Online Strip Packing with Polynomial Migration[*][†]

Klaus Jansen[1], Kim-Manuel Klein[2], Maria Kosche[3], and Leon Ladewig[4]

1 **Department of Computer Science, Kiel University, Kiel, Germany**
`kj@informatik.uni-kiel.de`
2 **Department of Computer Science, Kiel University, Kiel, Germany**
`kmk@informatik.uni-kiel.de`
3 **Department of Computer Science, Kiel University, Kiel, Germany**
`mkos@informatik.uni-kiel.de`
4 **Department of Computer Science, Technical University of Munich, Munich, Germany**
`ladewig@in.tum.de`

Abstract

We consider the relaxed online strip packing problem, where rectangular items arrive online and have to be packed into a strip of fixed width such that the packing height is minimized. Thereby, repacking of previously packed items is allowed. The amount of repacking is measured by the migration factor, defined as the total size of repacked items divided by the size of the arriving item. First, we show that no algorithm with constant migration factor can produce solutions with asymptotic ratio better than 4/3. Against this background, we allow amortized migration, i.e. to save migration for a later time step. As a main result, we present an AFPTAS with asymptotic ratio $1 + \mathcal{O}(\epsilon)$ for any $\epsilon > 0$ and amortized migration factor polynomial in $1/\epsilon$. To our best knowledge, this is the first algorithm for online strip packing considered in a repacking model.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms

Keywords and phrases strip packing, bin packing, online algorithms, migration factor

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2017.13

1 Introduction

In the classical *strip packing* problem we are given a set of two-dimensional items with heights and widths bounded by 1 and a strip of infinite height and width 1. The goal is to find a packing of all items into the strip without rotations such that no items overlap and the height of the packing is minimal. In many practical scenarios, the entire input is not known in advance. Therefore, an interesting field of study is the *online* variant of the problem. Here, items arrive over time and have to be packed immediately without knowing future items. Following the terminology of [11] for the online bin packing problem, in the *relaxed online strip packing* problem previous items may be repacked when a new item arrives.

There are different ways to measure the amount of repacking in a relaxed online setting. We follow the *migration model* introduced by Sanders, Sivadasan, and Skutella in [24]. For online job scheduling on identical parallel machines they defined the *migration factor* μ as

[*] A full version of the paper is available at `http://arxiv.org/abs/1706.04939`.
[†] This work was partially supported by DFG Project, "Robuste Online-Algorithmen für Scheduling- und Packungsprobleme", JA 612 /19-1, and by GIF-Project "Polynomial Migration for Online Scheduling".

follows: When a new job of size p_j arrives, jobs of total size μp_j can be reassigned. In the context of online strip packing the migration factor μ ensures that the total area of repacked items is at most μ times the area of the arrived item.

By a well known relation between strip packing and parallel job scheduling [14], any (online) strip packing algorithm applies to (online) scheduling of parallel jobs. The latter problem is highly relevant e. g. in computer systems [14, 27, 23].

Preliminaries. Since strip packing is NP-hard [1], research focuses on efficient approximation algorithms. Let $A(I)$ denote the packing height of algorithm A on input I and $\mathrm{OPT}(I)$ the minimum packing height. The *absolute (approximation) ratio* is defined as $\sup_I A(I)/\mathrm{OPT}(I)$ while the *asymptotic (approximation) ratio* as $\limsup_{\mathrm{OPT}(I)\to\infty} A(I)/\mathrm{OPT}(I)$. Typically, the performance of online algorithms is measured by *competitive analysis*, where an online algorithm is compared with an optimal offline algorithm. In the following, all ratios mentioned in the context of online algorithms are competitive.

1.1 Related Work

Offline. Strip packing is one of the classical packing problems and receives ongoing research interest in the field of combinatorial optimization. Since Baker, Coffman and Rivest [1] gave the first algorithm with asymptotic ratio 3, strip packing was investigated in many studies, considering both asymptotic and absolute approximation ratios. We refer the reader to [6] for a survey. For the asymptotic ratio, a well-known result is the AFPTAS by Kenyon and Rémila [21]. Concerning the absolute ratio, currently the best known algorithm of ratio $5/3+\epsilon$ is by Harren et al. [13].

An interesting result was given by Han et al. in 2007. In [12], they studied the relation between bin packing and strip packing and developed a framework between both problems. For the offline case, it is shown that any bin packing algorithm can be applied to strip packing while maintaining the same asymptotic ratio.

Online. The first algorithm for online strip packing was given by Baker and Schwarz [2] in 1983. Using the concept of shelf algorithms [1], they derived the algorithm *First-Fit-Shelf* with asymptotic ratio arbitrary close to 1.7 and absolute ratio 6.99 (where all rectangles have height at most $h_{\max} = 1$). Later, Csirik and Woeginger [8] showed a lower bound of $h_\infty \approx 1.69$ on the asymptotic ratio for the concept of shelf algorithms and gave an improved shelf algorithm with asymptotic ratio $h_\infty + \epsilon$ for any $\epsilon > 0$. The framework of Han et al. [12] is applicable in the online setting if the online bin packing algorithm belongs to the class *Super Harmonic*. Using Seiden's online bin packing algorithm *Harmonic++* [25], there exists an algorithm for online strip packing with an asymptotic ratio of 1.58889. In 2007 and 2009, the concept of *First-Fit-Shelf* by Baker and Schwarz was improved independently by two research groups, Hurink and Paulus [14] and Ye, Han, and Zhang [28]. Both improve the absolute competitive ratio of from 6.99 to 6.6623 without a restriction on $h_{\max}$. Further results on special variants of online strip packing were given by Imeh [16] and Ye, Han, and Zhang [29].

On the negative side, there is no algorithm for online strip packing (without repacking) with an asymptotic ratio better then 1.5404 since the lower bound in [3] for online bin packing is also valid for online strip packing. Regarding the absolute ratio, the first lower bound of 2 from [5] was improved in several studies [19, 15, 22]. Currently, the best known lower bound by Yu, Mao, and Xiao [30] is $(3+\sqrt{2})/2 \approx 2.618$.

Related results on the migration model. Since its introduction by Sanders, Sivadasan, and Skutella [24], the migration model became increasingly popular. In the context of online scheduling on identical machines, Sanders, Sivadasan, and Skutella [24] gave a PTAS with migration factor $2^{\mathcal{O}\left((1/\epsilon)\log^2(1/\epsilon)\right)}$ for the objective of minimizing the makespan. Thereby, the migration factor in [24] depends only on the approximation ratio ϵ and not on the input size. Such algorithms are called *robust*.

Skutella and Verschae [26] studied scheduling on identical machines while maximizing the minimum machine load, called *machine covering*. They considered the *fully dynamic* setting in which jobs are also allowed to depart. Due to the presence of very small jobs, Skutella and Verschae showed that there is no PTAS for this problem in the migration model. Instead, they introduced the *reassignment cost model*, in which an amortized analysis of the migration factor is allowed. Using the reassignment cost model, they gave a robust PTAS for the problem with amortized migration factor $2^{\mathcal{O}\left((1/\epsilon)\log^2(1/\epsilon)\right)}$.

Also online bin packing has been investigated in the migration model in a sequence of papers, inspired by the work of Sanders, Sivadasan, and Skutella [24]: The first robust APTAS for relaxed online bin packing was given in 2009 by Epstein and Levin [10]. They obtained an exponential migration factor $2^{\mathcal{O}\left((1/\epsilon^2)\log 1/\epsilon\right)}$. In 2013, Jansen and Klein [17] improved this result and gave an AFPTAS with polynomial migration factor $\mathcal{O}\left(\frac{1}{\epsilon^3}\log\frac{1}{\epsilon^4}\right)$. The development of advanced LP/ILP-techniques made this notable improvement possible. Furthermore, in [4] Berndt, Jansen, and Klein used the techniques developed in [17] to give an AFPTAS for fully dynamic bin packing with a similar migration factor.

Our contribution

To the authors knowledge, there exists currently no algorithm for online strip packing in the migration or any other repacking model. Therefore, we present novel ideas to obtain the following results: First, a relatively simple argument shows that in the (strict) migration model it is not possible to maintain solutions that are close to optimal. We prove the following theorem in Section 1.3:

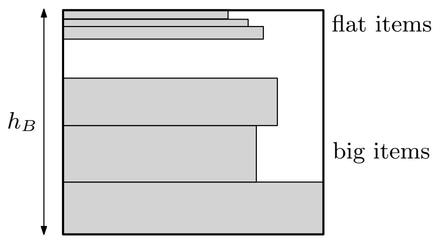
▶ **Theorem 1.1.** *In the (strict) migration model, there is no approximation algorithm for relaxed online strip packing with asymptotic competitive ratio better than* $4/3$.

For this reason, it is natural to extend the migration model such that amortization is allowed. We say that an algorithm has an *amortized* migration factor of μ if for every time step t, the total migration (i. e. the total area of repacked items) up to time t is bounded by $\mu\sum_{i=1}^{t}\mathrm{SIZE}(i_t)$, where $\mathrm{SIZE}(i_t)$ is the area of the item arrived at time t. Adapted to scheduling problems, this corresponds with the reassignment cost model introduced by Skutella and Verschae in [26]. Consequently, we focus on an approach that makes use of amortization and therefore admits an asymptotic approximation scheme. We adapt several offline and online techniques and combine them with our novel approaches to obtain the following main result:

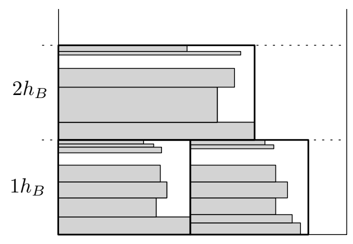
▶ **Theorem 1.2.** *There is a robust AFPTAS for relaxed online strip packing with an amortized migration factor polynomial in* $1/\epsilon$.

1.2 Technical Contribution

A general approach in the design of robust online algorithms is to rethink existing algorithmic strategies that work for the corresponding offline problem in a way that the algorithm can



(a) Packing of items in a container



(b) Packing of containers in the strip

Figure 1 Packing structure of our approach. Items are packed into containers of fixed height h_B, thus the packing of containers results in a bin packing problem.

adapt to a changing problem instance. The experiences that were made so far in the design of robust algorithms (see [17, 4, 26]) are to design the algorithm in a way such that the generated solutions fulfill very tight structural properties. Such solutions can then be adapted more easily as new items arrive.

A first approach would certainly be do adapt the well known algorithm for (offline) strip packing by Kenyon and Rémila [21] to the online setting. However, we can argue that the solutions generated by this algorithm do not fulfill sufficient structural properties. In the algorithm by Kenyon and Rémila, the strip is divided vertically into segments, where each segment is configured with a set of items. Thereby, each segment can have a different height. Now consider the online setting, where we are asked for a packing for the enhanced instance that maintains most parts of the existing packing. Obviously, it is not enough to place new items on top of the packing as this would exceed the approximation guarantee. To guarantee a good competitive ratio, existing configurations of the segments need to be changed. However, this seems to be hard to do as the height of a configuration can change. Gaps can occur in the packing as a segment might decrease in height or vice versa a segment might increase in height and therefore does not fit anymore in its current position. Over time this can lead to a very fragmented packing. On the other hand, closing gaps in a fragmented packing can cause a huge amount of repacking.

Containers. Therefore, we follow a different approach to develop an algorithm that guarantees solutions with a more modular structure. A central idea is to batch items to larger rectangles of fixed height, called *containers* (see Figure 1a). As each container has the same height h_B, it is natural to divide the strip into *levels* of equal height h_B (see Figure 1b) and fill each level with containers best possible. Thus, finding a container packing is in fact a bin packing problem, where levels correspond with bins and the sizes of the bin packing items are given by the container widths. This approach was studied in the offline setting by Han et al. in [12], while an analysis in the online setting is more sophisticated.

Thus, the packing of items into the strip is given by two assignments: By the *container assignment* each item is assigned a container where its is placed. Moreover, the *level assignment* describes which container is placed in which level (corresponds with the bin packing solution). To guarantee solutions with good approximation ratio, both functions have to satisfy certain properties.

Dynamic rounding / Invariant properties. For the container assignment, a natural choice would certainly be to assign the widest items to the first container, the second widest to the second container, and so on. In [12], Han et al. show that this container assignment is

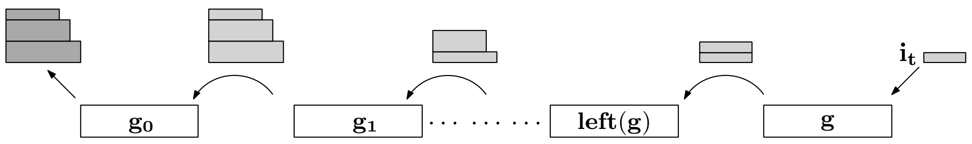


Figure 2 SHIFT operation moves widest items between groups to insert new item i_t.

somehow optimal. However, in the online setting we can not maintain this strict order while bounding the repacking size. Therefore, we use a relaxed ordering by introducing groups for containers of similar width and requiring the sorting over the groups, rather than over containers. For this purpose, we adapt the dynamic rounding technique developed by Berndt, Jansen, and Klein in [4] and formulate important characteristics as *invariant properties.*

Shift. In order to insert new items, we develop an operation called SHIFT. The idea is to move items between containers of different groups such that the invariant properties stay fulfilled. When inserting an item i_t via SHIFT into group[1] g, items are moved from g to the group *left*(g), where again items are shifted to the next group, and so on (see Figure 2). Thereby, the total height of the shifted items can increase in each step. However, it is limited such that items that can not be shifted further (at group g_0 in Figure 2) can be packed into one additional container. This way, we get a new container assignment for the enhanced instance which maintains the approximation guarantee and all desired structural properties.

LP/ILP-techniques. As a consequence of the shift operation, there may be a new container which has to be inserted into the packing. Obviously, placing new containers always into new levels may lead to a level assignment which does not satisfy the approximation guarantee. Therefore, the existing level assignment has to be changed, which causes further repacking. We apply the LP/ILP-techniques developed in [17] to maintain a good level assignment while the amortized migration factor is polynomial in $1/\epsilon$.

Packing of small items. Another challenging part regards the handling of items with small area. Without maintaining an advanced structure, small items can fractionate the packing in a difficult way. Such difficulties also arise in related optimization problems, see e.g. [26, 4]. For the case of flat items (with small height) we overcome these difficulties by the packing structure shown in Figure 1a: Flat items are separated from big items in the containers and are sorted by width such that the least wide item is at the top. Narrow items (small width) can be used to fill gaps in the packing while grouping narrow items of similar height. We sketch some ideas for the packing of small items in the Appendix A-B and refer to the full version [18] for all details.

1.3 Lower Bound

In this section we prove Theorem 1.1. We use an adversary to construct an instance with arbitrary optimal packing height, but $A(I) \geq \frac{4}{3}\,\mathrm{OPT}(I)$ for any such algorithm A.

Proof. Let A be an algorithm for relaxed online strip packing with migration factor μ. We show that for any height h there is an instance I with $\mathrm{OPT}(I) \geq h$ and $A(I) \geq \frac{4}{3}\,\mathrm{OPT}(I)$. The instance consists of two item types: A *big* item has width $\frac{1}{2} - \epsilon$ and height 1, while a *flat*

[1] In the following, by ‘group of an item’ we mean the group of the container in which the item is placed.

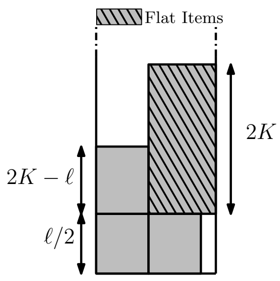


Figure 3 Optimal online packing.

item has width $\frac{1}{2}+\epsilon$ and height $\frac{1}{2\lceil\mu\rceil}$. For an item i let SIZE(i) denote its area. Note that A can not repack a big item b when a flat item f arrives, as $\text{SIZE}(b) > \mu\,\text{SIZE}(f)$ for $\epsilon < 1/6$.

First, the adversary sends $2K$ big items, where $K = 3\lceil h\rceil$. Let ℓ be the number of big items that are packed by A next to another big item. The packing has a height of at least $\frac{\ell}{2} + 2K - \ell = 2K - \frac{\ell}{2}$ (see Figure 3). Since the optimum packing height for $2K$ big items is K (always two items in one level), A has an absolute ratio of at least $2 - \frac{\ell}{2K}$. If $\ell \leq \frac{4K}{3}$, the absolute ratio is at least $\frac{4}{3}$ and nothing else is to show.

Now assume $\ell > \frac{4K}{3}$. In this case, the adversary sends $k = 4\lceil\mu\rceil K$ flat items of total height $2K$. In the optimal packing of height $2K$ big items and flat items form separate stacks that are placed next to each other. Note that no two flat items can be packed next to each other. Since A can not repack any big item when a flat item arrives, in the best possible packing achievable by A flat items of total height $2K - \ell$ are packed next to big items (see Figure 3, flat items are packed in the dashed area). Therefore, the packing height is at least $2K + \frac{\ell}{2}$ and hence the absolute ratio is at least $1 + \frac{\ell}{4K} \geq \frac{4}{3}$. In either case, it follows that the asymptotic ratio is at least $4/3$ by considering $K \to \infty$. ◀

1.4 Remainder of the Paper

In the remainder of this paper we give a high-level description of the proof of Theorem 1.2. Thereby, we focus on *big items* having minimum area ϵ^2 (see below). For most of the technical details as well as the handling of small items we refer to the full version [18].

Throughout the following sections, let $\epsilon \in (0, 1/4]$ be a constant such that $1/\epsilon$ is integer. We denote the height and width of an item i by $h(i)$ and $w(i)$ (both at most 1) and define $\text{SIZE}(i) = w(i)h(i)$. An item i is called *big* if $h(i) \geq \epsilon$ and $w(i) \geq \epsilon$. Let I_L be the set of big items. If R is a set of items, let $\text{SIZE}(R) = \sum_{i\in R}\text{SIZE}(i)$ and $h(R) = \sum_{i\in R} h(i)$.

2 Container Packing

Recall that we follow a two-level-approach to obtain the actual packing: Items are packed into containers of equal height h_B, whereby the widest item inside a container defines its width (see Figure 1a). The strip is divided into levels of height h_B, where the containers are packed (see Figure 1b). In this section we state important *invariant properties* concerning the relation between items and containers. Further, we show that if these invariant properties hold, the container packing yields a good approximation to the strip packing problem.

In order to find a container packing, we use a common LP formulation by Eisemann [9] (see also [4, 17]). However, the number of occurring container widths has to be bounded to solve the LP efficiently. Therefore, we introduce groups for the containers and round each container width to the largest width in its group, similar to rounding techniques in bin packing [20]. Nevertheless, the rounding has to be flexible enough for the online setting. We

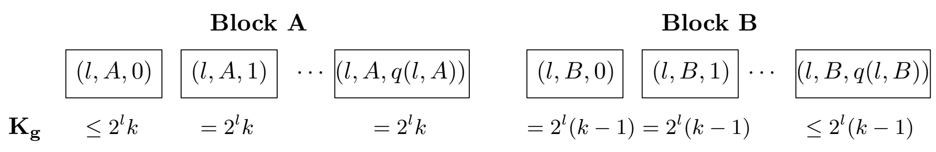


Figure 4 Groups of one category $l \in W$ and number of containers K_g per group.

adapt the dynamic rounding technique developed in [4], which is described in the following section.

2.1 Dynamic Rounding

Let $\mathcal{C}$ be the set of containers. Each container is assigned to a *(width) category* $l \in \mathbb{N}$, where container c has width category l if $w(c) \in (2^{-(l+1)}, 2^{-l}]$. Let W denote the set of all non-empty categories and define $\omega = |W|$ in the following. It follows immediately that $\omega = \mathcal{O}(\log 1/\epsilon)$. Furthermore, we build groups within the categories: A group $g \in G$ is a triple (l, X, r), where $l \in W$ is the category, $X \in \{A, B\}$ is the *block*, and $r \in \mathbb{N}$ is the *position* in the block. The maximum position of category l at block X that is non-empty is denoted by $q(l, X)$. Figure 4 outlines the group structure of one category $l \in W$ (the values for K_g will become clear in Section 2.2). For a group $g = (l, X, r)$ the groups *left*(g) and *right*(g) are defined as the respective neighboring groups[2] in the order shown in Figure 4.

By the notion of blocks, groups of one category can be partitioned into two types. This becomes helpful to maintain the invariant properties with respect to the growing set of items. More details on that are given in the later Sections 2.2 and 3.2.

The assignment from containers to groups is given by a *rounding function* $R \colon \mathcal{C} \to G$. Let $K_g = |\{c \in \mathcal{C} \mid R(c) = g\}|$ be the number of containers of group g. Let ${I_L}^g$ be the set of items in (containers of) group g.

2.2 Invariant Properties

In Section 1.2 we argued that only solutions with strong structural properties can be adapted appropriately in the online setting while maintaining a good competitive ratio. Definition 2.1 formalizes this central properties.

▶ **Definition 2.1** (Invariant properties). Let $k \in \mathbb{N}$ be a parameter and $h(g) = \sum_{i \in {I_L}^g} h(i)$ be the total height of items in group g.

(a) Items correspond to categories
$2^{-(l+1)} < w(i) \leq 2^{-l} \qquad \forall i \in {I_L}^g \text{ s.t. } g = (l, \cdot, \cdot)$

(b) Sorting of items over groups
$w(i) \geq w(i') \qquad \forall i \in {I_L}^g, i' \in {I_L}^{g'} \text{ s.t. } g = \mathit{left}(g')$

(c) Number of containers in block A
$K_{(l,A,0)} \leq 2^l k,$
$K_{(l,A,r)} = 2^l k \qquad \forall l \in W \text{ and } \forall 1 \leq r \leq q(l, A)$

(d) Number of containers in block B
$K_{(l,B,q(l,B))} \leq 2^l(k-1),$
$K_{(l,B,r)} = 2^l(k-1) \qquad \forall l \in W \text{ and } \forall 0 \leq r < q(l, B)$

(e) Total height of items per group
$(h_B - 1)(K_g - 1) \leq h(g) \leq (h_B - 1)K_g \qquad \forall g \in G$

[2] Set *left*$((l, A, 0)) = (l, A, -1)$ and *right*$((l, B, q(l, B))) = (l, B, q(l, B) + 1)$ as temporary groups.

Property (a) ensures that each item is inserted into the right category. Note that as a consequence, each container of a group $(l,\cdot,\cdot)$ has a width in $(2^{-(l+1)}, 2^{-l}]$. By property (b), all items in a group g have a width greater or equal than items in the group *right*(g). That is, instead of a strict order over all containers, (b) ensures an order over groups of containers. The properties (c) and (d) set the number of containers to a fixed value, except for special cases (see Figure 4): Groups in block A have more containers than groups in block B. Moreover, there are two *flexible groups* (namely $(l, A, 0)$ and $(l, B, q(l, B))$) whose number of containers is only upper bounded. Finally, property (e) ensures an important relation between items and containers of one group g: Since $h(g) \leq K_g(h_B - 1)$, at least one of the K_g containers has a filling height of at most $h_B - 1$ and thus can admit a new item. However, the lower bound $(h_B - 1)(K_g - 1) \leq h(g)$ ensures that each container is well filled in an average container assignment.

One of the important consequences of the invariant properties is the fact that the number of non-empty groups $|G|$ can be bounded from above, assuming that the instance is not too small. Therefore, the parameter k has to be set in a particular way:

▶ **Lemma 2.2.** *For* $k = \left\lfloor \frac{\epsilon}{4\omega h_B} \operatorname{SIZE}(I_L) \right\rfloor$ *the number of non-empty groups in* G *is bounded by* $\mathcal{O}\left(\frac{\omega}{\epsilon}\right) = \mathcal{O}\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right)$, *assuming that* $\operatorname{SIZE}(I_L) \geq \frac{24\omega h_B(h_B-1)}{\epsilon h_B - 2\epsilon}$.

Proof. Let $G_1 = G \setminus \left(\bigcup_{l\in W}(l, A, 0) \cup \bigcup_{l\in W}(l, B, q(l, B))\right)$ and let $g \in G_1$. Since by invariant (a) every container of group g has width greater than $2^{-(l+1)}$, it follows together with the further invariant properties

$$\begin{aligned}
\operatorname{SIZE}(I_L{}^g) &> 2^{-(l+1)}(h_B - 1)(K_g - 1) && \text{(a), (e)}\\
&\geq 2^{-(l+1)}(h_B - 1)(2^l(k-1) - 1) && \text{(c), (d)}\\
&= \frac{1}{2}(h_B - 1)(k-1) - 2^{-(l+1)}(h_B - 1)\\
&\geq \frac{1}{2}(h_B - 1)(k-1) - \frac{h_B - 1}{2}\\
&= \frac{1}{2}(h_B - 1)(k-2)\,.
\end{aligned}$$

Now, let $I_L^{(l)}$ be the set of items in I_L which belong to containers of category l. It holds that $\operatorname{SIZE}(I_L^{(l)}) \geq \sum_{g=(l,\cdot,\cdot)\in G_1} \operatorname{SIZE}(I_L{}^g) \geq (q(l, A) + q(l, B))\left(\frac{1}{2}(h_B - 1)(k-2)\right)$ and resolving leads to

$$q(l, A) + q(l, B) \leq \frac{2\operatorname{SIZE}(I_L^{(l)})}{(h_B - 1)(k-2)}\,. \tag{1}$$

We now show $(h_B - 1)(k-2) \geq \frac{\epsilon}{8\omega h_B}\operatorname{SIZE}(I_L)$. The assumption on $\operatorname{SIZE}(I_L)$ is equivalent to $\frac{\epsilon}{4\omega h_B}\operatorname{SIZE}(I_L) - 3 \geq \frac{\epsilon}{8\omega(h_B-1)}\operatorname{SIZE}(I_L)$. Therefore,

$$k - 2 = \left\lfloor \frac{\epsilon}{4\omega h_B}\operatorname{SIZE}(I_L) \right\rfloor - 2 \geq \frac{\epsilon}{4\omega h_B}\operatorname{SIZE}(I_L) - 3 \geq \frac{\epsilon}{8\omega(h_B - 1)}\operatorname{SIZE}(I_L)$$

and thus

$$(h_B - 1)(k-2) \geq \frac{(h_B - 1)\epsilon}{8\omega(h_B - 1)}\operatorname{SIZE}(I_L) = \frac{\epsilon}{8\omega}\operatorname{SIZE}(I_L)\,.$$

Further, we get

$$\frac{2\operatorname{SIZE}(I_L)}{(h_B - 1)(k-2)} \leq \frac{2\operatorname{SIZE}(I_L)}{\frac{\epsilon}{8\omega}\operatorname{SIZE}(I_L)} = \frac{16\omega}{\epsilon}\,. \tag{2}$$

As shown in Figure 4, for each category l there are $q(l,A)+q(l,B)+2$ groups. Now, summing over all categories $l \in W$ concludes the proof:

$$\begin{aligned}
&\sum_{l\in W} q(l,A)+q(l,B)+2 \\
&\le \sum_{l\in W}\left(\frac{2\,\mathrm{SIZE}(I_L^{(l)})}{(h_B-1)(k-2)}+2\right) && \text{eq. (1)}\\
&= 2\,|W| + \frac{2}{(h_B-1)(k-2)}\sum_{l\in W}\mathrm{SIZE}(I_L^{(l)})\\
&= 2\,|W| + \frac{2\,\mathrm{SIZE}(I_L)}{(h_B-1)(k-2)}\\
&\le 2\omega + \frac{16\omega}{\epsilon} && \text{eq. (2)}
\end{aligned}$$

◀

2.3 Approximation Guarantee

Furthermore, we can argue that if the invariant properties of Definition 2.1 are fulfilled, the rounded container packing yields a good approximation to a packing of the instance I_L.

Let $con\colon I_L \to \mathcal{C}$ be a container assignment and $R\colon \mathcal{C} \to G$ be a rounding function fulfilling the invariant properties (a-e). Formally, we define the rounded container instance C^R_{con} as follows: For each container $c \in \mathcal{C}$ such that there exists an item i with $con(i) = c$, define a rectangle of height $h(c) = h_B$ and width $w(c) = \max\{w(i) \mid i \in I_L, con(i) = c\}$. Then, round each container width to the largest width in its group defined by R.

By choosing $h_B = 13/\epsilon^2$ and $k = \left\lfloor \frac{\epsilon}{4\omega h_B}\,\mathrm{SIZE}(I_L)\right\rfloor$ as parameters of the invariant, we get the following result:

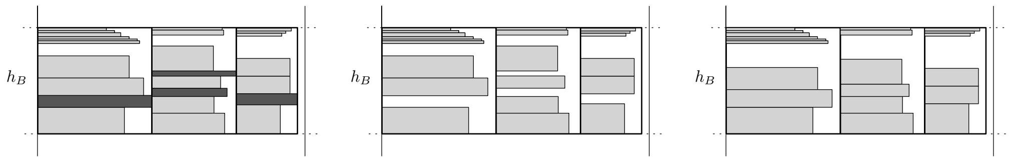
▶ **Lemma 2.3.** *Let C^R_{con} be the strip packing instance of rounded containers fulfilling all invariant properties from Definition 2.1. Assuming $\mathrm{SIZE}(I_L) \ge \frac{4\omega h_B}{\epsilon}(h_B+1)$, it holds that $\mathrm{OPT}(C^R_{con}) \le (1+4\epsilon)\,\mathrm{OPT}(I_L) + \mathcal{O}\left(1/\epsilon^4\right)$.*

Proof (Sketch). In [18] we give a detailed proof using a proof technique from [12]. For the sake of intuition, in this paper we only sketch the main arguments necessary to proof Lemma 2.3. The proof uses a nice combination of all invariant properties from Definition 2.1.

Intuitively, the goal is to show that by packing the containers C^R_{con} instead of the items I_L, we do not loose too much area in the packing. This can be shown formally by defining two sets of rectangles: Let $\hat{I}_L$ be the set of items in I_L where the width of each item from group g is set to the widest item in the group $right(g)$. Note that by invariant (b), the widths of items from $\hat{I}_L$ get rounded down. As the heights remain unchanged, it holds that $\mathrm{OPT}(\hat{I}_L) \le \mathrm{OPT}(I_L)$. Furthermore, let C_1 be the set of all container rectangles from C^R_{con}, except from the left- and right-most groups of each category l.

For the moment, assume that each container is filled up to the maximum filling height h_B. Therefore, we have a relation between $\hat{I}_L$ and C_1: Each stack of rounded-down rectangles from $\hat{I}_L$ corresponds with a container rectangle from C_1, namely with one of the group to the right. Therefore, packing C_1 instead of $\hat{I}_L$ is basically the same. By invariant (e), the total height of items in each group is bounded from below. Thus we can think of an average container assignment, in which each container is well-filled also in height. Therefore, the packing capacity of each container is used efficiently in height and width.

Finally, we have to argue that the containers dropped from C^R_{con} to obtain C_1 can be packed such that the total packing height increases only by a small term. By invariant (c–d),



(a) Set S_{out} (dark items) **(b)** Gaps after removal of S_{out} **(c)** After SINK

Figure 5 Operation SINK closes gaps during a SHIFT operation.

the left- and right-most groups of a category l have each at most $2^l k$ containers, all of width at most 2^{-l} by invariant (a). That is, $2k$ levels of height h_B are enough to place all residual containers in C^R_{con} not contained in C_1. By definition of k and ω, it follows that the additional packing height for the missing containers in C_1 is not more than $\epsilon\,\mathrm{SIZE}(I_L) \leq \epsilon\,\mathrm{OPT}(I_L)$. ◀

3 Shift Operation

So far, we introduced the packing structure and showed important characteristics of it. In this section we consider the online setting, where new items arrive and have to be integrated into the structure such that invariant properties (a-e) are maintained. In order to maintain (a-b) when inserting a new item i, a suitable group has to be found, defined as follows:

▶ **Definition 3.1** (Suitable group). For a group g, let $w_{min}(g)$ resp. $w_{max}(g)$ denote the width of an item with minimal resp. maximal width in ${I_L}^g$. Set $w_{min}(\mathit{left}((l, A, 0))) = \infty$ and $w_{max}(\mathit{right}((l, B, q(l, B)))) = 0$. Group $g = (l, X, r)$ is *suitable* for a new item i if $w(i) \in (2^{-(l+1)}, 2^{-l}]$, $w_{min}(\mathit{left}(g)) \geq w(i)$, and $w_{max}(\mathit{right}(g)) < w(i)$.

Basically, new items can be integrated into the container structure in two ways: They can be placed into new containers, or they can be placed into existing containers, where already packed items have to be removed possibly.

Since the first option occurs rather in special cases, in Section 3.1 we describe a simplified version of the SHIFT operation which inserts items via the second way. Note that in this case the number of containers remains unchanged and thus (c) and (d) are maintained anyway. Afterwards, we briefly describe the issue of new containers in the packing.

3.1 Shift Algorithm (simplified)

Algorithm 1 shows the (simplified) SHIFT operation. Suppose that S is a set of items to be inserted into the suitable group g. The easy case is when $h(g) + h(S)$ does not exceed the upper bound $(h_B - 1)K_g$ from invariant (e): Then, PLACE(g, S) in Line 3 packs each item in S into any container with sufficient small packing height[3] of this group. It can be easily seen that there must be such a container: Assume that item $i \in S$ can not be placed. Then, each of the K_g containers is filled with items of total height greater than $h_B - 1$. Thus, $h(g) + h(S) > K_g(h_B - 1)$, which contradicts (e).

However, the crucial point is that due to the insertion of S, the total height of items in g could exceed the upper bound from (e). In order to fulfill (e), items from g are removed. For this purpose, we choose the widest items from g, as they can be inserted into the group $\mathit{left}(g)$ while maintaining the sorting property (b). The function WIDESTITEMS$({I_L}^g \cup S, \Delta)$ in Line 5

[3] That is, the total height of items in this container plus the height of the new item does not exceed h_B.

Algorithm 1: Shift

```
Input : Group $g \in G$, Items $S \subset I_L$, suitable for $g$ according to Definition 3.1
1 $\Delta = h(g) + h(S) - (h_B - 1)K_g$
2 if $\Delta \le 0$ then                              // No violation of invariant (e)
3     Place($g$,$S$)
4 else
5     $S_{out}$ = WidestItems($I_L{}^g \cup S$, $\Delta$)
6     Remove $S_{out}$ from group
7     Sink($c_j$)                 // For all affected containers $c_j$
8     Place($g$,$S$)
9     Shift($left(g), S_{out}$)
```

returns a set of items $S_{out} \subseteq I_L{}^g \cup S$ s.t. $w(i) \ge w(i')$ for each $i \in S_{out}, i' \in (I_L{}^g \cup S) \setminus S_{out}$ and $h(S_{out}) \in [\Delta, \Delta + 1)$. Note that after removing the items S_{out}, gaps may occur in the packing. These have to be closed before new items can be placed, which is done by the operation Sink in Line 7 (see Figures 5a to 5c for an illustration). Now, there is enough room to place the items S in Line 8. The removed items get inserted into *left*(g) via a further Shift operation. If the group *left*(g) does not exist, one has to open a new container for the remaining items.

An important characteristic of Algorithm 1 is that it maintains all invariant properties. In the following we give a proof restricted to property (e), as this is somehow the most fundamental property.

▶ **Lemma 3.2.** *Suppose that invariant property (e) holds. After shifting items S into group g via Algorithm 1, invariant property (e) remains fulfilled.*

Proof. We show that the total height of items after the removal of S_{out} and insertion of S lies in the interval $[(h_B - 1)(K_g - 1), (h_B - 1)K_g]$. Since $h(S_{out}) \ge \Delta$, it holds $h(g) - h(S_{out}) + h(S) \le h(g) - \Delta + h(S) = h(g) - h(g) - h(S) + (h_B - 1)K_g + h(S) = (h_B - 1)K_g$. On the other side, $h(S_{out}) < \Delta + 1$ and thus $h(g) - h(S_{out}) + h(S) > h(g) - \Delta - 1 + h(S) = h(g) - h(g) - h(S) + (h_B - 1)K_g - 1 + h(S) = (h_B - 1)K_g - 1$. With $h_B \ge 2$ it follows that $(h_B - 1)K_g - 1 \ge (h_B - 1)(K_g - 1)$. Hence, property (e) is fulfilled. ◀

Since the set S_{out} is inserted via another shift operation into the next group, in general the insertion of an item i_t triggers a sequence of shift operations Shift(g^0, S^0), Shift(g^1, S^1), $\ldots$, Shift(g^d, S^d) with $S^0 = \{i_t\}$. Thereby, the total height of shifted items $h(S_{out})$ grows linearly in the position of the shift sequence, like the following lemma shows.

▶ **Lemma 3.3.** *Consider the above defined shift sequence and let S^j_{out} be the set S_{out} in the call* Shift(g^j, S^j)*. For any j with $0 \le j \le d$ it holds that $h(S^j_{out}) \le h(S^0) + j + 1$.*

Proof. Let Δ_j denote the value of Δ in the call Shift(g^j, S^j). First note that by invariant (e) $\Delta_j \le h(S^j)$ holds for each j . Further, the function WidestItems$(\cdot, \Delta_j)$ returns a set S^j_{out} with $h(S^j_{out}) < \Delta_j + 1$. For $j = 0$ it holds that $h(S^0_{out}) < \Delta_0 + 1 \le h(S^0) + 1$. Now suppose $h(S^j_{out}) \le h(S^0) + j + 1$ for some $j \ge 0$. Note that $S^{j+1} = S^j_{out}$, thus for the index $j + 1$ we have $h(S^{j+1}_{out}) < \Delta_{j+1} + 1 \le h(S^{j+1}) + 1 = h(S^j_{out}) + 1$. By assumption, $h(S^j_{out}) \le h(S^0) + j + 1$ and thus $h(S^{j+1}_{out}) \le h(S^0) + (j + 1) + 1$. ◀

Lemma 3.3 is particularly important to bound the amount of items arriving in the leftmost group. By choosing h_B appropriately, the remaining items fit into one additional container.

```
Algorithm 2: Insertion of a big item
  Input : Item $i_t \in I_L$
1 if SIZE$(I_L(t)) < \frac{4\omega h_B}{\epsilon}(h_B + 1)$ then                // Offline mode
2     Use offline algorithm
3 else                                                                          // Online mode
4     Find suitable group $g = (l, X, r)$ according to Definition 3.1
5     Shift($g, \{i_t\}$)
6     BlockBalancing
```

New containers. We already mentioned that most of the repacking happens inside existing containers and therefore new containers occur rather in special cases. However, note that these special cases are important: Items which have to be shifted out of group $(l, A, 0)$ can not be shifted further, as there is no group to the left (see Figures 2 and 4).

Therefore, we also have to deal with new containers in the container packing. Obviously, updating the level assignment such that new containers are placed in new levels is not enough to guarantee a good competitive ratio. Instead, a new level assignment has to be found, which maintains large parts of the existing assignment (in order to bound the repacking). Since this problem is closely related to an online bin packing problem, here we make use of the LP/ILP-techniques developed in [17]. For all technical details see [18].

3.2 Insertion Algorithm

Let $I_L(t) = \{i_1, i_2, \ldots, i_t\}$ denote the instance at time t. The insertion algorithm for big items, given in Algorithm 2, works in one of two modes: While SIZE$(I_L(t)) < \frac{4\omega h_B}{\epsilon}(h_B + 1)$, Algorithm 2 works in the *offline mode*. Here, an offline algorithm fulfilling all invariant properties repacks the whole instance each time a new item arrives. This is due to the fact that the operations modifying the LP-solutions require a minimum size of $I_L(t)$. As soon as SIZE$(I_L(t))$ is large enough, in the *online mode* the algorithm goes over to use Shift$(g, \{i_t\})$ to insert i_t into the suitable group g.

The last operation in Algorithm 2, denoted as BlockBalancing, adapts the total number of containers to the increasing value of SIZE$(I_L(t))$. Recall that by choice of the parameters (see Section 2.3), k depends on SIZE$(I_L(t))$ and thus increases over time. That is, at some point the parameter k changes to $k' = k + 1$. Obviously, we can not rebuild the whole container assignment to fulfill the new group sizes required by (c-d) according to the new parameter k'. Instead, the block structure (see Section 2.1) is exactly designed to deal with this situation: All groups of block A that satisfy invariant property (c) with parameter k satisfy (d) for parameter k', if they were in block B. In the procedure BlockBalancing groups are moved from block B to A parallel to the increasing fractional value of k. When block B is empty, groups from block A can be 'renamed' to block B groups. This way, (c-d) are fulfilled for the new parameter k' and the repacking is distributed among all time steps since the last parameter update. This technique was developed in [4]. For more details and a precise description of the operations see [18].

With respective results for Shift (including Lemma 3.2) and BlockBalancing, Algorithm 2 maintains all invariant properties. Furthermore, we can show that all operations modifying the LP/ILP-solutions of the level assignment return feasible solutions with the desired approximation guarantee. Therefore we obtain the following result:

▶ **Theorem 3.4.** *Algorithm 2 is an AFPTAS for the insertion of big items with asymptotic competitive ratio* $1 + \mathcal{O}(\epsilon)$.

4 Migration Analysis

It remains to analyze the migration factor of Algorithm 2. Recall the definition of the migration factor $\mu = \frac{\text{SIZE}(\mathit{Repacking}(t))}{\text{SIZE}(i_t)}$, where *Repacking*$(t)$ is the set of repacked items and i_t the item arriving at time t. Since in this extended abstract we focus on big items, the migration factor can be bound without amortization. First, note that in the offline mode of Algorithm 2, the repacking size is clearly bounded by $\text{SIZE}(I_L(t)) < \mathcal{O}\left(\frac{1}{\epsilon^5}\log\frac{1}{\epsilon}\right)$. The analysis for the online mode is quite involved since the operation Shift consists of several repacking steps performed in different groups. In the maximum shift sequence each group occurs once (see again Figure 2), thus the maximum number of shift operations can be at most the number of groups $|G|$. Again, one crucial argument is that $|G| \leq \mathcal{O}\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right)$ (see Lemma 2.2). We give a detailed analysis for the repacking of the shift operation in [18] and get eventually:

▶ **Lemma 4.1.** *The total repacking in a maximum shift sequence is at most* $\mathcal{O}\left(\frac{1}{\epsilon^7}\left(\log\frac{1}{\epsilon}\right)^2\right)$.

Recall that in the online mode of Algorithm 2 the procedure BlockBalancing performs repacking as well. However, it can be shown that its repacking size is dominated by the Shift part. Since big items have minimum size ϵ^2, we obtain the following corollary:

▶ **Corollary 4.2.** *Algorithm 2 has the migration factor* $\mu = \mathcal{O}\left(\frac{1}{\epsilon^9}\left(\log\frac{1}{\epsilon}\right)^2\right)$.

Acknowledgement. We would like to thank Marten Maack for helpful discussions on the lower bound (Section 1.3).

References

1 Brenda S. Baker, Edward G. Coffman, Jr, and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

2 Brenda S. Baker and Jerald S. Schwarz. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing*, 12(3):508–525, 1983.

3 János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science*, 440:1–13, 2012.

4 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 135–151, 2015.

5 Donna J. Brown, Brenda S. Baker, and Howard P. Katseff. Lower bounds for on-line two-dimensional packing algorithms. *Acta Informatica*, 18(2):207–225, 1982.

6 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 2017.

7 Edward G. Coffman, Jr, Michael R. Garey, David S. Johnson, and Robert E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

8 János Csirik and Gerhard J. Woeginger. Shelf algorithms for on-line strip packing. *Information Processing Letters*, 63(4):171–175, 1997.

9 Kurt Eisemann. The trim problem. *Management Science*, 3(3):279–284, 1957.

10 Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Mathematical Programming*, 119(1):33–49, 2009.

11 Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. Algorithms for the relaxed online bin-packing model. *SIAM Journal on Computing*, 30(5):1532–1551, 2000.

12 Xin Han, Kazuo Iwama, Deshi Ye, and Guochuan Zhang. Strip packing vs. bin packing. In *International Conference on Algorithmic Applications in Management (AAIM)*, pages 358–367. Springer, 2007.

13 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob Van Stee. A (5/3+ ε)-approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014.

14 Johann L. Hurink and Jacob J. Paulus. Online algorithm for parallel job scheduling and strip packing. In *International Workshop on Approximation and Online Algorithms (WAOA)*, pages 67–74. Springer, 2007.

15 Johann L. Hurink and Jacob J. Paulus. Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*, 36(1):51–56, 2008.

16 Csanád Imreh. Online strip packing with modifiable boxes. *Operations Research Letters*, 29(2):79–85, 2001.

17 Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 589–600. Springer, 2013.

18 Klaus Jansen, Kim-Manuel Klein, Maria Kosche, and Leon Ladewig. Online strip packing with polynomial migration. *CoRR*, abs/1706.04939, 2017. URL: `http://arxiv.org/abs/1706.04939`.

19 Berit Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9(5):433–452, 2006.

20 Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Foundations of Computer Science (FOCS)*, pages 312–320, Nov 1982.

21 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

22 Walter Kern and Jacob J. Paulus. A note on the lower bound for online strip packing. *Statistics and Computing*, 2009.

23 Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 15–1, 2004.

24 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

25 Steven S. Seiden. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

26 Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Mathematics of Operations Research*, 41(3):991–1021, 2016.

27 Christoph Steiger, Herbert Walder, Marco Platzner, and Lothar Thiele. Online scheduling and placement of real-time tasks to partially reconfigurable devices. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 224–225. IEEE, 2003.

28 Deshi Ye, Xin Han, and Guochuan Zhang. A note on online strip packing. *Journal of Combinatorial Optimization*, 17(4):417–423, 2009.

29 Deshi Ye, Xin Han, and Guochuan Zhang. Online multiple-strip packing. *Theoretical Computer Science*, 412(3):233–239, 2011.

30 Guosong Yu, Yanling Mao, and Jiaoliao Xiao. A new lower bound for online strip packing. *European Journal of Operational Research*, 250(3):754–759, 2016.

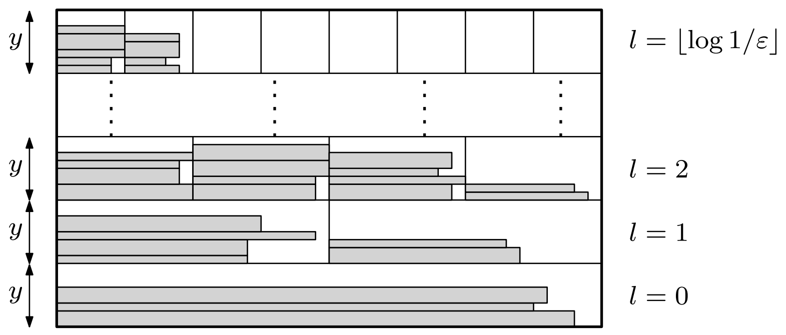


Figure 6 F-buffer contains 2^l slots of height y for each category l.

A Flat Items

We say an item i is *flat* if $w(i) \geq \epsilon$ and $h(i) < \epsilon$. The main difficulty of flat items becomes clear in the following scenario: Imagine that flat items of a group g are elements of $S_{out} = \textsc{WidestItems}(\cdot, \Delta)$ in a shifting process. Remember that generally each container, from which items are removed, has to be sinked (see Section 3.1), i. e. at most $|S_{out}|$ containers. In case of big items, due to their minimum height ϵ we get $|S_{out}| \leq \lfloor \Delta/\epsilon \rfloor$. In contrast, flat items can have an arbitrary small height and thus no such bound is possible. But $\textsc{Sink}$ on all K_g containers would lead to unbounded migration (since K_g depends on $\text{SIZE}(I_L)$).

Therefore, we aim for a special packing structure for flat items that avoids the above problem of sinking too many containers. Like shown in Figure 1a, flat items build a sorted stack at the top of the container such that the least wide item is placed at the top edge. Thereby, widest items can be removed from the container without leaving a gap.

To maintain the sorting, we introduce a buffer for flat items called *F-buffer*. It is located in a rectangular segment of width 1 and height ωy, somewhere in the packing, for some constant y. Note that the additional height for the F-buffer is bounded by $\omega y = \mathcal{O}((\log 1/\epsilon) y)$. The internal structure of the F-buffer is shown in Figure 6: For each category l, there are 2^l slots in one level of height y. Items can be placed in any slot of their category.

An incoming flat item may overflow the F-buffer, more precisely, the level of one category in the F-buffer. For this purpose, Algorithm 3 iterates over all groups $g_q, g_{q-1}, \ldots, g_0$ of this category, where g_q is the rightmost and g_0 the leftmost group[4]. For each group, the set S contains those items in the F-buffer for which the group is suitable. The set S is split into smaller subsets of total height at most 1, then each subset gets inserted via a single call of $\textsc{Shift}$.

Note that the concept of a 'buffered insertion' for small items, like in Algorithm 3, corresponds with the notion of amortized migration: While flat items can be placed in the F-buffer, no repacking is performed at all. We save this migration for a later time step, namely when the F-buffer is full. Then, all items from the F-buffer get inserted into the containers, maintaining the packing structure and resulting in an empty F-buffer.

[4] Note that the direction of the iterative shifting is crucial: Calling $\textsc{Shift}$ for a group g may reassign items in all groups left to g. Therefore, iterating from 'right to left' is necessary to guarantee that after shifting into group g, no group to the right of g is suitable for a remaining item in S. In other words, with this direction one shift call for each group is enough, which is in general not true for the direction 'left to right'.

Algorithm 3: Insertion of a flat item

```
Input: Flat item $i_t$ of category $l$
1  if $i_t$ can be placed in the F-buffer then
2      Place $i_t$ in the F-buffer
3  else
4      Let $B(l)$ be the set of items in the buffer slots of category $l$
5      for $j = q(l,A) + q(l,B) + 1, \dots, 0$ do
6          Let $g_j = \begin{cases} (l,A,j) & j \leq q(l,A) \\ (l,B,j-q(l,A)-1) & j > q(l,A) \end{cases}$
7          Let $S = \{i \in B(l) \mid g_j \text{ is suitable for } i\}$
8          Let $S_1, \dots, S_n$ be partition of $S$ with $h(S_r) \in (1-\epsilon, 1]$ for all $1 \leq r \leq n$.
9          for $r = 1, \dots, n$ do
10             Shift($g_j$,$S_r$)
11         Remove $S$ from $B(l)$
12     BlockBalancing
```

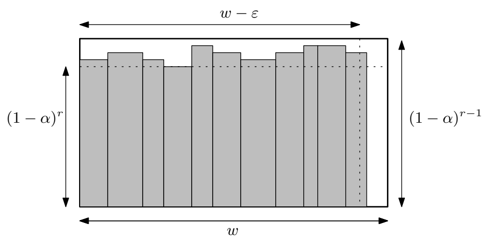


Figure 7 Shelf for narrow items of group r (dense).

B Narrow Items

We say an item i is *narrow* if $w(i) < \epsilon$. Narrow items can be packed efficiently if items of similar height are packed in a row. This is the concept of *shelf algorithms* introduced by Baker and Schwarz [2] which is described in the next subsection.

However, the goal is to integrate narrow items into the container packing introduced in Section 2. We show in Section B.2 how to fill gaps in the container packing with shelfs of narrow items. This leads to a modified first-fit-algorithm for narrow items with asymptotic approximation ratio of $1 + \mathcal{O}(\epsilon)$, as finally shown in Lemma B.2.

B.1 Shelf Packing

For a parameter $\alpha \in (0,1)$ item i belongs to *group* $r \in \mathbb{N} \setminus \{0\}$ if $h(i) \in [(1-\alpha)^r, (1-\alpha)^{r-1})$. Narrow items of group r are placed into a *shelf of group* r, which is a rectangle of height $(1-\alpha)^{r-1}$. Figure 7 shows a shelf for group r. Analogously to [2], we say a shelf of width w is *dense* when it contains items of total width greater than $w - \epsilon$ and *sparse* otherwise.

When the instance consists only of narrow items, the concept of shelf algorithms yields an online AFPTAS immediately. Consider the following first-fit shelf algorithm: Place an item of group r into the first shelf of group r where it fits, open a new shelf of group r only if necessary[5].

[5] Note that this simple algorithm works in the online setting since no sorting is necessary (in contrast to the NFDH algorithm [7], for example).

▶ **Lemma B.1.** *If I contains only narrow items, the shelf algorithm with parameter $\alpha = \frac{\epsilon^2}{1-\epsilon^2}$ yields a packing of height at most $(1+\epsilon)\,\mathrm{OPT}(I) + \mathcal{O}\left(1/\epsilon^4\right)$.*

Proof. For a group r, let $I_r = \{i \in I \mid h(i) \in [(1-\alpha)^r, (1-\alpha)^{r-1})\}$. Consider the packing obtained by the shelf algorithm and let β_r the number of shelfs of group r. Each dense shelf for group r contains items of size at least $(1-\alpha)^r(1-\epsilon)$, see Figure 7. Note that by the first-fit-principle, for each group at most one shelf is sparse. Thus there are at least $\beta_r - 1$ dense shelfs for each group r, hence $\mathrm{SIZE}(I_r) \geq (\beta_r - 1)(1-\alpha)^r(1-\epsilon)$, or equivalently

$$\beta_r \leq \mathrm{SIZE}(I_r)(1-\alpha)^{-r}(1-\epsilon)^{-1} + 1\,. \tag{3}$$

The packing consists of β_r shelfs of height $(1-\alpha)^{r-1}$ for each group r (set $\beta_r = 0$, if the group does not exist). Therefore, the packing height is:

$$\begin{aligned}
&\sum_{r=0}^{\infty} \beta_r (1-\alpha)^{r-1} \\
\leq &\sum_{r=0}^{\infty} \left(\mathrm{SIZE}(I_r)(1-\alpha)^{-r}(1-\epsilon)^{-1} + 1\right)(1-\alpha)^{r-1} && \text{eq. (3)}\\
= &\sum_{r=0}^{\infty} \mathrm{SIZE}(I_r)(1-\alpha)^{-1}(1-\epsilon)^{-1} + (1-\alpha)^{r-1} \\
= &\frac{1}{(1-\epsilon)(1-\alpha)} \sum_{r=0}^{\infty} \mathrm{SIZE}(I_r) + \sum_{r=0}^{\infty} (1-\alpha)^{r-1} \\
\leq &\frac{1}{(1-\epsilon)(1-\alpha)} \mathrm{SIZE}(I) + \frac{1}{\alpha - \alpha^2} && \text{Geometric series}\\
\leq &\frac{1}{(1-\epsilon)(1-\alpha)} \mathrm{OPT}(I) + \frac{1}{\alpha - \alpha^2} \\
= &(1+\epsilon)\,\mathrm{OPT}(I) + \mathcal{O}\left(\frac{1}{\epsilon^4}\right) && \text{Choice of } \alpha
\end{aligned}$$

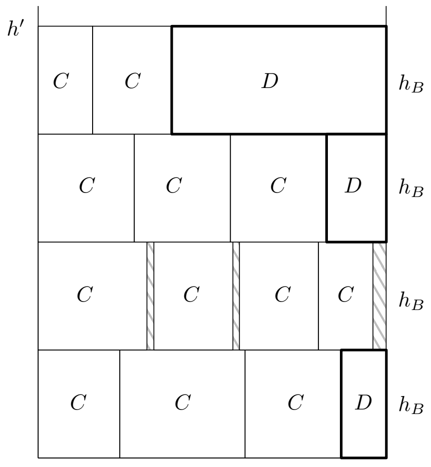
Note that the total height of sparse shelfs is bounded by a constant, even if the number of groups is unbounded. This follows by the geometric series:

$$\sum_{r=0}^{\infty} (1-\alpha)^{r-1} = \frac{1}{1-\alpha} \sum_{r=0}^{\infty} (1-\alpha)^r = \frac{1}{1-\alpha}\frac{1}{\alpha} = \frac{1}{\alpha - \alpha^2}$$ ◀

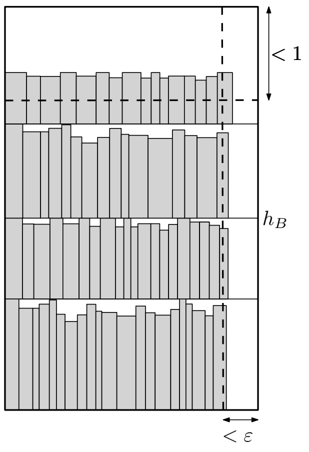
B.2 Filling Gaps in the Container Packing

As shown in the previous section, shelfs are a good way to pack narrow items efficiently. But before opening a new shelf that increases the packing height, we have to ensure that the existing packing is well-filled. Therefore, the idea is to fill gaps in the container packing with shelfs of narrow items first. Thereby, a *gap* is the rectangle of height h_B that fills the remaining width of a level. Only if no significant gaps exist, new shelfs are packed on top of the packing.

Figure 8a shows the packing structure of the strip on a high level: Here, all C-rectangles represent containers for big and flat items. If the total width of containers in a level is less than a threshold value (say $1 - \mathcal{O}(\epsilon)$), these containers get *aligned* such that the only gap occurs at the right end of a level. We call these gaps D-containers. Inside, each D-container is organized in shelfs of narrow items (see Figure 8b).



(a) Well-filled container packing



(b) D-container

Figure 8 D-containers are introduced to fill gaps in the container packing with shelfs.

Since the width of containers changes due to shift operations, without aligning a level could be fragmented such that no contiguous area can be used for a D-container. As aligning levels means further repacking, the insertion algorithm for narrow items makes use of a special buffer, similar to the case of flat items.

Finally, it has to be proven that inserting narrow items this way maintains the overall approximation guarantee. Note that the first-fit strategy for narrow items (sketched above), has two important properties: If the item can be placed in a gap, the packing height does not increase. Now suppose that an item gets placed in a new shelf on top of the packing. This only occurs, if the existing packing is well-filled, since no significant gaps were found. As a consequence of this important observation we get the following lemma (proven in [18]):

▶ **Lemma B.2.** *Let h' be the height of the container packing. The insertion of narrow items returns a packing of height h_{final}, such that $h_{final} \le \max\left\{h', (1+\epsilon')\,\mathrm{SIZE}(I) + \mathcal{O}\left(\frac{\omega}{\epsilon^3}\right)\right\}$, where $\epsilon' \in \mathcal{O}(\epsilon)$.*

Note that I denotes the set of all items (including big, flat, and narrow items). Lemma B.2 immediately implies that the final packing height is at most $(1+\mathcal{O}(\epsilon))\,\mathrm{OPT}(I)+\mathcal{O}(poly(1/\epsilon))$: We can use Lemma 2.3 to bound the height h' of the container packing and the fact that $\mathrm{SIZE}(I) \le \mathrm{OPT}(I)$.