Better Algorithms for Online Bin Stretching

Martin Böhm[1,⋆], Jiří Sgall[1,⋆], Rob van Stee[2] and Pavel Veselý[1,⋆]

[1] Computer Science Institute of Charles University, Prague, Czech Republic.
{bohm,sgall,vesely}@iuuk.mff.cuni.cz.
[2] Department of Computer Science, University of Leicester, Leicester, UK.
rob.vanstee@leicester.ac.uk.

Abstract. Online Bin Stretching is a semi-online variant of bin packing in which the algorithm has to use the same number of bins as the optimal packing, but is allowed to slightly overpack the bins. The goal is to minimize the amount of overpacking, i.e., the maximum size packed into any bin.
We give an algorithm for Online Bin Stretching with a stretching factor of 1.5 for any number of bins. We also show a specialized algorithm for three bins with a stretching factor of $11/8 = 1.375$.

1 Introduction

The most famous algorithmic problem dealing with online assignment is arguably Online Bin Packing. In this problem, known since the 1970s, items of size between 0 and 1 arrive in a sequence and the goal is to pack these items into the least number of unit-sized bins, packing each item as soon as it arrives.

Online Bin Stretching, which has been introduced by Azar and Regev in 1998 [2], deals with a similar online scenario. Again, items of size between 0 and 1 arrive in a sequence, and the algorithm needs to pack them as soon as each item arrives, but it has two advantages: (i) The packing algorithm knows m, the number of bins that an optimal offline algorithm would use, and must also use only at most m bins, and (ii) the packing algorithm can use bins of capacity R for some $R \geq 1$. The goal is to minimize the stretching factor R.

While formulated as a bin packing variant, Online Bin Stretching can also be thought of as a semi-online scheduling problem, in which we schedule jobs in an online manner on exactly m machines, before any execution starts. We have a guarantee that the optimum offline algorithm could schedule all jobs with makespan 1. Our task is to present an online algorithm with makespan of the schedule being at most R.

History. Online Bin Stretching has been proposed by Azar and Regev [2]. The original lower bound of 4/3 for three bins has appeared even before that, in [10], for two bins together with a matching algorithm. Azar and Regev extended the same lower bound to any number of bins and gave an online algorithm with a stretching factor 1.625.

⋆ Supported by the project 14-10003S of GA ČR and by the GAUK project 548214.

The problem has been revisited recently, with both lower bound improvements and new efficient algorithms. On the algorithmic side, Kellerer and Kotov [9] have achieved a stretching factor $11/7 \approx 1.57$ and Gabay et al. [7] have achieved $26/17 \approx 1.53$. In the case with only three bins, the previously best algorithm was due to Azar and Regev [2], with a stretching factor of 1.4.

On the lower bound side, the lower bound 4/3 of [2] was surpassed only for the case of three bins by Gabay et al. [6], who show a lower bound of 19/14, using an extensive computer search.

Our contributions. In Section 2, we present a new algorithm for Online Bin Stretching with a stretching factor of 1.5. We build on the techniques of [9,7] who designed two-phase algorithms where the first phase tries to fill some bins close to $R - 1$ and achieve a fixed ratio between these bins and empty bins, while the second phase uses the bins in blocks of fixed size and analyzes each block separately. This technique, with some case analysis, seemed to be able to lead to improved results approaching 1.5. To actually reach 1.5, we needed to significantly improve the analysis using amortization techniques (represented by a weight function in our presentation) to amortize among blocks and bins of different types.

In Section 3, we focus on the case of three bins. For this case, there is a recent lower bound of $19/14 \approx 1.357$ [6]. We present an algorithm for three bins of capacity $11/8 = 1.375$. This is the first improvement of the stretching factor 1.4 of Azar and Regev [2] for three bins and significantly decreases the remaining gap.

Related work. The NP-hard problem Bin Packing was originally proposed by Ullman [11] and Johnson [8] in the 1970s. Since then it has seen major interest and progress, see the survey of Coffman et al. [4] for many results on classical Bin Packing and its variants. While our problem can be seen as a variant of Bin Packing, note that the algorithms cannot open more bins than the optimum and thus general results for Bin Packing do not translate to our setting.

As noted, Online Bin Stretching can be formulated as the online scheduling on m identical machines with known optimal makespan. Such algorithms were studied and are important in designing constant-competitive algorithms without the additional knowledge, e.g., for scheduling in the more general model of uniformly related machines [1,3,5].

Definitions and notation. Our main problem, Online Bin Stretching, can be described as follows:
Input: an integer m and a sequence of items $I = i_1, i_2, \ldots$ given online one by one. Each item has a *size* $s(i) \in [0, 1]$ and must be packed immediately and irrevocably.
Parameter: The *stretching factor* R, a limit of the capacity of all bins.
Output: Partitioning (packing) of I into bins $B_1, \ldots, B_m$ so that $\sum_{i \in B_j} s(i) \leq R$ for all $j = 1, \ldots, m$.
Guarantee: there exists a packing of all items in I into m bins of capacity 1.
Goal: Design an online algorithm with the stretching factor R as small as possible which packs all input sequences satisfying the guarantee.

For a bin B, we define the *size of the bin* $s(B) = \sum_{i \in B} s(i)$. Unlike $s(i)$, $s(B)$ can change during the course of the algorithm, as we pack more and more items into the bin. To easily differentiate between items, bins and lists of bins, we use lowercase letters for items (i, b, x), uppercase letters for bins (A, B, X), and calligraphic letters for lists of bins ($\mathcal{A}$, $\mathcal{C}$, $\mathcal{L}$).

In both sections of our paper, we rescale the item sizes and bin capacities for simplicity. Therefore, in our setting, each item has an associated size $s(i) \in [0, k]$, where $k \in \mathbb{N}$ is also the capacity of the bins which the optimal offline algorithm uses. The online algorithm for Online Bin Stretching uses bins of capacity $t \in \mathbb{N}$, $t \geq k$. The resulting stretching factor is thus t/k.

We omit some proofs due to space restrictions. Full version available at `http://arxiv.org/abs/1404.5569`.

2 Upper bound for an arbitrary number of bins

We rescale the bin sizes so that the optimal bins have size 12 and the bins of the algorithm have size 18.

We follow the general two-phase scheme of recent results [9,7] which we sketch now. In the first phase of the algorithm we try to fill the bins so that their size is at most 6, as this leaves space for an arbitrary item in each bin. Of course, if items larger than 6 arrive, we need to pack them differently, namely in bins of size at least 12, whenever possible. We stop the first phase when the number of non-empty bins of size at most 6 is three times the number of empty bins. In the second phase, we work in blocks of three non-empty bins and one empty. The goal is to show that we are able to fill the bins so that the average size is at least 12, which guarantees we are able to pack the total size of $12m$ which is the upper bound on the size of all items.

The limitation of the previous results using this scheme was that the volume achieved in a typical block of four bins is slightly less than four times the size of the optimal bin, which then leads to bounds strictly above 3/2. This is also the case in our algorithm: A typical block may contain in three bins items from the first phase of size just above 4 plus one item of size 7 from the second phase, while the last bin contains two items of size 7 from the second phase—a total of 47 instead of desired $4 \cdot 12$. However, we notice that such a block contains five items of size 7 which the optimum cannot fit into four bins. To take an advantage of this, we cannot analyze each block separately. Instead, we need to show that a bin with no item of size more than 6 typically has size at least 13 and amortize among the blocks of different types. Technically this is done using a weight function w that takes into account both the total size of items and the number of items larger than 6. This is the main new technical idea of our proof.

There are other complications. We need to guarantee that a typical bin of size at most 6 has size at least 4 after the first phase. However, this is impossible to guarantee if the items packed there have size between 3 and 4. Larger items are fine, as one per bin is sufficient, and the smaller ones are fine as well as we can always fit at least two of them and this guarantees that we have only two bins

filled below 4. This motivates our classification of items: Only the regular items of size in $(0,3]\cup(4,6]$ are packed in the bins filled up to size 6. The medium items of size in $(3,4]$ are packed in their own bins (four or five per bin). Similarly, large items of size in $(6,9]$ are packed in pairs in their own bins. Finally, the huge items of size larger than 9 are handled similarly as in the previous papers: If possible, they are packed with the regular items, otherwise each in their own bin.

The introduction of medium size items in turn implies that we need to revisit the analysis of the first phase and also of the case when the first phase ends with no empty bin. These parts of the proof are similar to the previous works, but due to the new item type we need to carefully revisit it; it is now convenient to introduce another weight function v that counts the items according to their type. The analysis of the second phase when empty bins are present is more complicated, as we need to take care of various degenerate cases, and it is also here where the novel amortization is used.

Lower bound. We note that this two-phase approach cannot give a better stretching factor than 1.5. Consider the following instance. Send two items of size 6 which are in the first phase packed separately into two bins. Then send $m-1$ items of size 12 and one of them must be put into a bin with an item of size 6, i.e., one bin receives items of size 18, while all the items can be packed into m bins of size 12. This instance and its modifications with more items of size 6 or slightly smaller items at the beginning thus show that decreasing the upper bound below 1.5 would need a significantly different approach, as we would be forced to pack these items in pairs. This also shows that the analysis of our algorithm is tight.

Now we are ready to proceed with the formal statement of the algorithm and proof.

Theorem 1. *There exists an algorithm for* Online Bin Stretching *with a stretching factor of* 1.5 *for an arbitrary number of bins.*

We take an instance with an optimal packing into m bins of size 12 and, assuming that our algorithm fails, we derive a contradiction. One way to get a contradiction is to show that the size of all items is larger than $12m$. We also use two other bounds in the spirit of weight functions: weight $w(i)$ and value $v(i)$. The weight $w(i)$ is a slightly modified size to account for items of size larger than 6. The value $v(i)$ only counts the number of items with relatively large sizes. For our calculations, it is convenient to normalize the functions so that they are at most 0 for optimal bins (see Lemma 1).

We classify the items and define their value $v(i)$ as follows.

Type	huge	large	medium	regular
$s(i)$	$(9,12]$	$(6,9]$	$(3,4]$	$(0,3]\cup(4,6]$
$v(i)$	3	2	1	0

Definition 1. *For a set of items A, we define the value $v(A)=(\sum_{i\in A} v(i))-3$ and we define weight $w(A)$ as follows. Let $k(A)$ be the number of large and huge*

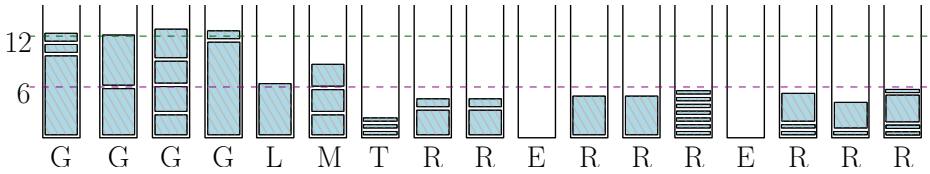

Fig. 1. A typical state of the algorithm after the first phase. The bin labels correspond to the bin types of the first phase. The non-complete bins (other than G) are ordered as in the list $\mathcal{L}$ at the beginning of the second phase with regular bins.

items in A. Then $w(A) = s(A) + k(A) - 13$. For a set of bins $\mathcal{A}$ we define $v(\mathcal{A}) = \sum_{A\in\mathcal{A}} v(A)$, $w(\mathcal{A}) = \sum_{A\in\mathcal{A}} w(A)$ and $k(\mathcal{A}) = \sum_{A\in\mathcal{A}} k(A)$.

Lemma 1. *For any packing of a valid instance into m bins $\mathcal{A}$ of any size, we have $w(\mathcal{A}) \leq 0$ and $v(\mathcal{A}) \leq 0$.* □

First phase. During the first phase, our algorithm maintains the invariant that only bins of the following types exist. See Figure 1 for an illustration of the types of the bins.

- **E** **Empty bins:** bins that have no item.
- **G** **Complete bins:** all bins A that have $w(A) \geq 0$ and $s(A) \geq 12$;
- **H** **Huge-item bins:** all bins A that contain a huge item (plus possibly some other items) and have $s(A) < 12$;
- **L** **One large-item bin:** a bin containing only a single large item;
- **M** **One medium-item bin:** a bin A with $s(A) \leq 13$ and only medium items;
- **T** **One tiny bin:** a bin with $s(A) \leq 3$;
- **R** **Regular bins:** all other bins with $s(A) \in (3, 6]$;

First-phase algorithm:

Let e be the number of empty bins and r the number of regular bins. If $r \geq 3e$, stop the first phase.

Assign the current item i according to its item type, using the first possible option in the particular column. The first letter in a cell indicates the required type of the bin before the assignment and the second column denotes the type of the bin after the assignment. If there are two types listed, the new bin type depends on the new size and weight of the bin.

Note: As an additional rule when packing regular items, the item is packed in a regular or tiny bin only if the total size packed into this bin does not exceed 6 afterwards.

Item type	huge	large	medium	regular
Option 1	**R** → **G**	**L** → **G**	**M** → **G/M**	**H** → **G/H**
Option 2	**T** → **H**	**E** → **L**	**E** → **M**	**R** → **R**
Option 3	**E** → **H**			**T** → **T/R**
Option 4				**E** → **T/R**

First we observe that the algorithm described in the box above is properly defined. For every type of item, packing it into an empty bin is an option, and the stopping criterion guarantees that the algorithm stops when no empty bin is available. We now state properties of the algorithm; all are simple invariants that follow from the description of the algorithm.

Lemma 2. *At any time during the first phase the following holds:*

(i) *All bins used by the algorithm are of type* **E**, **G**, **H**, **L**, **M**, **T**, *or* **R**.
(ii) *All complete bins B have $s(B) \geq 12$, $v(B) \geq 0$, and $w(B) \geq 0$.*
(iii) *If there is a huge-item bin, there is no regular and no tiny bin.*
(iv) *There is at most one large-item bin and at most one medium-item bin.*
(v) *There is at most one tiny bin T. If T exists, then for any regular bin, $s(T) + s(R) > 6$. There is at most one regular bin R with $s(R) \leq 4$.*
(vi) *At the end of the first phase $3e \leq r \leq 3e + 3$.* □

If the algorithm packs all items in the first phase, it stops. Otherwise according to Lemma 2(iii) we split the algorithm in two very different branches. If there is no regular bin, follow the second phase with huge-item bins below. If there is at least one regular bin, follow the second phase with regular bins.

Let $\mathcal{G}$ be the set of all complete bins; we do not use these bins in the second phase. In addition to $\mathcal{G}$ and either huge-item bins, or the regular and empty bins, there may exist at most three *special bins* denoted and ordered as follows: the large-item bin L, the medium item bin M, and the tiny bin T.

Second phase with huge-item bins. Let the list of bins $\mathcal{L}$ contain first all the huge-item bins, followed by the special bins L, M, in this order, if they exist. There are no other non-empty bins by Lemma 2 and no empty bins because we have $3e \leq r = 0$. We use First Fit on $\mathcal{L}$, without allowing new bins to be opened. Suppose that we have an instance that has a packing into bins of capacity 12 and on which our algorithm fails. We may assume that the algorithm fails on the last item f. By considering the total volume, there always exists a bin with size at most 12. Thus $s(f) > 6$ and $v(f) \geq 2$.

If during the second phase an item n with $s(n) \leq 6$ is packed into the last bin in $\mathcal{L}$, we know that all other bins have size more than 12, thus all the remaining items fit into the last bin. Otherwise we consider $v(\mathcal{L})$. Any bin $B \in \mathcal{G}$ has $v(B) \geq 0$ by Lemma 2(ii) and each huge-item bin gets nonnegative value too. Also $v(L) \geq -1$ if L exists. This shows that M must exist, since otherwise $v(\mathcal{L}) + v(f) \geq -1 + 2 \geq 1$, a contradiction.

M is the last bin of $\mathcal{L}$ and thus in this last case M contains only medium items from the first phase and possibly large and/or huge items from the second phase. We claim that $v(M) + v(f) \geq 2$ using the fact that f does not fit into M and M contains no item a with $v(a) = 0$: If f is huge we have $s(M) > 6$, thus M must contain either two medium items or one medium item and one large or huge item and $v(M) \geq -1$. If f is large, we have $s(M) > 9$; thus M contains either three medium items or one medium and one large or huge item and $v(M) \geq 0$. Thus we always have $v(\mathcal{L}) \geq -1 + v(M) + v(f) \geq 1$, a contradiction.

Second phase with regular bins. Let $\mathcal{E}$ resp. $\mathcal{R}$ be the set of empty resp. regular bins at the beginning of the second phase, and let $e = |\mathcal{E}|$. Let $\lambda \in \{0, 1, 2, 3\}$ be such that $|\mathcal{R}| = 3e + \lambda$; Lemma 4(vi) implies that it exists.

We order the bins that are not complete into a list $\mathcal{L}$ as follows. We group the bins in $\mathcal{E} \cup \mathcal{R}$ into blocks of typically one empty and three regular bins as follows. Denote the empty bins $E_1, E_2, \ldots, E_e$. The regular bins are denoted by $R_{i,j}$, $i = 1, ..., e+1$, $j = 1, 2, 3$. The ith block $\mathcal{B}_i$ consists of bins $R_{i,1}, R_{i,2}, R_{i,3}, E_i$ in this order. There are two exceptions: The last block $\mathcal{B}_{e+1}$ has no empty bin, only exactly 3 regular bins. The first block contains only λ regular bins instead of 3 and an empty bin. As the first regular bin we choose the one with size less than 4, if there is such a bin. By Lemma 2(v) there exists at most one such bin and all the remaining $R_{i,j}$ have size at least 4. Denote the first regular bin by $\bar{R}$ if $\mathcal{R} \neq \emptyset$; note that $\bar{R}$ is either the first bin in $\mathcal{B}_1$ or the first bin in $\mathcal{B}_2$ if $\lambda = 0$.

The list of bins $\mathcal{L}$ we use in the second phase contains first the special bins and then all the blocks $\mathcal{B}_1, \ldots, \mathcal{B}_{e+1}$. Thus the list $\mathcal{L}$ is (some or all of the first six bins may not exist):

$$L, M, T, R_{1,1}, R_{1,2}, R_{1,3}, E_1, R_{2,1}, R_{2,2}, R_{2,3}, E_2, \ldots, E_e, R_{e+1,1}, R_{e+1,2}, R_{e+1,3}.$$

Whenever we refer to the ordering of the bins, we mean the ordering in $\mathcal{L}$. See Figure 1 for an illustration.

We use First Fit on the reversed list $\mathcal{L}$ for huge items (that is, we pack each huge item to the last bin in $\mathcal{L}$ where it fits) and we use First Fit on $\mathcal{L}$ for all other items.

Suppose that we have an instance that has a packing into bins of capacity 12 and on which our algorithm fails. We may assume that the algorithm fails on the last item. Let us denote this item by f. Call the items that arrived in the second phase *new* (including f), the items from the first phase are *old*. See Figure 2 for an illustration of a typical final situation. Our overall strategy is to obtain a contradiction by showing that

$$w(\mathcal{L}) + w(f) > 0\,.$$

In some cases, we instead argue that $v(\mathcal{L}) + v(f) > 0$ or $s(\mathcal{L}) + s(f) > 12|\mathcal{L}|$. Any of these is sufficient for a contradiction, as all bins in $\mathcal{G}$ have both volume and weight nonnegative and size larger than 12. Note also that $s(f) > 6$ since by considering the total volume, there always exists a bin with size at most 12.

Let $\mathcal{H}$ denote all the bins from $\mathcal{L}$ with a huge item, and let $h = |\mathcal{H}| \bmod 4$. First we show that the average size of bins in $\mathcal{H}$ is large and exclude some degenerate cases.

Lemma 3. *Let ρ be the total size of old items in $\bar{R}$ if $\bar{R} \in \mathcal{H}$, otherwise set $\rho = 4$.*

(i) *The bins $\mathcal{H}$ are a final segment of the list and $\mathcal{H} \subsetneq \mathcal{E} \cup \mathcal{R}$.*
(ii) *We have $s(\mathcal{H}) \geq 12|\mathcal{H}| + h + \rho - 4$.*
(iii) *If $\mathcal{H}$ does not include $\bar{R}$, then $s(\mathcal{H}) \geq 12|\mathcal{H}| + h \geq 12|\mathcal{H}|$.*
(iv) *If $\mathcal{H}$ includes $\bar{R}$, then $s(\mathcal{H}) \geq 12|\mathcal{H}| + h - 1 \geq 12|\mathcal{H}| - 1$.* □

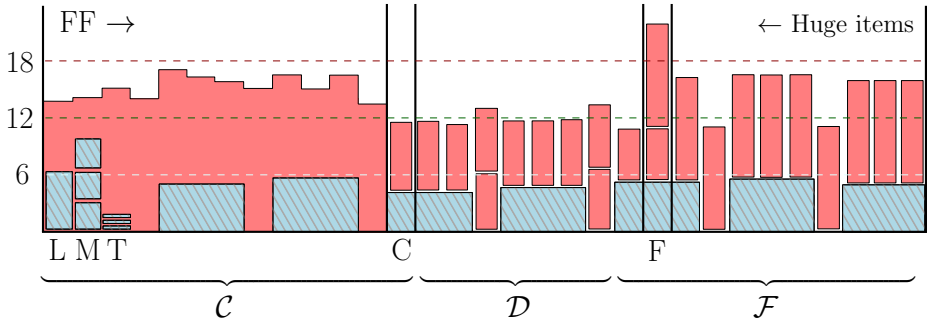

Fig. 2. A typical state of the algorithm after the second phase with regular bins. The gray (hatched) areas denote the old items (i.e., packed in the first phase), the red (solid) regions and rectangles denote the new items (i.e., packed in the second phase). The bins that are complete at the end of the first phase are not shown. The item f on which the algorithm fails is shown as packed into the final bin F and exceeding the capacity of the bin.

Let F, the *final bin* be the last bin in $\mathcal{L}$ before $\mathcal{H}$, or the last bin if $\mathcal{H} = \emptyset$; by Lemma 3 we have $F \in \mathcal{E} \cup \mathcal{R}$. Now modify the packing so that f is put into F, f is also considered a new item. Thus $s(F) > 18$ and f as well as all the new items packed in F or a bin before F satisfy the property that they do not fit into any previous bin. Let C, the *critical bin*, be the first bin in $\mathcal{L}$ of size at most 12; such a bin exists, as otherwise the total size is more than $12m$.

We start by some easy observations. Only items of size at most 9 are packed in bins before F, in F itself only the item f can be larger. All the new items in the bins after C are large; f can be also huge. Each bin, possibly with the exception of L and M, contains a new item, as it enters the phase with size at most 6, and the algorithm failed. Each bin in $\mathcal{E}$ before F contains two new items. The bin F always has two new items, one that did fit into it and f. More observations are given in the next two lemmata.

Lemma 4. *(i) Let B be any bin before F. Then $s(B) > 9$.*
(ii) Let B, B', B'' be any three bins in this order before or equal to F and let B'' contain two new items. Then $s(B) + s(B') + s(B'') > 36 + o$, where o is the size of old items in B''.
(iii) Let B be arbitrary and $B' \in \mathcal{R}$ after both B and C.
If $B' \neq \bar{R}$ then $s(B) + s(B') > 22$, in particular $s(B) > 11$ or $s(B') > 11$.
If $B' = \bar{R}$ then $s(B) + s(B') > 21$. □

Lemma 5. *The critical bin C is before F, there are at least two bins between C and F and C is not in the same block as F.* □

Now we partition $\mathcal{L}$ into several parts, see Figure 2 for an illustration of these parts. Let $\mathcal{F} = \mathcal{B}_i \cup \mathcal{H}$, where $F \in \mathcal{B}_i$. Let $\mathcal{D}$ be the set of all bins after C and before $\mathcal{F}$. Let $\mathcal{C}$ be the set of all bins before and including C. Lemma 5

shows that the parts are non-overlapping. We analyze the weight of the parts separately, essentially block by block. The proof is relatively straightforward if C is not special (and thus also $\mathcal{F} \notin \mathcal{B}_1$), which is the most important case driving our choices for w. A typical block has nonnegative weight, we gain more weight in the block of F which exactly compensates the loss of weight in $\mathcal{C}$, which occurs mainly in C itself.

Lemma 6. *If F is not in the first block then $w(\mathcal{F}) > 5$, else $w(\mathcal{F}) > 4$.* □

Lemma 7. *If $C \in \mathcal{R}$ then $w(\mathcal{C}) \geq -6$. If $C \in \mathcal{E}$ then $w(\mathcal{C}) \geq -5$. If C is a special bin then $w(\mathcal{C}) \geq -4$.* □

Lemma 8. (i) *For every block $\mathcal{B}_i \subseteq \mathcal{D}$ we have $w(\mathcal{B}_i) \geq 0$.*
(ii) *If there is no special bin in $\mathcal{D}$, then $w(\mathcal{D}) \geq 0$. If also $C \in \mathcal{R}$ then $w(\mathcal{D}) \geq 1$.*

We are now ready to derive the final contradiction. If $\mathcal{D}$ does not contain a special bin, we add the appropriate bounds from Lemmata 7, 8 and 6. If $C \in \mathcal{R}$ then F is not in the first block and $w(\mathcal{L}) = w(\mathcal{C})+w(\mathcal{D})+w(\mathcal{F}) > -6+1+5 = 0$. If $C \in \mathcal{E}$ then F is not in the first block and $w(\mathcal{L}) = w(\mathcal{C})+w(\mathcal{D})+w(\mathcal{F}) > -5+0+5 = 0$. If C is the last special bin then $w(\mathcal{L}) = w(\mathcal{C}) + w(\mathcal{D}) + w(\mathcal{F}) > -4 + 0 + 4 = 0$. In all subcases $w(\mathcal{L}) > 0$, a contradiction.

If $\mathcal{D}$ does contain a special bin we need to analyze several cases depending on the number of special bins and regular bins in $\mathcal{B}_1$.

In all of the cases we can derive a contradiction, which implies that our algorithm cannot fail. This concludes the proof of Theorem 1. □

3 Bin stretching for three bins

We scale the input sizes by 16. The stretched bins in our setting therefore have capacity 22 and the optimal offline algorithm can pack all items into three bins of capacity 16 each. The three bins of our setting are named A, B, and C. We prove the following theorem.

Theorem 2. *There exists an algorithm that solves* Online Bin Stretching *for three bins with stretching factor $1 + 3/8 = 1.375$.*

A natural idea is to try to pack first all items in a single bin, as long as possible. In general, this is the strategy that we follow. However, somewhat surprisingly, it turns out that from the very beginning we need to put items in two bins even if the items as well as their total size are relatively small.

It is clear that we have to be very cautious about exceeding a load of 6. For instance, if we put 7 items of size 1 in bin A, and 7 such items in B, then if two items of size 16 arrive, the algorithm will have a load of at least 23 in some bin. Similarly, we cannot assign too much to a single bin: putting 20 items of size 0.5 all in bin A gives a load of 22.5 somewhere if three items of size 12.5 arrive next.

On the other hand, it is useful to keep one bin empty for some time; many problematic instances end with three large items such that one of them has to be

placed in a bin that already has high load. Keeping one bin free ensures that such items must have size more than 11 (on average), which limits the adversary's options, since all items must still fit into bins of size 16.

Deciding when exactly to start using the third bin and when to cross the threshold of 6 for the first time was the biggest challenge in designing this algorithm: both of these events should preferably be postponed as long as possible, but obviously they come into conflict at some point.

Good situations. Before stating the algorithm itself, we list several *good situations* (GS). These are configurations of the three bins which allow us to complete the packing regardless of the following input. Obviously the identities of the bins are not important here; for instance, in the first good situation, all that we need is that *any* two bins together have items of size at least 26. We have used names only for clarity of presentation and of the proofs.

Good Situation 1. *Given a partial packing such that* $s(A) + s(B) \geq 26$ *and* $s(C)$ *is arbitrary, there exists an online algorithm that packs all remaining items into three bins of capacity* 22*.*

Proof. Since the optimum can pack into three bins of size 16, the total size of items in the instance is at most $3 \cdot 16 = 48$. If two bins have size $s(A)+s(B) \geq 26$, all the remaining items (including the ones already placed on C) have size at most 22. Thus we can pack them all into bin C. □

Good Situation 2. *Given a partial packing such that* $s(A) \in [4, 6]$ *and* $s(B)$ *and* $s(C)$ *are arbitrary, there exists an online algorithm that packs all remaining items into three bins of capacity* 22*.*

Proof. Let A be the bin with size between 4 and 6 and B be one of the other bins (choose arbitrarily). Put all the items greedily into B. When an item does not fit, put it into A, where it fits, as originally $s(A)$ is at most 6. Now the size of all items in B plus the last item is at least 22. In addition, A has items of size at least 4 before the last item by the assumption. Together we have $s(A) + s(B) \geq 26$, allowing us to apply GS1. □

Good Situation 3. *Given a partial packing such that* $s(A) \in [15, 22]$ *and either* (i) $s(C) \leq 6$ *and* $s(B)$ *is arbitrary or* (ii) $s(B)+s(C) \geq 22$*, there exists an online algorithm that packs all remaining items into three bins of capacity* 22*.*

Proof. If $s(B) + s(C) \geq 22$, then $\max(s(B), s(C)) \geq 11$, so we are in GS1 on bins A and B or on bins A and C. Else, if $s(C) \leq 6$, we pack arriving items into B. If $s(B) \geq 11$, we apply GS1 on bins A and B. Thus we can assume $s(B) < 11$ and we cannot continue packing into B any further. This implies that an item i arrives such that $s(i) > 11$. As $s(C) \leq 6$, we pack i into it and apply GS1 on bins A and C. □

Good Situation 4. *Given a partial packing such that* $s(A)+s(B) \geq 15+\frac{1}{2}s(C)$*,* $s(B) < 4$*, and* $s(C) < 4$*, there exists an online algorithm that packs all remaining items into three bins of capacity* 22*.* □

Good Situation 5. *Given a partial packing such that an item a with $s(a) > 6$ is packed into bin A, $s(B) \in [3, 6]$, and C is empty, there exists an algorithm that packs all remaining items into three bins of capacity 22.* □

Good Situation 6. *If $s(C) \leq 6 \leq s(B)$ and $s(A) \geq s(B) + 4 - s(C)$, there exists an algorithm that packs all remaining items into 3 bins of capacity 22.* □

Good Situation 7. *Suppose $s(C) \leq s(B) < 6 < s(A)$. If $s(A) \leq 9 + \frac{1}{2}(s(C) + s(B))$ and for some item x we have $s(A)+x > 22$, there exists an online algorithm that packs all remaining items into three bins of capacity 22.* □

The algorithm. We now proceed to describe the bin packing algorithm itself. Its analysis is omitted due to space restrictions. The algorithm will often use a special variant of First Fit, described as follows:

Definition 2. *Let $\mathcal{L} = (X|_k, Y|_l, \ldots)$ be a list of bins $X, Y, \ldots$ where each bin X has an associated integral capacity k satisfying $s(X) \leq k$.* GSFF($\mathcal{L}$) *(Good Situation First Fit) is an online algorithm for bin stretching that works as follows:*

Algorithm GSFF($\mathcal{L}$)**:** *For each item i:*
If it is possible to pack i into any bin (including bins not in $\mathcal{L}$, and using capacities of 22 for all bins) such that a good situation is reached, do so and continue with the algorithm of the relevant good situation.
Otherwise, traverse the list $\mathcal{L}$ in order and pack i into the first bin X such that $X|_k \in \mathcal{L}$ and $s(X) + s(i) \leq k$.

For example, GSFF($A|_4, B|_{22}$) checks whether either $(A \cup \{j\}, B, C)$, $(A, B \cup \{j\}, C)$ or $(A, B, C \cup \{j\})$ is a partial packing of any good situation. If this is not the case, the algorithm packs j into bin A provided that $s(A) + s(j) \leq 4$. If $s(A) + s(j) > 4$, the algorithm packs j into bin B with capacity 22. If j cannot be placed into B, GSFF($A|_4, B|_{22}$) reports failure and another online algorithm must be applied.

In the first phase, we pack items into two bins so that either an item of size 6 arrives relatively early in the input sequence, or we can reach a good situation.

Algorithm First Phase**:**
(1) GSFF($A|_4, B|_4$). Rename the bins so that $s(A) \geq s(B)$.
(2) Let the item on which Step (1) failed be j. If $s(j) > 6$ place j into A, and go to the Second Phase.
(3) Place j into B and rename the bins so that $s(A) > s(B) \geq s(C) = 0$.
(4) GSFF($B|_4, A|_q, C|_4$) where $q := 9 + \frac{1}{2}(s(B) + s(C))$. As soon as C receives its first item, rename the bins B and C so that $s(B) \geq s(C)$ and continue. Note that also the value of q may change between packing of different items.

We start the algorithm Second Phase only after Step (2) of First Phase fails when an item j of size $s(j) > 6$ arrives. We have not entered GS5 before placing j on A, and so we have $s(A \setminus \{j\}) \leq 3$.

Algorithm SECOND PHASE**:**
(1) GSFF$(A|_q, B|_4)$, where $q := 6 + s(j)$.
(2) If the next item x fits into A, apply GSFF$(A|_{22}, B|_{22}, C|_{22})$.
(3) Else: Place x into B. Let j' be the smallest item of $\{j, x\}$.
(4) Reorder the bins A and B so that $j' \in A$.
(5) GSFF$(A|_q, B|_{22})$, where $q := 6 + s(j')$.
(6) Place next item y into C. Let j'' be the smallest item of $\{j', y\}$.
(7) Reorder the bins A and C so that $j'' \in A$.
(8) GSFF$(A|_q, B|_{22}, C|_{22})$, where $q := 6 + s(j'')$.

Conclusions. With our algorithm for $m = 3$, the remaining gap is small. For arbitrary m, we have seen at the beginning of Section 2 that a significantly new approach would be needed for an algorithm with a better stretching factor than 1.5. Thus, after the previous incremental results, our algorithm is the final step of this line of study. It is quite surprising that there are no lower bounds for $m > 3$ larger than the easy bound of 4/3.

Acknowledgment. The authors thank Emese Bittner for useful discussions during her visit to Charles University.

References

1. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 44:486–504, 1997.
2. Y. Azar and O. Regev. On-line bin-stretching. In *Randomization and Approximation Techniques in Computer Science*, pages 71–81. Springer, 1998.
3. P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *J. Algorithms*, 35:108–121, 2000.
4. E. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin Packing Approximation Algorithms: Survey and Classification, In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer New York, 2013.
5. T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53:504–522, 2009.
6. M. Gabay, N. Brauner, V. Kotov. Computing lower bounds for semi-online optimization problems: Application to the bin stretching problem. HAL preprint hal-00921663, 2013.
7. M. Gabay, V. Kotov, N. Brauner. Semi-online bin stretching with bunch techniques. HAL preprint hal-00869858, 2013.
8. D. Johnson. *Near-optimal Bin Packing Algorithms.* Massachusetts Institute of Technology, project MAC. Massachusetts Institute of Technology, 1973.
9. H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013.
10. H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.
11. J. Ullman. The Performance of a Memory Allocation Algorithm. *Technical Report 100*, 1971.