Improved Online Load Balancing with Known Makespan

Martin Böhm
University of Wrocław, Poland

Matej Lieskovský
Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czechia

Sören Schmitt
Department of Mathematics, University of Siegen, Germany

Jiří Sgall
Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Prague, Czechia

Rob van Stee
Department of Mathematics, University of Siegen, Germany

Abstract

We break the barrier of 3/2 for the problem of online load balancing with known makespan, also known as bin stretching. In this problem, m identical machines and the optimal makespan are given. The load of a machine is the total size of all the jobs assigned to it and the makespan is the maximum load of all the machines. Jobs arrive online and the goal is to assign each job to a machine while staying within a small factor (the competitive ratio) of the optimal makespan.

We present an algorithm that maintains a competitive ratio of $139/93 < 1.495$ for sufficiently large values of m, improving the previous bound of 3/2. The value 3/2 represents a natural bound for this problem: as long as the online bins are of size at least 3/2 of the offline bin, all items that fit at least two times in an offline bin have two nice properties. They fit three times in an online bin and a single such item can be packed together with an item of any size in an online bin. These properties are now both lost, which means that putting even one job on a wrong machine can leave some job unassigned at the end. It also makes it harder to determine good thresholds for the item types. This was one of the main technical issues in getting below 3/2.

The analysis consists of an intricate mixture of size and weight arguments.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, bin stretching, bin packing

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.10

Category APPROX

Related Version *Full Version*: `https://arxiv.org/abs/2407.08376` [6]

Funding *Martin Böhm*: Research supported by National Science Centre in Poland under grant SONATA 2022/47/D/ST6/02864.
Matej Lieskovský: Partially supported by GAUK project 234723, and GA ČR project 24-10306S.
Jiří Sgall: Partially supported by GA ČR project 24-10306S.

1 Introduction

Online Load Balancing with Known Makespan is an online problem defined as follows. At the start of the input, the number m of machines is revealed, followed by a sequence of jobs with sizes in $[0, 1]$, arriving one by one. Each job needs to be assigned to a machine, and the load of a machine is the total size of the jobs assigned to it. The algorithm is guaranteed a priori that the entire sequence of jobs can be scheduled on m machines so that the makespan (the load of the most-loaded machine) is at most 1. The objective of the algorithm is to schedule the jobs on the machines as they arrive, minimizing the makespan R of the online schedule, which is allowed to be larger than 1. The value R is also known under the name *stretching factor*.

The problem was first introduced in 1998 by Azar and Regev [3, 4] under the name Online Bin Stretching, and studied intensively since [8, 9, 15, 17, 21]. Among its given applications is container repacking [7] and reallocation during a server upgrade. This scheduling problem shares its terminology and some algorithmic ideas with Online Bin Packing. The overarching goal of the research of Online Bin Stretching and other related problems over the last few decades is to learn how a small amount of additional knowledge ahead of time (such as knowledge of the makespan) impacts the best possible competitive ratio for the quintessential online problem Online Load Balancing [16].

To that end, another closely related problem is Online Load Balancing with Known Sum of Processing Times, where we have a guarantee that the total volume of jobs is at most m, but the optimum can be larger than 1. (e.g., if jobs larger than 1 appear in the input sequence). For comparison in Online Bin Stretching we have a guarantee on the makespan which is stronger, while in the classical Online Load Balancing problem we have no guarantee. Having information on the total volume of jobs or the makespan could be viewed as particular kinds of *advice* given to the online algorithm [10, 11, 24].

To answer the general question above quantitatively, the state of the art is the following. For Online Load Balancing, Fleischer and Wahl [13] presented a deterministic algorithm with competitive ratio approximately 1.92, and Rudin [25] showed that no deterministic algorithm can be better than 1.88-competitive. Kellerer et al. [18] showed that having a guarantee on the sum of processing times allows an approximately 1.585-competitive algorithm as m goes to infinity, matching the lower bound of Albers and Hellwig [2]. Finally, for Online Bin Stretching, Böhm et al. [8] presented an algorithm with stretching factor 3/2, and Azar and Regev [4] showed that no algorithm can have a stretching factor below 4/3.

Our contribution

We propose an online algorithm for Online Bin Stretching that is able to surpass the 3/2 threshold:

▶ **Theorem 1.1.** *For $m \geq 60000$ and for $\varepsilon = 1/31$, there exists an online algorithm for Online Bin Stretching with stretching factor $3/2 - \varepsilon/6 = 139/93 < 1.495$.*

For $\varepsilon = 1/62$ the algorithm works already for $m \geq 3300$. Our algorithm builds upon the main concepts of its immediate predecessors [15, 8], by keeping a portion of the bins empty until a later phase of the input, and by tracking combinatorial properties of the items using a *weight-based analysis*. Any feasible algorithm must follow this general structure. However, once the stretching factor is set below 3/2, new types of items appear which require great care to pack efficiently. See Figure 1 and the full version. The level of complexity of our algorithm as well as its analysis significantly surpasses the previously best-known results. For instance, it now becomes necessary to use new item types when we start to fill up previously used bins later in the algorithm, as most of the initial item types do not fit well in the remaining space. Achieving a ratio below 3/2 for *all* values of m seems to be much harder still, as we often have constantly many bins which are only half-full; only when m is large is the number of such bins negligible.

History and related work

The first results on Online Bin Stretching have appeared even before the introductory paper of Azar and Regev in 1998; a year before, Kellerer et al. [19] already discovered the matching lower and upper bound of $4/3$ for the case $m = 2$. Since the beginning, some research works focus on the stretching factor for general number of bins m, while others focus on the special cases for m small and fixed. One interesting property of Online Bin Stretching with fixed m is that both the best-known lower bounds [21] and some algorithms [22] were designed using a computer-aided approach based on the *Minimax* algorithm, as initially proposed by Gabay et al. [14].

For any value $m \geq 2$ a general lower bound of $4/3$ comes from Azar and Regev [4]. For $m = 3$, the best-known algorithm is by Böhm et al. [7]. The remaining lower and upper bounds for the range $3 \leq m \leq 8$, listed in the table below, were designed by multiple variants of computer-aided search; the results are by Böhm and Simon [9], Lhomme et al. [21] and Lieskovský [23].

m	3	4	5	6	7	8	≥ 9
Lower bound	1.365 [9]	1.357 [9]	1.357 [9]	1.363 [21]	1.363 [21]	1.363 [21]	$1.\overline{3}$ [4]
Upper bound	1.375 [7]	1.393 [23]	1.410 [23]	1.429 [23]	1.455 [23]	1.462 [23]	1.5 [8]

For general m, Böhm et al. [8] presented the so far best algorithm in 2017 which achieves stretching factor $3/2$; this result was preceded by a long sequence of steady improvements on the algorithmic front, among others by Kellerer and Kotov [17] and Gabay et al. [15]. Recently in [20], Lhomme et al. give first results for randomized algorithms. They show that for $m = 2$ there exists a $5/4$-competitive randomized algorithm that outperforms the optimal deterministic algorithm. Furthermore, they provide lower bounds for $2 \leq m \leq 4$ on the competitive ratio of randomized algorithms.

For some small fixed values of m, especially $m = 2$, also specialized problems related to Online Bin Stretching have been investigated previously; for example, Epstein [12] considered online bin stretching with two machines (bins) of uniformly related speed and Akaria and Epstein [1] considered online bin stretching on two bins with grade of service and migration.

2 Structure of the algorithm

From now on, as is common in the literature on Online Bin Stretching and because we are dealing with a packing problem, we refer to bins, levels of bins and items instead of machines, loads of machines and jobs, respectively. Our initial setting is that the offline optimum bins have size 1 and the bins usable by our algorithm have size $R = 3/2 - \varepsilon/6$.

We assume that the number of bins m is at least 60000. We scale the sizes of the bins such that an offline bin has size 12 and an online bin has size $18 - 2\varepsilon$. (We use 2ε here so that half of the size of an online bin is a more convenient value.) Our goal is to construct an algorithm which works for the largest possible value of ε. We will eventually set $\varepsilon = 1/31$, but for an easier understanding of the relationships between the various values we will mostly use symbolic calculations. Scaling the offline bin size to 12 allows us to work with near-integer type thresholds, which is convenient. After scaling, the total size of the jobs on input is at most $12m$.

Our algorithm uses the algorithms Best Fit and First Fit as subroutines. These algorithms work as follows. Both algorithms open a new bin if the item does not fit into any existing bin. Otherwise, Best Fit places an item in a bin where the item can still fit and that, after placement, leaves the least amount of remaining empty space in the bin. This means it uses the most-filled bin that can still accept the item. First Fit always places the item in the first bin in which it will fit, using the order in which it opened the bins. In this paper, we will sometimes fix the order in which the bins are to be used in advance, namely if these bins already contain some items. This means that we are applying First Fit to variable-sized "bins" (the empty spaces in the actual bins). We give a proof for the performance of First Fit on variable-sized bins which may be of independent interest.

▶ **Lemma 2.1.**[1] *Consider a set V of bins that is packed by First Fit of which at least the last $|V|-2$ bins contain at least k items. If $|V| \geq 3$, the total level of the bins in V is more than*

$$\frac{k|V|}{k+1}.$$

▶ **Lemma 2.2.** *For any set of v variable-sized bins that is packed using First Fit, the following property holds. If at least $k < v/2$ items are packed into each bin, the total size of all the items packed into these bins is at least*

$$\frac{k}{k+1}\sum_{j=k}^{v-k} s(j),$$

where the size of the j-th bin is denoted by $s(j)$. This even holds if the number of bins increases while First Fit is running (in this case v is the final number of bins).

Proof. Let the bins be sorted by the order of First-Fit.

We look at an $(k+1)$-tuple $(j, j+1, \ldots, j+k)$ with $1 \leq j \leq v-k$. Let α be the largest empty space of bins $j, \ldots, j+(k-1)$. The items in bin $j+k$ have size at least α. Bins $j, \ldots, j+(k-1)$ on the other hand are filled to at least $s(j)-\alpha, \ldots, s(j+(k-1))-\alpha$. We know that at least k items of size at least α are packed in bin $j+k$, so in these $k+1$ bins we have an overall load of at least $\sum_{i=j}^{j+(k-1)} s(i)$. Applying this bound for $j = 1, 2, \ldots$ we find guarantees for First-Fit of at least $\sum_{i=1}^{k} s(i) + \sum_{i=k+2}^{2k+1} s(i) + \ldots, \sum_{i=2}^{k+1} s(i) + \sum_{i=k+3}^{2k+2} s(i) + \ldots$, etc. Adding all these bounds gives

$$(k+1)\cdot FF \geq \sum_{i=1}^{k-1} i \cdot s(i) + \sum_{i=k}^{v-k} k \cdot s(i) + \sum_{i=v-(k-1)}^{v-1} (v-i)s(i) > k\sum_{i=k}^{v-k} s(i).$$ ◀

2.1 Item types

Our algorithm initially uses the following item types; once we start filling up the bins in the fill-up phase, it will be necessary to use different types because of the amount of space that will be left. We group some item types into supertypes. There are two intervals for small items, as these items are packed the same way. The three weighting functions are related to the three types of items that fit only once in an offline bin. Having three separate such types is a consequence of the existence of quarter items (see Figure 1).

[1] The simple proof of this lemma can be found, e.g., in [5].

Supertype	–			middle			dominant	
Type	small	quarter	small	nice	half	large	big	top
Maximum size	$3-3\varepsilon$	$4-4\varepsilon$	$5+\varepsilon$	$6-2\varepsilon$	$6+2\varepsilon$	$9-\varepsilon$	$10+6\varepsilon$	12
Weight w_{top}	0	1	1	2	2	2	3	4
Weight w_{big}	0	0	0	2	2	2	4	4
Weight w_{large}	0	0	0	0	2	4	4	4

We describe the ideas behind these type thresholds in detail in the full version of the paper. Here we describe some fundamental properties of the various (super-)types. See also Figure 1.

Dominant items fit only once in an online bin. Nice items fit twice in an offline bin and can be placed in an online bin while still leaving room for another item of any size. (These items are indeed in principle nice to pack, but we still need to be very careful with them.) Half and large items fit twice in an online bin, but a large item cannot be packed together with a half item in an offline bin (due to the thresholds $6-2\varepsilon$ and $6+2\varepsilon$), whereas two half items *may* fit together in an offline bin.

In our algorithm, we will pack small items only to a level of $6-6\varepsilon$ at the beginning to leave room for one top item or two half items. As described above, using First Fit guarantees that more than $4-4\varepsilon$ is packed in almost all bins that are packed like this. Of course we get the same guarantee for small items of size more than $4-4\varepsilon$, and this is what motivates the upper bound $4-4\varepsilon$ for quarter items. It is also the same guarantee that we will achieve on average for (a certain subset of) the bins with quarter items (some bins will contain two quarter items). A big item fits in an online bin with two quarter items, and this is the reason that the dominant items are divided into two types.

▶ **Definition 2.3.** *For a partial input $I_{partial}$, let the value* TopThreat *(resp.,* BigThreat*,* LargeThreat*) be the maximum number of top items (resp., big items, large items) in I_{future} so that $I_{partial} \cup I_{future}$ can be packed in m bins of size 12, and let* TopBlock *(resp.,* BigBlock*,* LargeBlock*) be the set of bins that contain more than $6-2\varepsilon$ (resp., $18-2\varepsilon-(10+6\varepsilon)=8-8\varepsilon, 9-\varepsilon$).*

For any packing of a partial input, we have TopThreat $\le$ BigThreat $\le$ LargeThreat and LargeBlock $\le$ BigBlock $\le$ TopBlock.

▶ **Lemma 2.4.** *For any feasible input I and weighting function $w \in \{w_{top}, w_{big}, w_{large}\}$, we have $w(I) \le 4m$. For any $k \ge 0$ and any partial input $I_{partial}$:*
- *if $w_{top}(I_{partial}) \ge 4k$, then* TopThreat $\le m-k$*,*
- *if $w_{big}(I_{partial}) \ge 4k$, then* BigThreat $\le m-k$*,*
- *if $w_{large}(I_{partial}) \ge 4k$, then* LargeThreat $\le m-k$*.*

Proof. The bound $w_{\text{large}}(I) \le 4m$ follows from the type thresholds (a large and a half item do not fit together in an offline bin). For the other two weighting functions, note that for an item i of type j, the weight $w_{\text{top}}(i) = \lfloor \frac{5}{12} s_j \rfloor$ and $w_{\text{big}}(i) = 2(\lfloor \frac{3}{12} s_j \rfloor)$, where s_j is the infimum size of an item of type j (where the small items are split into two separate types for this calculation, one for each range of small items). Intuitively, w_{top} counts the number of items larger than $\frac{12}{5}$, that is, items that fit at most four times in an offline bin. Similarly, w_{big} counts items larger than $\frac{12}{3}$, and multiplies the result by two. The bounds $w_{\text{top}}(I) \le 4m$ and $w_{\text{big}}(I) \le 4m$ follow. ◀

The following invariant is a necessary property of any feasible algorithm and we will maintain it and other invariants throughout the processing of the input.

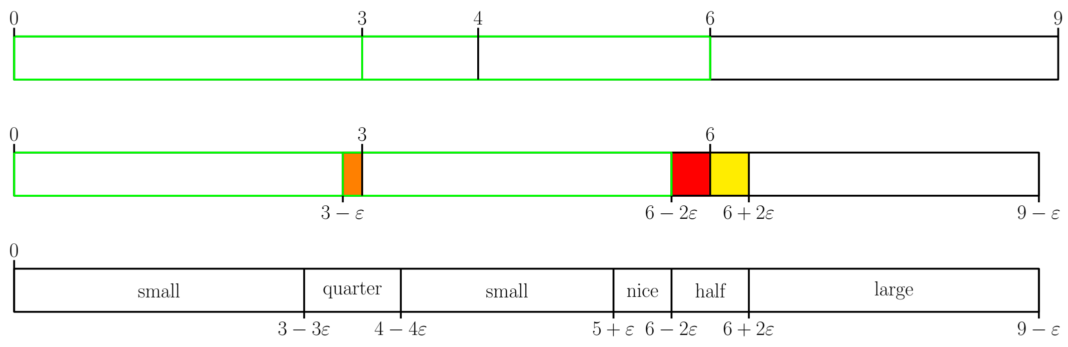

Figure 1 (Sketch, using $\varepsilon = 1/6$) A comparison of the important thresholds for an algorithm with competitive ratio 3/2 (top) and an algorithm with competitive ratio strictly less than 3/2 (middle). The thresholds our algorithm uses are displayed at the bottom. The offline bin size is scaled to be 12, so all items in the input have size at most 12. The green box indicates (half) the difference between the online and offline bin size. In the top figure, 6 is also a point where the amounts that can be packed in a bin change, both online and offline.
We immediately see that in the middle figure items exist which did not exist before (red); for a competitive ratio of 3/2, the online algorithm can pack more items per bin for *all* items smaller than 9. Moreover, items in the orange range can block some items of maximal size from being packed in the same bin, if we pack two such items in one bin. Finally, the fact that the red range exists means that items just larger than this (yellow) also need to be packed more carefully than before.

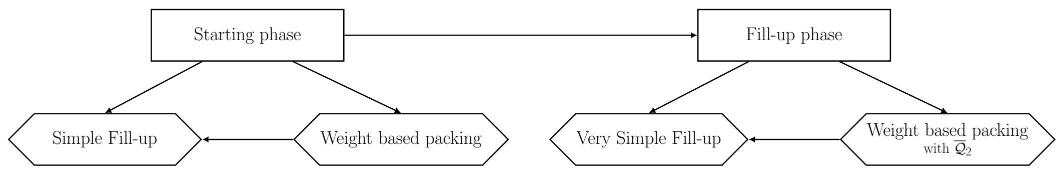

Figure 2 An overview of the phases and states.

▶ **Invariant 2.5.** *We have* TopThreat $\leq m -$ TopBlock *and* BigThreat $\leq m -$ BigBlock*.*

We will *not* be able to maintain LargeThreat $\leq m -$ LargeBlock throughout the algorithm (not even in the starting phase). However, fortunately large items can be placed twice in an online bin. Since these items can have size up to $9-\varepsilon$, a bin must be completely empty in order to guarantee that two large items may be packed in it.

2.2 Phases and states

In the starting phase, we use bins one by one, while staying below a level of $6-2\varepsilon$ unless there is a very good reason not to do so. If many relatively large items arrive, we may reach a state where it is sufficient to use First Fit for all remaining items (Simple Fill-Up) or where we know by weight that all items can be packed (Weight-based packing). Otherwise, we will eventually go to the Fill-up phase, where we start filling up the bins that previously received less than $6-2\varepsilon$ (or up to $8-8\varepsilon$ in the case of bins with two quarter items). In this phase we will eventually also reach a state where we know that the remaining input can be packed, either by size or by a weight argument.

The starting phase

From the lens of a single bin, our algorithm typically either packs items until a bin is full – which is typical for bins containing a single item type, such as the middle items – or it packs them only up to a level of $6 - 6\varepsilon$, particularly for items of size at most $6 - 6\varepsilon$.

However, as we have already seen, quarter items do not fit in this framework. On the one hand, we need to avoid packing many quarter items alone in bins (bad packing guarantee) while on the other hand, we also cannot pack too many quarter items in pairs in bins that do not yet contain anything else: that could block top items from being packed (Invariant 2.5 would be violated).

Ideally, we would like to pack items as follows:

- top items or pairs of half items with small items
- big items with quarter items
- large items in pairs
- nice items three per bin

In this way, all bins would have a weight of at least 4 in w_{top} and w_{big} and they would also all be more than 12 full (except for bins that contain one big item and one quarter item and bins that contain a top item/two half items smaller than 6 and not enough small items). There are several problems in using these methods, however:

- For bins that are planned to contain items of two different types, or two items of one type, it is not known whether the second type or item will ever arrive.
- Packing large items and smaller middle items into separate bins can easily lead to instances that cannot be packed (if there are two bins with single middle items that fit together in an offline bin, and then many top items arrive).

We can work around the first problem by changing our packing methods after a certain number of bins have received items of only one type, in particular if many small or quarter items arrive. Basically, our algorithm will first aim to reach the ideal packing described above. When sufficient volume has been packed, we go back and start filling up the already used bins. This is the fill-up phase of our algorithm.

The second problem requires us to be very careful with nice items in particular, since some nice items fit with some large items in an offline bin. Packing nice items three per bin in dedicated bins will be fine. However, we cannot afford to do this already starting from the very first bin with nice items, as there could also be a bin with one half item and another bin with one large item at the same time, blocking too many bins for top items so that the algorithm fails. Fortunately, a bin with only a nice item can still receive an item of any other type, so we will pack one nice item alone before starting to pack them three per bin from the second bin onwards. We still need to be very careful if both half and nice items arrive.

Good situations and the fill-up phase

We may be fortunate and reach a situation where many bins are filled to (significantly) more than 12. In this case it will be sufficient to pack the remaining items by essentially using First Fit. This is one example of a *good situation*. This is our term for a configuration which ensures that all remaining items can be packed, usually by using a very simple algorithm. This one is called the **First Fit case**.

It may also happen that many relatively large items arrive early. In this case we may reach a state where we know that a small or quarter item will never need to be packed into an empty bin anymore, because they are packed in existing bins first and we would reach the

First Fit case before using an empty bin. To ensure that the algorithm does succeed in all cases, even if all bins receive items, we will always use Best Fit as last resort for any item (after exhausting all other rules and all empty bins). We call this the **Rule of last resort**.

If many bins contain items but not enough of them contain a total size of more than 12 or sufficiently large items, it becomes important whether there exist bins that contain only small items or only single quarter items. If that is the case, we will go to the fill-up phase, in which we start filling up the nonempty bins using different item types. Otherwise, we will remain in the starting phase and we will eventually reach a good situation or the input will end.

The $(9 - \varepsilon)$-guarantee

We need to determine when exactly it is safe to start filling up bins in which we have already packed some items, without failing for instance to the threat of top items. To be precise, once we start filling up bins, we need a guarantee that this *remains* feasible no matter what the remaining input is. This will certainly require us to pack a sufficient total size in each bin that we fill up, as we always need to maintain Invariant 2.5.

Our cutoff for starting to fill up bins will be the point at which we know for certain that the future number of *big* items is (and will remain!) strictly smaller than the number of bins in which big items can still be packed (so, BigThreat $< m -$ BigBlock). There are in principle two ways by which we can know this: by considering weight and by considering volume. The problem with using a weight-based guarantee is that for instance small items can start arriving, which do not have weight. If we start filling up bins using small items, we can soon reach a point where the weight-based bound for BigThreat has not changed, but BigBlock has increased and we fail when many big items arrive.

We therefore use a volume-based bound. We need to be careful also here. Suppose that already $2m/3$ bins contain small items, and each such bin has a level in the range $(4 - 4\varepsilon, 6 - 6\varepsilon]$. Now suppose that many big items start arriving one by one. These big items do *not* bring us really closer to the point where we can safely start filling up the nonempty bins, because every time that we pack a big item BigThreat decreases by 1 and $m -$ BigBlock decreases by 1. Similarly, top items bring us only slowly closer to this point (since they are slightly larger than big items).

We will start the fill-up phase once we know the so-called **$(9 - \varepsilon)$-guarantee** holds:

$$\boxed{\text{Whenever new items of total size } 9 - \varepsilon \text{ arrive, BigThreat decreases by at least 1.}}$$

Having the $(9 - \varepsilon)$-guarantee essentially ensures that packing $9 - \varepsilon$ per bin is sufficient to maintain Invariant 2.5, although the problem of m large items arriving remains and needs to be dealt with separately. Maintaining this average is not at all straightforward, since we also have to make sure not to use too many *empty* bins too early, in order to pack as many pairs of large items into them as possible.

We present a very careful method of filling the nonempty bins which takes care to use the remaining space in those bins as efficiently as possible, using new item types which are tailored to the remaining space. This method consists of several stages.

2.3 Bin types

During the execution of the algorithm, each bin in the instance will be assigned a specific type. Sets of bins of a certain (sub)type are denoted typically by script letters (possibly with an index). We define six main types of bins. We use the corresponding lower case letters to refer to numbers of bins of a type: for instance, $\ell = |\mathcal{L}|$ and $\delta = |\Delta|$.

$\mathcal{E}$ Empty bins.

$\mathcal{L}$ Large-complete bins. This is a set of bins that reduce LargeThreat; more generally, they reduce the number of items with weight that can still arrive. Specifically, the number of large items that can still arrive will always be at most $m - \ell$, and the total weight that can still arrive will be at most $4(m - \ell)$. Since items without weight can still arrive however, and bins in $\mathcal{L}$ do not necessarily contain at least 12, large-complete bins do still accept items. A formal definition of these bins follows below.

$\mathcal{S}$ Bins with only *small* items. At most $6 - 6\varepsilon$ of small items is packed in each such bin.

Δ At most four bins that contain two nice items or a single middle item and maybe some other items. See below for more details.

$\mathcal{Q}$ Bins that are not in Δ in which the first item (or the second, if the bin was previously in Δ) is a quarter item and that are unmatched. (The algorithm sometimes matches some bins in $\mathcal{Q}$; these bins are then moved to a special subset $\mathcal{Q}_{\text{match}}$.)

$\mathcal{N}$ Bins in which the first two items are nice items and the third item is nice or half.

Bins started by nice items are filled to triples of nice items in the ideal packing and kept separate to achieve this; these bins can become large-complete upon receiving a dominant item. With this large-scale picture in mind, the large-complete bins are defined as follows. These bins require a careful definition because nice items may exist.

▶ **Definition 2.6.** *A bin is called* large-complete *if it satisfies all of the following conditions.*

- *it has $w_{big} \geq 4$,*
- *it contains an item larger than* 6 *or two items larger than* $6 - 2\varepsilon$*,*
- *the bin was never in $\mathcal{Q}$.*

It can be seen that each bin with $w_{\text{big}} \geq 4$ has a big item or $w_{\text{top}} \geq 4$. A large-complete bin does not necessarily contain a large item or a dominant item. The first condition ensures that these bins contain as much weight as any offline bin. The second condition implies that LargeThreat $\leq m - \ell$ at all times. Note that this does not follow from the first condition alone, as a bin could contain two nice items, and a nice and a large item may fit together in an offline bin.

The set Δ contains at most four exceptional bins used for careful handling of middle items. Each of these bins will be created explicitly in our algorithm if they are needed. There are the following four kinds of bins in Δ.

$\Delta_{\textbf{large}}$ one bin that contains a single large item, nothing else.

$\Delta_{\textbf{half}}$ one bin that contains a single half item and possibly small items of total size at most $6 - 6\varepsilon$ (notation Δ^S_{half}) or a quarter item (notation Δ^Q_{half}), nothing else. If the bin contains *only* a half item we call it $\overline{\Delta}_{\text{half}}$, else Δ^+_{half}.

$\Delta_{\textbf{nice,1}}$ at most two bins that contain one nice item and nothing else.

$\Delta_{\textbf{nice,2}}$ one bin that contains two nice items and nothing else.

Our algorithm will use the so-called **nice rule** as long as possible: do not pack nice items into $\Delta_{\text{large}} \cup \Delta_{\text{half}}$ and do not pack half or large items into $\Delta_{\text{nice},1}$. This rule ensures that nice items get packed into dedicated bins as much as possible (three per bin) so that we gain on these items (both by weight and by packed size per bin) compared to the optimal packing. This in turn ensures that nice items will hardly occur in inputs that are important for our analysis; see the weighting function w_{large}.

▶ **Definition 2.7.** *A bin is in $\mathcal{Q}$ if it satisfies the following properties:*

- *The first item is a quarter item,*
- *The bin is not in Δ_{half},*
- *The bin has not been matched (see algorithm).*

Additionally, a bin that was in $\overline{\Delta}_{half}$ and then received a quarter item and finally another half or larger item is also in $\mathcal{Q}$ as long as it has not been matched.

We define the subset of $\mathcal{Q}$ of bins in which the first two items are quarter items by $\mathcal{Q}_2$, and $\mathcal{Q}_1 := \mathcal{Q} \setminus \mathcal{Q}_2$. A bin in $\mathcal{Q}_1$ may leave $\mathcal{Q}$ (and $\mathcal{Q}_1$) by receiving a half item (it enters Δ_{half}); such a bin may later rejoin $\mathcal{Q}$ by receiving a half or larger item. Bins in $\mathcal{Q}$ may also leave $\mathcal{Q}$ permanently by being matched (two bins in $\mathcal{Q}_1$ to one bin in $\mathcal{Q}_2$).

Instead of using the partition $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$, we will also consider the useful partition of $\mathcal{Q}$ in the table below. (Some of these subsets may be empty.)

Bin type	Conditions on contents
$\overline{\mathcal{Q}}_1$	A single quarter item, nothing else
$\overline{\mathcal{Q}}_2$	Two quarter items, nothing else
$\mathcal{Q}_{1,\text{big}}$	First item is a quarter item, second item is big
$\mathcal{Q}_5$	First item is a quarter item, $w_{\text{big}} \geq 4$, bin is not in $\mathcal{Q}_{1,\text{big}}$
	Or: The first three items are (in this order) half, quarter, half or larger

We let $\overline{\mathcal{Q}} = \overline{\mathcal{Q}}_1 \cup \overline{\mathcal{Q}}_2$. Bins in $\mathcal{Q}_5$ can be in $\mathcal{Q}_1$ (for instance, bins with a top item) or in $\mathcal{Q}_2$; we keep track of their membership via the sets $\mathcal{Q}_{1,5} := \mathcal{Q}_5 \cap \mathcal{Q}_1$ and $\mathcal{Q}_{2,5} := \mathcal{Q}_5 \cap \mathcal{Q}_2$. All bins in $\mathcal{Q}_5$ will have w_{top}-weight 5 (or more), explaining the name $\mathcal{Q}_5$. For comparison, bins in $\mathcal{Q}_1$ have w_{top}-weight at least 1 and bins in $\mathcal{Q}_2$ have w_{top}-weight at least 2.

Bins in $\mathcal{Q}_{1,\text{big}}$ may get matched (pairwise) to bins in $\mathcal{Q}_{2,5}$; this is explained in the algorithm (Step 3). The set of matched bins is denoted by $\mathcal{Q}_{\text{match}}$.

Since the first two items in each bin in $\mathcal{Q}_2$ are quarter items, we have $\mathcal{Q}_2 \cap \mathcal{Q}_{1,\text{big}} = \emptyset$. We use the membership of bins in $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to keep track of the distribution of quarter items in the non-large-complete bins. In our proofs, we will assign weight from the quarter items in $\mathcal{Q}_1$ to bins in $\mathcal{Q}_2$ on the one hand (so we need sufficiently many bins in $\mathcal{Q}_1$) and assign volume from bins in $\mathcal{Q}_2$ to bins in $\mathcal{Q}_1$ on the other hand (so we need sufficiently many bins in $\mathcal{Q}_2$). The separation from large-complete bins and the separation of $\mathcal{Q}_{1,5}$ and $\mathcal{Q}_{2,5}$ will help us maintain an almost fixed ratio $q_1 : q_2$. Because of various half-full bins, we will need some additional bins in $\mathcal{Q}_1$ (at most 15 in the fill-up phase) before starting to create bins in $\mathcal{Q}_2$. A bin in $\overline{\mathcal{Q}}_1$ that receives a half item leaves $\mathcal{Q}$ and enters Δ_{half} (and Δ^{Q}_{half}). If it later receives another half item or a large, it returns to $\mathcal{Q}$, namely $\mathcal{Q}_{1,5}$, or moves to $\mathcal{L}$. Summarizing, we have the following disjoint unions.

$$\mathcal{Q}_1 = \overline{\mathcal{Q}}_1 \cup \mathcal{Q}_{1,\text{big}} \cup \mathcal{Q}_{1,5} \tag{1}$$

$$\mathcal{Q}_2 = \overline{\mathcal{Q}}_2 \cup \mathcal{Q}_{2,5}. \tag{2}$$

We define the set of *complete* bins $\mathcal{C}$ as the set of bins that from the point of view of the algorithm (and the analysis) do not need to receive any specific items, as follows:

$$\mathcal{C} := \mathcal{L} \cup \mathcal{Q}_5 \cup \mathcal{Q}_{\text{match}} \cup \mathcal{N}.$$

Into these bins, any item may be packed. Finally, the unmatched nonempty bins that are not large-complete are called *regular* (set $\mathcal{R}$). We have

$$\mathcal{R} = \mathcal{S} \cup \mathcal{Q} \cup \mathcal{N} \cup \Delta = \mathcal{S} \cup \overline{\mathcal{Q}}_1 \cup \overline{\mathcal{Q}}_2 \cup \mathcal{Q}_{1,\text{big}} \cup \mathcal{Q}_5 \cup \mathcal{N} \cup \Delta. \tag{3}$$

At all times, each bin is in exactly one of the sets $\mathcal{R}, \mathcal{Q}_{\text{match}}, \mathcal{L}, \mathcal{E}$.

We will show eventually that in the starting phase, some bins remain empty or we can guarantee that all remaining items can be packed (possibly using different methods). However, the partitioning of the sets shown here remains valid even after we run out of empty bins apart from the fact that some bins may contain some items that do not belong there; for instance, there could be some small items packed into $\overline{\mathcal{Q}}_2$. Also, after we run out of empty bins the nice rule may be violated. We will maintain the following invariant.

▶ **Invariant 2.8.** *There is never a bin in $\overline{\mathcal{Q}}_1 \cup \overline{\mathcal{Q}}_2$ at the same time as a bin with a big item as its only item with weight.*

▶ **Invariant 2.9.** *We have* LargeThreat $\leq m - (c - n)$ *as long as the nice rule is followed.*

▶ **Lemma 2.10.** *Invariant 2.9 holds for any packing of items.*

Proof. As long as the nice rule is followed, all complete bins except for the ones in $\mathcal{N}$ contain two half items or an item larger than 6 and have $w_{\text{big}} \geq 4$. ◀

From this bound it can be seen that the possible existence of bins in $\mathcal{N}$ force us to keep bins empty for *pairs* of large items, since we cannot ensure LargeThreat $\leq m - c$.

2.4 Proof overview

The present version omits essentially all of the proofs. Here we merely give an overview. The proof begins with some initial observations regarding how many bins there can be of different types and how much they contain. We then focus on the set $\mathcal{Q}$ and prove that up to an additive constant, $2q_2 = q_1$ throughout the starting phase. The (almost) fixed ratio $q_2 : q_1$ is used to help show Invariant 2.5 for top items and to show a packing guarantee for $\mathcal{Q}$. There will be constantly many bins that do not satisfy our packing guarantees, these bins will be in a set $\mathcal{X}$.

In the starting phase, either some bins remain empty, $\overline{q}_2 > 0$, or all items get packed. It turns out that Invariant 2.5 is maintained as long as we do not use the rule of last resort (essentially, as long as some bins are empty). There exist so-called good situations in which we can guarantee that all remaining items can be packed (possibly using a different algorithm). We show that Invariant 2.5 is maintained in the entire starting phase or we reach a good situation. More generally, the algorithm does not fail in the starting phase. We find that packing $9 - \varepsilon$ additionally in each non-complete bin in the fill-up phase is enough to maintain Invariant 2.5 in the fill-up phase as well.

To analyze the fill-up phase, we first consider some simple cases (essentially, new good situations). We then continue by showing that the algorithm does not fail in the first three stages of the fill-up phase. Linear programs are used to show that the algorithm does not fail in the fill-up phase.

3 Algorithm in the starting phase

Whenever the algorithm uses or attempts to use a set A to pack an item in the following description, we use First Fit on the bins in A, unless otherwise stated. The notation $A \to B$ means that a bin in the set A moves to B by receiving an item of the current type.

Step 1: Using and creating complete bins Try the following in this order.

- for half and large items: $\Delta_{\text{nice},2} \to \mathcal{N} \subseteq \mathcal{C}$,
- for non-small items: $\mathcal{Q}_{1,\text{big}} \to \mathcal{Q}_{1,5} \subseteq \mathcal{C}$
- use a complete bin (a bin in $\mathcal{C} = \mathcal{L} \cup \mathcal{Q}_5 \cup \mathcal{Q}_{\text{match}} \cup \mathcal{N}$).

- create a complete bin if this does not violate the nice rule (page 9).
  First try the bins $\overline{\mathcal{Q}}_2, \overline{\mathcal{Q}}_1, \Delta^Q_{\text{half}}$ in this order. Among other bins, use Best Fit to create a bin in $\mathcal{L}$, but do not pack a half item into Δ_{large} (yet).[2]

Step 2: Packing rules for each item type If an item is not packed yet, we apply the following rules depending on the item type.

Small: First Fit on bins in $\mathcal{S} \cup \Delta^S_{\text{half}}$ while packing at most $6 - 6\varepsilon$ of small items in each bin, $\overline{\Delta}_{\text{half}} \to \Delta^S_{\text{half}}$, $\mathcal{E} \to \mathcal{S}$.
Quarter: If $|\mathcal{Q}_1| + \delta^Q_{\text{half}} \geq 2|\mathcal{Q}_2| + 15$ then $\overline{\mathcal{Q}}_1 \to \overline{\mathcal{Q}}_2$, else $\overline{\Delta}_{\text{half}} \to \Delta^Q_{\text{half}}$, $\mathcal{E} \to \overline{\mathcal{Q}}_1$.
Nice: $\Delta_{\text{nice},2} \to \mathcal{N}$, if $\delta_{\text{nice},1} = 2$ then $\Delta_{\text{nice},1} \to \Delta_{\text{nice},2}$, $\mathcal{E} \to \Delta_{\text{nice},1}$.
Half: Best Fit on bins in $\mathcal{S} \cup \overline{\mathcal{Q}}_1 \to \Delta_{\text{half}}, \Delta_{\text{large}} \to \mathcal{L}, \mathcal{E} \to \Delta_{\text{half}}$.
Large: $\mathcal{E} \to \Delta_{\text{large}}$.
Dominant: Always packed in Step 1.

Rule of last resort If some item cannot be packed according to these rules, which can only happen after we run out of empty bins, we use Best Fit for this item, except that we still follow the nice rule as long as possible. If the nice rule has already been violated, we simply use Best Fit. For future items we still use the packing rules above first.

Step 3: Matching rule This step minimizes the number of bins in $\mathcal{Q}_{1,\text{big}}$.
If $|\mathcal{Q}_{1,\text{big}}| \geq 2$ and there is a bin in $\mathcal{Q}_{2,5}$, two bins in $\mathcal{Q}_{1,\text{big}}$ are matched to a bin in $\mathcal{Q}_{2,5}$ and all three bins are moved from $\mathcal{Q}$ to $\mathcal{Q}_{\text{match}}$.

Step 4: Swapping rule Each time that a new bin $\bar{b}$ in $\overline{\mathcal{Q}}_1$ is created, if there exists a large-complete bin b with a big item but no other items with weight (such a bin must contain also other items, or we would not have created $\bar{b}$), we virtually swap some items. That is, the bin $\bar{b}$ is treated as a bin in $\mathcal{S}$ from now on, and the bin b supposedly contains a big item and a quarter item. The quarter item is not considered to be the first item in b, so b is not in $\mathcal{Q}$. This ensures that Invariant 2.8 is maintained.

The swapping rule ensures that big items can be packed together with small items without violating Invariant 2.8 even if quarter items arrive later. Whenever the swapping rule is applied on two bins $\bar{b} \in \overline{\mathcal{Q}}_1$ and $b \in \mathcal{L}$, the total size packed into these bins is more than $18 - 2\varepsilon$ at this point (else $\bar{b}$ would not have been opened). If the bin $\bar{b}$ contains less than $4 - 4\varepsilon$ we reassign volume such that the bin $\bar{b}$ ends up with exactly $4 - 4\varepsilon$. We see that more than $18 - 2\varepsilon - (4 - 4\varepsilon) = 14 + 2\varepsilon$ remains for the bin b. This just means that $\bar{b}$ possibly has slightly more space for additional items than the algorithm calculates with (because it views $\bar{b}$ as containing the small items that were in b).

The large-complete bins used by the swapping rule differ from the other bins in $\mathcal{L}$ only in that they are not used to pack any future item. That is, we ignore such bins in Step 1 (this is not written explicitly in the algorithm; it seemed cleaner to explain this here).

Transitioning to the fill-up phase

In general, the transition from the starting phase to the fill-up phase happens once the $(9 - \varepsilon)$-guarantee starts to hold. This is roughly speaking after packing an average of $3 + \varepsilon$ on the (non-complete) regular and empty bins and 12 on the complete bins. More precisely

[2] If $\mathcal{S} \cup \overline{\mathcal{Q}}_1 \neq \emptyset$, we prefer packing half items there rather than in Δ_{large}, because this improves certain packing guarantees. (If $\mathcal{S} \cup \overline{\mathcal{Q}}_1 \neq \emptyset$, Δ_{half} exists and a large item arrives, we will have that $\Delta_{\text{half}} = \Delta^S_{\text{half}}$ which already improves the guarantee.)

after packing $T_1 := (3+\varepsilon)m + (9-\varepsilon)(c + q_{1,\text{big}} + 14)$. We can guarantee that the algorithm keeps at least roughly $\mathcal{E}_0 := \frac{1-5\varepsilon}{4-4\varepsilon}(m - \ell - q_{\text{match}}) + \frac{2+8\varepsilon}{4-4\varepsilon}n + \frac{1-5\varepsilon}{4-4\varepsilon}q_{\text{match}} + \frac{4\varepsilon}{4-4\varepsilon}\ell - 48$ bins empty when reaching the packing threshold. For more details see the full version.

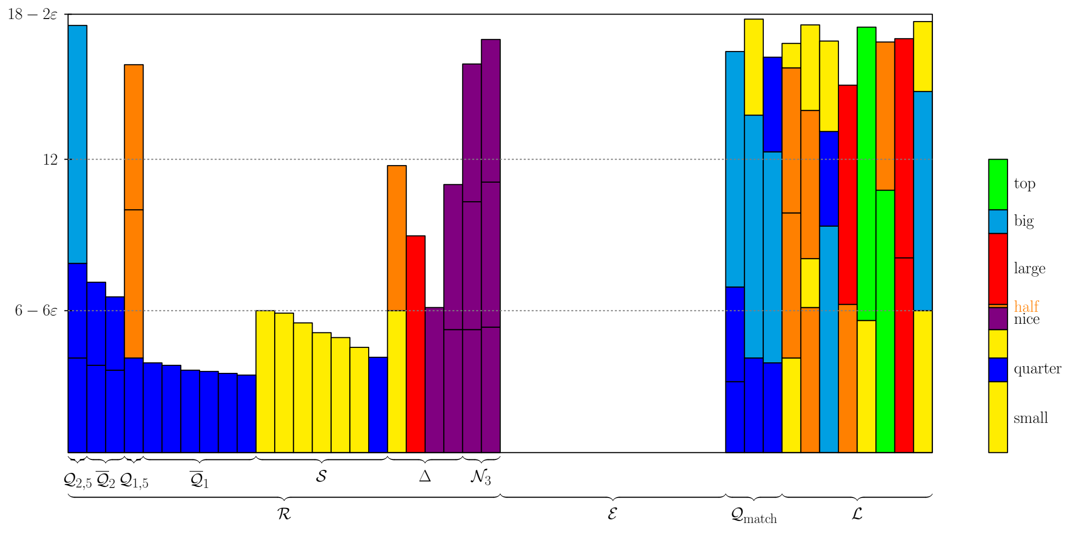

Figure 3 (Sketch) Overview of bins in the starting phase. The three bins in $\mathcal{Q}_{\text{match}}$ were moved there by the matching rule. The second quarter item in the rightmost bin in $\mathcal{Q}_{\text{match}}$ arrived there when the bins were already in $\mathcal{Q}_{\text{match}}$. The swapping rule was applied to the rightmost bin in $\mathcal{S}$ and the rightmost bin in $\mathcal{L}$. The small items on top of the big item in the rightmost bin arrived before the swapping rule was applied. For visual clarity we have left out a number of bins in $\overline{\mathcal{Q}}_1$.

4 The fill-up phase

4.1 Preliminaries

Once the fill-up phase is reached we refer to the bins by their type they had when the fill-up phase was started and no longer update these sets. E.g., a bin in $\overline{\mathcal{Q}}_2$ at the start of the fill-up phase that receives a big item in the fill-up phase does not become a bin in $\mathcal{Q}_{2,5}$ but is referred to as a bin in $\overline{\mathcal{Q}}_2$ even after receiving the big item. We assume all bins in $\mathcal{S}$ contain more than $4-4\varepsilon$, overestimating its total content by at most $2\cdot(4-4\varepsilon)$. The bin in Δ_{large} is assumed to contain a large$^+$ item once we enter this phase, overestimating its content by at most $3+\varepsilon$. The bin Δ^+_{half} contains an easy item which matches the packing rules in the fill-up phase.

There are three possible states when entering the fill-up phase:

- $s + \overline{q}_1 > 0$
- $s + \overline{q}_1 + \overline{q}_2 = 0$
- $s + \overline{q}_1 = 0$ and $\overline{q}_2 > 0$

The first case is what we will call the standard case where $e \geq \mathcal{E}_0 = \Omega(m)$ holds, for which we can guarantee that when using our packing rules we will eventually end in a good situation or the input ends. For the second and third case we can guarantee that we are already in good situations that do not have requirements on e or r. For more details we refer to the full version.

▶ **Definition 4.1.** *Let e_0 be the number of empty bins at the start of the fill-up phase.*

For the fill-up phase, we introduce the set $\mathcal{U}$ of *unused* bins. These are mostly bins that have not received items in the fill-up phase but that we do plan to use for items. At the start of the fill-up phase, these are the bins that are not in $\mathcal{L} \cup \mathcal{Q}_{\text{match}}$, so $|\mathcal{U}| = m - \ell - q_{\text{match}} = r + e_0$.

Bins in $\mathcal{N}$ are also not used for items anymore, but are initially counted as part of $\mathcal{U}$ so that $\textsc{LargeThreat} \leq u$ (see Invariant 2.9). Bins in $\mathcal{Q}_5$ are initially in $\mathcal{U}$ to maintain the proper ratio $q_1 : q_2$. At the start of the fill-up phase, the bins in $\mathcal{U}$ are sorted from left to right. We use the ordering[3]

$$\mathcal{N}, \mathcal{Q}_{2,5}, \overline{\mathcal{Q}}_2, \mathcal{S}, \mathcal{Q}_1, \mathcal{E}, \overline{\Delta}_{\text{half}} \cup \Delta_{\text{large}}$$

where the subsets $\mathcal{S}$ and $\mathcal{Q}_1$ are ordered by non-increasing levels and Δ^+_{half} is placed among them if it exists. Indeed, the entire set $\mathcal{U}$ is essentially sorted by the levels of small and quarter items at the start of the fill-up phase, so for instance bins in $\mathcal{Q}_1$ (including bins in $\mathcal{Q}_{1,5}$ and $\mathcal{Q}_{1,\text{big}}$) have level at most $4 - 4\varepsilon$ for the sorting. Throughout the fill-up phase, by the level of a bin in $\mathcal{Q}$ we will always mean the total size of the quarter items in this bin at the end of the starting phase. Regarding $\mathcal{N}$, it is often convenient to divide the contents of these bins in a part of size at least $6 - 6\varepsilon$ and a part of size exactly $9 - \varepsilon$ (and this is why these bins are first in the ordering).

During the fill-up phase, we will maintain a set $\mathcal{D}$ such that $\textsc{TopThreat} \leq u - d$ will hold throughout the fill-up phase. We define a specific initial set $\mathcal{D}$ below and we will update this set throughout, using the following rules.

Rule 1 Each bin that is used (in particular bins in $\mathcal{D}$) will receive at least $9 - \varepsilon$ (including parts assigned to a bin but not packed in it) to ensure $\textsc{BigThreat} \leq m - \textsc{BigBlock}$ continues to hold. We already note that for bins that are empty at the start of the fill-up phase the bound of $9 - \varepsilon$ can be reached simply by using Next Fit (it will hold for all but at most one bin at any time).

Rule 2 Whenever some item cannot be packed into some bin in $\mathcal{D}$ that already received items of the same type in the fill-up phase (types are defined below), that bin will leave $\mathcal{D}$ and $\mathcal{U}$. Each time we pack and/or assign $10 + 6\varepsilon$ to $\mathcal{D}$ in the fill-up phase, a new bin is added to $\mathcal{D}$. (Sometimes we will assign parts of items packed into other bins to bins in $\mathcal{D}$.)

Rule 3 Each bin that is not in $\mathcal{D}$ will receive at least $10 + 6\varepsilon$ on average to maintain $\textsc{TopThreat} \leq u - d$.

Reducing the unused bins

We begin the fill-up phase by removing the bins in $\mathcal{N}$ and $\mathcal{Q}_5$ from the unused bins. The contents of these bins were and remain counted. For some later calculations it will still be important that these bins may exist, which is why we include them initially and gave a specific ordering for them.

Recall that the initial value of u is $m - \ell - q_{\text{match}}$. By the transition of the starting phase to the fill-up phase, $\textsc{TopThreat} \leq \frac{9-\varepsilon}{10+6\varepsilon}(m - c - 14)$. So at least

$$\begin{aligned} m - \frac{9-\varepsilon}{10+6\varepsilon}(m-c-14) &= \frac{1+7\varepsilon}{10+6\varepsilon}m + \frac{9-\varepsilon}{10+6\varepsilon} \cdot (14+c) \\ &= \frac{1+7\varepsilon}{10+6\varepsilon}(m-c) + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14 + c \end{aligned}$$

[3] The at most two bins in $\Delta_{\text{nice},1} \cup \Delta_{\text{nice},2}$ can be placed anywhere. However, they are ignored when determining β_0 later.

bins will not receive top items in the fill-up phase (but c of those bins are unavailable for top items anyway). Then after removing $\mathcal{N}$ and $\mathcal{Q}_5$ from the unused bins, we have $u = m - c$, so at least

$$\frac{1+7\varepsilon}{10+6\varepsilon} \cdot u + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14$$

unused bins will not receive top items. We will initially set $d = \frac{1+7\varepsilon}{10+6\varepsilon} \cdot u + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14$.

Analogously, BigThreat $\leq m - c - 14$, so at least $c + 14$ bins will not receive big items, meaning that at least 14 unused bins will not receive big items in the fill-up phase. While packing the input, at any time there will be *half-full* bins. These are bins which have received some items during the fill-up phase but have not yet received (or been counted for) $9 - \varepsilon$ or, in the case of non-$\mathcal{D}$ bins, $10 + 6\varepsilon$. These half-full bins need to be taken into account to ensure that Invariant 2.5 is maintained. Denote their number by h.

▶ **Invariant 4.2.** *At any time, the number of bins in $\mathcal{D}$ that have not yet received any item in the fill-up phase it at least $\frac{1+7\varepsilon}{10+6\varepsilon} \cdot u + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14 - h$, where $u = m - c$ initially and u is updated according to Rule 2.*

▶ **Lemma 4.3.** *As long as we pack items according to Rule 1 and update $\mathcal{D}$ and $\mathcal{U}$ according to Rule 2 for all except at most 10 bins, or pack items according to Rule 3, and at most 13 bins are half-full at any time, TopThreat $\leq u - d$, BigThreat $\leq m -$ BigBlock and Invariant 4.2 are maintained.*

Proof. If we pack items according to Rule 3, the claims follow from the fact that we pack at least $10 + 6\varepsilon$ in every non-$\mathcal{D}$ bin, decreasing TopThreat and BigThreat by at least 1 while increasing BigBlock by at most 1 and removing exactly 1 bin from $\mathcal{U}$ when we start using a new bin. Hence $u - d$ and TopThreat both decrease by 1, and BigThreat decreases by at least 1. In this case the ratio $d : u$ increases.

Regarding items that get packed according to Rule 1, we pack at least $9 - \varepsilon$ in every bin in $\mathcal{D}$ (and then remove such bins from $\mathcal{D}$ and $\mathcal{U}$) and add a new bin to $\mathcal{D}$ after packing $10 + 6\varepsilon$, which means that we add a bin on average after using $\frac{10+6\varepsilon}{9-\varepsilon}$ bins in $\mathcal{D}$. The ratio $d : u$ remains constant during this process apart from at most one bin.

The ratio can be seen as follows. After packing a total size of x into $\mathcal{D}$, we have removed at most $x/(9 - \varepsilon)$ bins from $\mathcal{D}$ (because we only remove a bin from $\mathcal{D}$ once we start using the next one) and we have added $\lfloor x/(10 + 6\varepsilon) \rfloor$ bins to $\mathcal{D}$. Ignoring the rounding, overall d has decreased by at most $x(\frac{1}{9-\varepsilon} - \frac{1}{10+6\varepsilon})$ and u has decreased by at most $x/(9 - \varepsilon)$. The ratio is maintained. The rounding means that the set $\mathcal{D}$ may be 1 smaller during processing. This together with the initial value $d = \frac{1+7\varepsilon}{10+6\varepsilon} \cdot u + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14$ leaves 10 bins for which the rules do not need to be followed, since $\frac{9-\varepsilon}{10+6\varepsilon} \cdot 14 > 11$.

Finally, maintaining BigThreat $\leq m -$ BigBlock given that initially BigThreat $\leq m - c - 14$ means that it is sufficient (due to the bin $\Delta_{\text{nice},2}$) that at most 13 bins will be half-full at any time during the fill-up phase. ◀

A consequence of Invariant 4.2 is that the set $\mathcal{D}$ does not become empty during the packing if we indeed maintain $h \leq 13$. Maintaining this invariant means that all remaining big and top items can be packed at any point during the fill-up phase. Note that if at some point indeed very many top items arrive, they can perhaps not all of them be packed outside of $\mathcal{D}$, as there can be various half-full bins outside of $\mathcal{D}$. However, by Invariant 4.2, as long as the total number of half-full bins is at most 13, all top items can indeed be packed.

Item types

Naturally, as we start filling up bins in the fill-up phase, new thresholds become important. Rather than leaving enough space for items that may arrive in the future as in the starting phase, we now want to use the remaining space efficiently. Bins in $\mathcal{S}$ have at least $(18-2\varepsilon)-(6-6\varepsilon) = 12+4\varepsilon$ space remaining and bins in $\overline{\mathcal{Q}}_2$ leave at least $(18-2\varepsilon)-2(4-4\varepsilon) = 10+6\varepsilon$ space. If some item type fits at least three times on top of bins in $\overline{\mathcal{Q}}_2$, then First Fit gives a stronger bound on the packing and Rule 1 is satisfied. As there is at least $10+6\varepsilon$ remaining space, we define small items to be of size at most $\frac{10+6\varepsilon}{3}$. Items of size more than $\frac{9-\varepsilon}{2}$ that fit twice in this space are also small items. Apart from the range $(5+\varepsilon, 5+3\varepsilon]$, these size ranges are a subset of what defined small items in the starting phase.

The next items are *quarter* items which may not fit three times on $\overline{\mathcal{Q}}_2$ but at least three times on $\mathcal{S}$. For these we need to be slightly more careful; this is described below. *Small* items fit at least four times in an empty bin and at least two times on $\mathcal{S} \cup \overline{\mathcal{Q}}_2$. It can be seen that two items that are larger than small items satisfy Rule 1. To fill the remaining space well, the remaining items are split into quarter$^+$ and quarter^{++} items. Analogously, *easy* items fit twice on $\mathcal{S}$ and satisfy Rule 3, explaining the (unchanged) threshold $6+2\varepsilon$. These items have good sizes as well as large weights. To pack large items in the range $(6+2\varepsilon, 9-\varepsilon]$ efficiently we introduce a new threshold at $\frac{18-2\varepsilon-\beta}{2}$ separating *large*$^-$ and *large*$^+$ items which will be explained later.

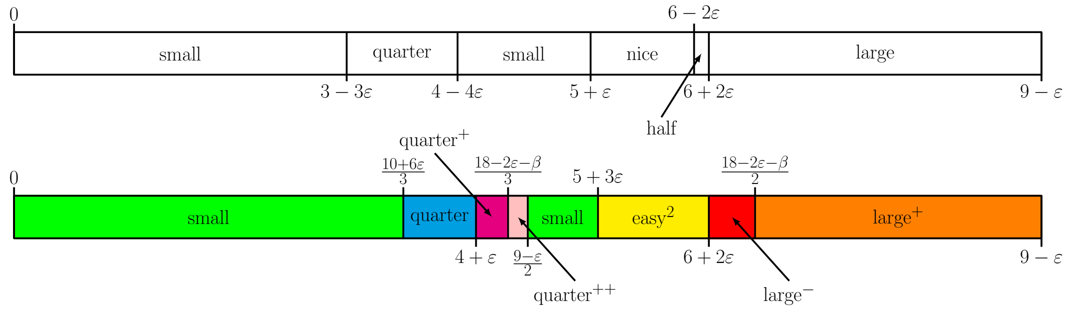

Figure 4 (Sketch, using $\varepsilon = 1/31, \beta = 5$) A comparison of the important thresholds for our algorithm in the starting phase (top) and in the fill-up phase (bottom). The small (green) items fit at least three times (resp. twice) in the remaining empty space in a $\overline{\mathcal{Q}}_2$ bin $(10+6\varepsilon)$, the easy^2 (yellow) items fit at least twice in the remaining empty space in a $\mathcal{S}$ bin $(12+4\varepsilon)$ and the large$^-$ (red) (resp. quarter$^+$ (purple)) items fit at least twice (resp. three times) in the remaining empty space in a bin filled to at most β $(18-2\varepsilon-\beta)$.

The value β will be defined later and will change during the fill-up phase; we will have $\beta \in (3-3\varepsilon, 6-6\varepsilon]$. There are ten size ranges, but the items in six ranges are straightforward to pack (see below). We call the quarter$^+$, quarter^{++}, large$^-$ and large$^+$ items *hard* items.

Type	**Max size**	**Type**	**Max size**
small	$\frac{10+6\varepsilon}{3}$	easy	$6+2\varepsilon$
quarter	$4+\varepsilon$	large$^-$	$\frac{18-2\varepsilon-\beta}{2}$
quarter$^+$	$\frac{18-2\varepsilon-\beta}{3}$	large$^+$	$9-\varepsilon$
quarter^{++}	$\frac{9-\varepsilon}{2}$	big	$10+6\varepsilon$
small	$5+3\varepsilon$	top	12

We see that quarter items and small items may be slightly larger than in the starting phase. This does not decrease their weight. From the starting phase, we will only use the facts that bins in $\overline{\mathcal{Q}}_2$ have level at most $8-8\varepsilon$ and bins in $\mathcal{S}$ have level at most $6-6\varepsilon$; the old size thresholds play no further part in the analysis.

4.2 Packing methods for non-hard items

small and big These are items such that when packing them into $\overline{\mathcal{Q}}_2$ using First Fit (which are the fullest bins (ignoring $\mathcal{X}$) that possibly still need to receive items in the fill-up phase), each bin (apart from constantly many) will receive at least $9 - \varepsilon$ of items.
These items are always packed in the leftmost available bin that did not already receive items from another type in the fill-up phase. That is, we essentially use First Fit, but items of the three different size ranges are not packed together into bins. We only use bins in $\mathcal{D}$. This is feasible because new bins will enter the set $\mathcal{D}$ as we pack items in it (Rule 2).

easy and top When packing easy or top items using First Fit into any bin that is not in $\overline{\mathcal{Q}}_2 \cup \Delta_{\text{large}} \cup \Delta_{\text{nice},1}$, Rule 3 is automatically satisfied. (We have $|\Delta_{\text{large}} \cup \Delta_{\text{nice},1}| \leq 2$.)
These items are packed exactly like the small items except that we skip the bins in $\overline{\mathcal{Q}}_2$. We use First Fit for easy and top items (on the bins not in $\overline{\mathcal{Q}}_2$) separately.

quarter The quarter items are packed by First Fit, alternating between bins in $\overline{\mathcal{Q}}_2$ and bins not in $\overline{\mathcal{Q}}_2$. If we run out of bins in $\overline{\mathcal{Q}}_2$ we use First Fit on dedicated bins not in $\overline{\mathcal{Q}}_2$. If we run out of nonempty bins not in $\overline{\mathcal{Q}}_2$ we can always pack all remaining items as is shown in the full version.

hard These are the quarter$^+$, quarter^{++}, large$^-$ and large$^+$ items. These are the only types of items that we will sometimes (have to) pack into empty bins although nonempty bins that are not in $\overline{\mathcal{Q}}_2$ are still available. When packing hard items of one type using First Fit into empty bins, Rule 3 is satisfied *and* there is a surplus above the requirement of $10 + 6\varepsilon$. We use this surplus to pack as many items as possible into nonempty bins, which is necessary to succeed. See Section 4.3.

Once First Fit uses a new bin for an item, the bin previously considered (in which the item did not fit anymore) is removed from $\mathcal{U}$.

4.3 Hard (quarter$^{+(+)}$ and large) items

The above lemmas show that in the fill-up phase, all items except quarter$^{+(+)}$ and large items can be packed while following the rules – as long of course as quarter$^{+(+)}$ and large items are packed following the rules as well. We next consider quarter$^{+(+)}$ and large items separately. For this we first need to define some important sets.

4.3.1 Sets considered for packing hard items

The set $\mathcal{S}_{-L}$ and the parameter β_0

At the beginning of the fill-up phase, if there are e empty bins, then if we pack two large items in each empty bin, there will be e nonempty bins that will *not* be needed to pack large items that have not arrived yet, even if u of them arrive in total. In principle, we let $\mathcal{S}_{-L}$ be the e *rightmost* nonempty bins. We only deviate from this if there are less than e nonempty bins. In that case $\mathcal{S}_{-L}$ consists of all nonempty unused bins (after the removal of $\mathcal{N} \cup \mathcal{Q}_{2,5}$). We define β_0 as the level of the leftmost bin in $\mathcal{S}_{-L}$ unless all other nonempty bins are in $\mathcal{N} \cup \mathcal{Q}_{2,5}$, in which case β_0 is set to $6 - 6\varepsilon$. Of course, due to other items arriving, it may well be that some large items end up getting packed in $\mathcal{S}_{-L}$ after all.

Since $u = m - \ell - q_{\text{match}}$ at the start of the fill-up phase, it is not possible that more than $u - q_5$ large items arrive in the fill-up phase. This holds because each bin counted in $q_5 + q_{\text{match}}$ contains two items larger than $6 - 2\varepsilon$ or a dominant item, and each bin counted in ℓ contains two items larger than $6 - 2\varepsilon$ or an item larger than 6. Moreover, as above, Invariant 4.2 is maintained by the $(9 - \varepsilon)$-guarantee. As soon as u starts to decrease in the fill-up phase, more than $u - q_5$ large items may still arrive.

Construction of the set $\mathcal{D}$

We initially define $\mathcal{D}$ as the leftmost $\frac{1+7\varepsilon}{10+6\varepsilon} \cdot u + \frac{9-\varepsilon}{10+6\varepsilon} \cdot 14$ unused bins (where bins in $\mathcal{Q}_5$ and $\mathcal{N}$ have already been eliminated).

Definition of the set $\mathcal{S}_L$

We define $\mathcal{S}_L$ as the set of the remaining nonempty bins in $\mathcal{U}$ (that is, all the unused bins that are not in $\mathcal{D} \cup \mathcal{S}_{-L} \cup \mathcal{E}$). The set $\mathcal{S}_L$ may be empty.

Note that the above construction is done only once. The sets $\mathcal{S}_L$ and $\mathcal{S}_{-L}$ do not increase and a bin leaves such a set only if the bin leaves $\mathcal{U}$ or is added to $\mathcal{D}$. While packing items in the fill-up phase, bins will be added to $\mathcal{D}$ from left to right. Hence entering $\mathcal{D}$ will happen first to bins in $\mathcal{S}_L$ and then (possibly) to $\mathcal{S}_{-L}$ and $\mathcal{E}$. Such bins leave their original sets.

Bins in $\mathcal{S}$ are filled to at most $6 - 6\varepsilon$ and hence have at least $12 + 4\varepsilon$ empty space. Easy^2 items fit at least twice. Bins in $\mathcal{S}_{-L}$ are filled to at most β_0 and hence have at least $18 - 2\varepsilon - \beta_0$ empty space. Large$^-$ items therefore fit at least twice (explaining that threshold). Bins initially in $\mathcal{D}$ are loaded to at most $2(4 - 4\varepsilon)$ and hence have at least $18 - 2\varepsilon - 2(4 - 4\varepsilon) = 10 + 6\varepsilon$ space.

4.3.2 Packing methods for hard items: Five stages

Hard items are packed as follows. There are five possible stages; depending on what the packing looks like when the fill-up phase starts, not all stages might be applicable. Generally, we start by taking advantage of any $\mathcal{Q}_{1,5}$ bins that are available, then we start filling the nonempty bins from right to left, always trying to avoid using empty bins as much as possible.

Hard items are always distributed among several types of bins $(\mathcal{D}, \mathcal{S}_L, \mathcal{S}_{-L}, \mathcal{E})$. One of the used types will always be $\mathcal{D}$. In several cases, we will specify that nonintegral numbers of bins need to be used for some items. This can be implemented as follows. We always start by using a bin that is not in $\mathcal{D}$. Then, we keep using bins in $\mathcal{D}$ until we would exceed the desired ratio. At that point we use a bin that is not in $\mathcal{D}$ again, and repeat the process. In the long term we get closer and closer to the desired ratio.

Stage 1

In this stage, as mentioned above we first exploit any bins in $\mathcal{Q}_{1,5}$ that may exist. These bins can be intermixed with $\overline{\mathcal{Q}}_1$ in the sorted order, but we use them first. To be more precise, these bins are not used to pack any new items (as they are already quite full) but rather to pack additional items into $\mathcal{D}$, as follows. Note that these bins are not in $\mathcal{U}$. For this stage we define the upper bound for quarter$^+$ items as $\frac{9-\varepsilon}{2}$ and the upper bound for large$^-$ items as $9 - \varepsilon$ and hence neither quarter^{++} items nor large$^+$ items exist. This also means that the value of β (see table of item type thresholds for the fill-up phase) does not yet play a role.

Quarter$^{+(+)}$: As long as there exist bins in $\mathcal{Q}_{1,5}$ with partially uncounted contents, for packing quarter$^{+(+)}$ items such bins are considered to contain quarter$^{+(+)}$ items (of the size of the ignored items). Each such bin allows us to pack $\frac{1+7\varepsilon}{1-3\varepsilon}$ bins in $\mathcal{D}$ with two quarter$^{+(+)}$ items each. In these $\frac{1+7\varepsilon}{1-3\varepsilon}$ bins we then pack (or count) more than $2\frac{1+7\varepsilon}{1-3\varepsilon}(4 + \varepsilon) + (1 + 7\varepsilon) = \frac{1+7\varepsilon}{1-3\varepsilon} \cdot (9 - \varepsilon)$, which includes the so far uncounted part of the bin in $\mathcal{Q}_{1,5}$.

Large: As long as there exist bins in $\mathcal{Q}_{1,5}$ with partially uncounted contents, for packing large items such bins are considered to contain large items (of the size of the ignored items). Each such bin allows us to pack $\frac{1+7\varepsilon}{3-3\varepsilon}$ bins in $\mathcal{D}$ with one large item each. In these $\frac{1+7\varepsilon}{3-3\varepsilon}$ bins we then pack (or count) more than $\frac{1+7\varepsilon}{3-3\varepsilon}(6+2\varepsilon)+(1+7\varepsilon)=\frac{1+7\varepsilon}{3-3\varepsilon}\cdot(9-\varepsilon)$, which includes the so far uncounted part of the bin in $\mathcal{Q}_{1,5}$.

It can be seen that by packing quarter$^{+(+)}$ and large items this way, Rule 1 and Rule 3 are followed. Once we run out of bins in $\mathcal{Q}_{1,5}$, we start using the methods in the following table. The value of β will change over time and is set at the start of each of the following stages. Note here that for $\beta\le\frac{9-\varepsilon}{2}$ the upper bound for quarter$^+$ items is at least the upper bound for quarter^{++} items and hence quarter^{++} items do not exist as long as $\beta\le\frac{9-\varepsilon}{2}$.

Item type	Bin type	Nr bins	Per bin	Counted	Average
quarter$^+$	$\mathcal{D}$	3	2	$2(4+\varepsilon)$	$9+\frac{9}{4}\varepsilon$
	$\overline{\mathcal{Q}}_1,\mathcal{S}_{-L}(,\mathcal{S}_L),\mathcal{E}$	1	3	$3(4+\varepsilon)$	
quarter^{++}	$\mathcal{E}$	1	4	$\frac{4}{3}(18-2\varepsilon-\beta)$	$9-\varepsilon$
	$\mathcal{D}$	$\frac{45-4\beta-5\varepsilon}{2\beta-(9-\varepsilon)}$	2	$\frac{2}{3}(18-2\varepsilon-\beta)$	
large$^-$	$\mathcal{D}$	1	1	$6+2\varepsilon$	$9+3\varepsilon$
	$\overline{\mathcal{Q}}_1,\mathcal{S}_{-L}(,\mathcal{S}_L),\mathcal{E}$	1	2	$12+4\varepsilon$	
large$^+$	$\mathcal{D}$	$\frac{18-2\varepsilon}{\beta}-2$	1	$\frac{18-2\varepsilon-\beta}{2}$	$9-\varepsilon$
	$\mathcal{E}$	1	2	$18-2\varepsilon-\beta$	

The third column indicates the number of bins used each time. The column "Average" contains the average amount counted over all bin types used for this item.

This table is implemented as follows. Items from these four types are packed into separate bins. For each type except quarter^{++} items, we first use a bin in $\mathcal{D}$. For quarter^{++} items, we use $\mathcal{E}$ first to ensure that the average packed in the bins with these items is always greater than $9-\varepsilon$, apart from at most one bin: the bin currently being used. For all other types, we have this guarantee for all used bins up to and including the most recently filled bin in $\mathcal{E}$, which is all but at most four bins for each type.

For each bin used we pack items in it until we have packed the number of items in the column Per bin. We first use bins in $\mathcal{D}$ until we have packed the number of items in the column Per bin and until we have used the number of bins in the column Nr bins, followed by one bin in $\mathcal{E}$. (If the number of bins supposed to be used in $\mathcal{D}$ is not an integer, then the number of bins used in $\mathcal{D}$ is always off by less than 1 compared to the desired ratio of $\mathcal{D}:\mathcal{E}$ usage.) This procedure keeps repeating. Whenever we start using a new bin, the previous bin for this type is called *closed*. Once a bin is closed it is removed from $\mathcal{U}$.

In the following we will always check if bins with level at most β exist. While this is true we remain in the current stage. Else we update β to the next higher bound.

Stage 2

We set $\beta:=\min(\beta_0,4-4\varepsilon)$. As long as there are bins with level at most β available, we use these bins (including bins in $\mathcal{S}_L$ if needed) for packing quarter$^+$ and large$^-$ items. There are no quarter^{++} items and only large$^+$ items are packed into empty bins.

Stage 3

As Stage 2, but with $\beta:=\min(\beta_0,\frac{9-\varepsilon}{2})$. Bins in $\mathcal{S}_L$ with level at most β will still be used for quarter$^+$ and large$^-$ items. There are no quarter^{++} items and only large$^+$ items are packed into empty bins.

Stage 4

This stage only exists if $\beta_0 > \frac{9-\varepsilon}{2}$. We set $\beta := \beta_0$.

Quarter^{++} and large^{+} items are packed into empty bins. All items are packed according to the table. Bins in $\mathcal{S}_L$ are *not* used for quarter^{+} items; we would go to Stage 5 instead. This is done to ensure that enough bins are left for large^{-} items and that we do not waste bins on quarter^{+} items.

Stage 5

Only bins in $\mathcal{S}_L \cup \mathcal{D} \cup \mathcal{E}$ are left. At this point quarter^{+} and large^{-} items are also packed into empty bins.

Stage 6 would be the stage where all empty bins have been filled while some bins in $\mathcal{S}_L \setminus \mathcal{D}$ remain. The case where all nonempty bins get filled first is discussed in the full version. We will show that Stage 6 is not reached or we are in a good situation.

As is in the starting phase, if some item cannot be packed according to the packing rules (including the case where we change the packing rules if we reach a good situation) we use the rule of last resort.

References

1 Islam Akaria and Leah Epstein. Bin stretching with migration on two hierarchical machines. *Math. Methods Oper. Res.*, 98(1):111–153, 2023.

2 Susanne Albers and Matthias Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012.

3 Yossi Azar and Oded Regev. On-line bin-stretching. In *RANDOM*, volume 1518 of *Lecture Notes in Computer Science*, pages 71–81. Springer, 1998.

4 Yossi Azar and Oded Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1):17–41, 2001.

5 János Balogh, József Békési, György Dósa, Jiří Sgall, and Rob van Stee. The optimal absolute ratio for online bin packing. *J. Comput. Syst. Sci.*, 102:1–17, 2019. `doi:10.1016/j.jcss.2018.11.005`.

6 Martin Böhm, Matej Lieskovský, Sören Schmitt, Jiří Sgall, and Rob van Stee. Improved online load balancing with known makespan. *arXiv e-prints*, page arXiv:2407.08376, 2024. `doi:10.48550/arXiv.2407.08376`.

7 Martin Böhm, Jiří Sgall, Rob van Stee, and Pavel Veselý. Online bin stretching with three bins. *Journal of Scheduling*, 20(6):601–621, 2017.

8 Martin Böhm, Jiří Sgall, Rob Van Stee, and Pavel Veselỳ. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34(3):810–828, 2017.

9 Martin Böhm and Bertrand Simon. Discovering and certifying lower bounds for the online bin stretching problem. *Theor. Comput. Sci.*, 938:1–15, 2022.

10 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, August 2016. `doi:10.1145/2993749.2993766`.

11 Jérôme Dohrau. Online makespan scheduling with sublinear advice. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 177–188. Springer, 2015.

12 Leah Epstein. Bin stretching revisited. *Acta Informatica*, 39(2):97–117, 2003.

13 Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

14 Michaël Gabay, Nadia Brauner, and Vladimir Kotov. Improved lower bounds for the online bin stretching problem. *4OR*, 15(2):183–199, 2017.

15 Michaël Gabay, Vladimir Kotov, and Nadia Brauner. Online bin stretching with bunch techniques. *Theoretical Computer Science*, 602:103–113, 2015.

16 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

17 Hans Kellerer and Vladimir Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013.

18 Hans Kellerer, Vladimir Kotov, and Michaël Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *J. Sched.*, 18(6):623–630, 2015. doi:10.1007/s10951-015-0430-4.

19 Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

20 Antoine Lhomme, Nicolas Catusse, and Nadia Brauner. Computational bounds on randomized algorithms for online bin stretching. *CoRR*, abs/2405.19071, 2024. doi:10.48550/arXiv.2405.19071.

21 Antoine Lhomme, Olivier Romane, Nicolas Catusse, and Nadia Brauner. Online bin stretching lower bounds: Improved search of computational proofs. *CoRR*, abs/2207.04931, 2022.

22 Matej Lieskovský. Better algorithms for online bin stretching via computer search. *CoRR*, abs/2201.12393, 2022. arXiv:2201.12393.

23 Matej Lieskovský. Better algorithms for online bin stretching via computer search. In *LATIN (1)*, volume 14578 of *Lecture Notes in Computer Science*, pages 241–253. Springer, 2024.

24 Marc P Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theoretical Computer Science*, 600:155–170, 2015.

25 John F Rudin III. *Improved bounds for the online scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.