separation Problem, and "small" otherwise. For the bin packing sequence produced by the reduction, an optimal algorithm pairs each of the large items with one of the first n_1 items (those with size $\frac{1}{2} + \varepsilon_{min}$), placing them in the first n_1 bins. OPT pairs the small items with their complements, starting one of the next n_2 bins with each of these small items. Hence, the cost of an optimal algorithm is $n_1 + n_2 = n$. The values ε_{min} and ε_{max} in Algorithm 2 must be small enough so that no more than two of any of the items given in the algorithm can fit together in a bin. No other restriction is necessary.

We claim that each extra bin used by the bin packing algorithm, but not by OPT, results in at most four mistakes made by the derived algorithm on the given instance of the Binary Separation Problem. Consider an extra bin in the final packing of $\mathbb{A}$. This bin is opened by a large item which is incorrectly guessed as being small (bins which are opened by small items also appear in OPT's packing). Note that large items do not fit in the same bins as complements of small items. The extra bin has enough space for another large item. Moreover, there are at most two small items which are incorrectly guessed as being large and placed in the space dedicated to the large items of the extra bin. Hence, there is an overhead of at least one for four mistakes. To summarize, $\mathbb{A}$ has to decide if a given item is small or large and performs accordingly, and it pays a cost of at least $1/4$ for each incorrect decision. If $\mathbb{A}$ opens at most $n + r(n)/4$ bins, the algorithm derived from $\mathbb{A}$ for the Binary Separation Problem makes at most $r(n)$ mistakes. ◀

▶ **Theorem 18.** *Consider the online bin packing problem on sequences of length n. To achieve a competitive ratio of c ($1 < c < 9/8$), an online algorithm needs to receive at least $(n(1 + (4c-4)\log(4c-4) + (5-4c)\log(5-4c)) - (\lceil \log(n+1) \rceil + 2\lceil \log(\lceil \log(n+1) \rceil + 1) \rceil + 1))/2$ bits of advice.*

Proof. Consider a bin packing algorithm $\mathbb{A}$ that receives $b(n)$ bits of advice and achieves a competitive ratio of c. This algorithm opens at most $(c-1)\,\text{OPT}(\sigma)$ bins more than OPT, so when $\text{OPT}(\sigma) = n/2$, it opens at most $(c-1)n/2$ more bins. By Lemma 17, the existence of such an algorithm implies that there is an algorithm $\mathbb{A}$ that solves the Binary Separation Problem on sequences of length $n/2$ with b bits of advice and makes at most $2(c-1)n$ errors. By Lemma 16, this implies that there is an algorithm $\mathbb{B}$ that solves the Binary String Guessing Problem on sequences of length $n/2$ with b bits of advice and makes at most $2(c-1)n$ mistakes, i.e., it correctly guesses the other $n/2 - 2(c-1)n = (5-4c)n/2$ items. Let $\alpha = 5 - 4c$, and note that α is in the range $[1/2, 1)$ when c is in the range $(1, 9/8)$. Lemma 14 implies that in order to correctly guess more than $\alpha n/2$ of the items in the binary sequence, we must have $b(n)$ larger than or equal to $((1 + (1-\alpha)\log(1-\alpha) + \alpha\log\alpha)n - e(n))/2$. Replacing α with $5 - 4c$ completes the proof. ◀

Thus, to obtain a competitive ratio strictly better than $9/8$, a linear number of bits of advice is required. For example, to achieve a competitive ratio of $17/16$, at least $0.188n$ bits of advice are required asymptotically.

Algorithm 2 Implementing Binary Separation via Special Case Bin Packing.

1: Choose ε_{min} and ε_{max} so that $0 < \varepsilon_{min} < \varepsilon_{max} < \frac{1}{6}$
2: Choose a decreasing function $f \colon \mathbb{R} \to (\varepsilon_{min}..\varepsilon_{max})$
3: **for** i = 1 to NumberOfLargeItems **do**
4:     BinPacking.Treat($\frac{1}{2} + \varepsilon_{min}$) {The decision can only be to open a bin.}
5: **repeat**
6:     Let τ be the next request
7:     decision = BinPacking.Treat($\frac{1}{2} - f(\tau)$)
8:     **if** decision = "packed with an $\frac{1}{2} + \varepsilon_{min}$ item" **then**
9:         class_guess = "large"
10:     **else**
11:         class_guess = "small"
       actual_class = Guess(class_guess)
12:     **if** actual_class = "small" **then**
13:         SmallItems.append($\frac{1}{2} - f(\tau)$) {Collecting small items for later.}
14: **until** end of request sequence
15: **for** i = 1 to len(SmallItems) **do**
16:     BinPacking.Treat(1 − SmallItems[i]) {The decision is not used.}

▶ **Corollary 19.** *Consider the bin packing problem for packing sequences of length n. To achieve a competitive ratio of $9/8 - \delta$, in which δ is a small, but fixed positive number, an online algorithm needs to receive $\Omega(n)$ bits of advice.*

6   Concluding Remarks

We conjecture that a sublinear number of bits of advice is enough to achieve competitive ratios smaller than $4/3$. Note that our results imply that we cannot hope for ratios smaller than $9/8$ with sublinear advice.

References

1   E. Asgeirsson, U. Ayesta, E. Coffman, J. Etra, P. Momcilovic, D. Phillips, V. Vokhshoori, Z. Wang, and J. Wolfe. Closed on-line bin packing. *Acta Cybernet.*, 15(3):361–367, 2002.

2   J. Baewicz and K. Ecker. A linear time algorithm for restricted bin packing and scheduling problems. *Oper. Res. Lett.*, 2(2):80–83, 1983.

3   János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoret. Comput. Sci.*, 440–441:1–13, 2012.

4   Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, and Lucia Keller. On-line coloring of bipartite graphs with and without advice. In *COCOON '12*, volume 7434 of *LNCS*, pages 519–530, 2012.

5   Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovic, Sacha Krug, and Björn Steffen. On the advice complexity of the online $L(2,1)$-coloring problem on paths and cycles. In *COCOON '13*, volume 7936 of *LNCS*, pages 53–64, 2013.

6   Hans-Joachim Böckenhauer, Juraj Hromkovic, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. In *COCOON '13*, volume 7936 of *LNCS*, pages 493–505, 2013.

7   Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, and Richard Královič. On advice complexity of the k-server problem. In *ICALP '1s1*, volume 6755 of *LNCS*, pages 207–218, 2011.

8   Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *ISAAC '09*, volume 5878 of *LNCS*, pages 331–340, 2009.

9   Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, and Peter Rossmanith. On the advice complexity of the knapsack problem. In *LATIN '12*, volume 7256 of *LNCS*, pages 61–72, 2012.

10   Edward G. Coffman, Michael R. Garey, and David S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

11   Stefan Dobrev, Rastislav Královic, and Euripides Markou. Online graph exploration with advice. In *SIROCCO '12*, volume 7355 of *LNCS*, pages 267–278, 2012.

12   Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Inform. Theor. Appl.*, 43(3):585–613, 2009.

13   Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoret. Comput. Sci.*, 412(24):2642 – 2656, 2011.

14   W. Fernandez de la Vega and G. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

15   Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *LATA '12*, volume 7183 of *LNCS*, pages 228–239, 2012.

16   Edward F. Grove. Online binpacking with lookahead. In *SODA '95*, pages 430–436, 1995.

17   Juraj Hromkovič, Rastislav Královič, and Richard Královič. Information complexity of online problems. In *MFCS '10*, volume 6281 of *LNCS*, pages 24–36, 2010.

18   D. Komm and R. Královič. Advice complexity and barely random algorithms. *RAIRO Inform. Theor. Appl.*, 45(2):249–267, 2011.

19   Dennis Komm, Richard Královic, and Tobias Mömke. On the advice complexity of the set cover problem. In *CSR '12*, volume 7353 of *LNCS*, pages 241–252, 2012.

20   Marc P. Renault and Adi Rosén. On online algorithms with advice for the k-server problem. In *WAOA '11*, volume 7164 of *LNCS*, pages 198–210, 2011.

21   Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In *CIAC '13*, volume 7878 of *LNCS*, pages 345–357, 2013.

22   Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49:640–671, 2002.

23   Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2004.