Online Bin Covering with Limited Migration

Sebastian Berndt
Department of Computer Science, Kiel University, Kiel, Germany
seb@informatik.uni-kiel.de

Leah Epstein
Department of Mathematics, University of Haifa, Haifa, Israel
lea@math.haifa.ac.il

Klaus Jansen
Department of Computer Science, Kiel University, Kiel, Germany
kj@informatik.uni-kiel.de

Asaf Levin
Faculty of Industrial Engineering and Management, The Technion, Haifa, Israel
levinas@technion.ac.il

Marten Maack
Department of Computer Science, Kiel University, Kiel, Germany
mmaa@informatik.uni-kiel.de

Lars Rohwedder
Department of Computer Science, Kiel University, Kiel, Germany
lro@informatik.uni-kiel.de

Abstract

Semi-online models where decisions may be revoked in a limited way have been studied extensively in the last years.

This is motivated by the fact that the pure online model is often too restrictive to model real-world applications, where some changes might be allowed. A well-studied measure of the amount of decisions that can be revoked is the migration factor β: When an object o of size $s(o)$ arrives, the decisions for objects of total size at most $\beta \cdot s(o)$ may be revoked. Usually β should be a constant. This means that a small object only leads to small changes. This measure has been successfully investigated for different, classical problems such as bin packing or makespan minimization. The dual of makespan minimization – the Santa Claus or machine covering problem – has also been studied, whereas the dual of bin packing – the bin covering problem – has not been looked at from such a perspective.

In this work, we extensively study the bin covering problem with migration in different scenarios. We develop algorithms both for the static case – where only insertions are allowed – and for the dynamic case, where items may also depart. We also develop lower bounds for these scenarios both for amortized migration and for worst-case migration showing that our algorithms have nearly optimal migration factor and asymptotic competitive ratio (up to an arbitrary small ε). We therefore resolve the competitiveness of the bin covering problem with migration.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, dynamic algorithms, competitive ratio, bin covering, migration factor

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.18

Related Version A full version of the paper is available at `https://arxiv.org/abs/1904.06543`.

Acknowledgements This work was partially supported by the DFG Project, "Robuste Online-Algorithmen für Scheduling- und Packungsprobleme", JA 612 /19-1, and by GIF-Project "Polynomial Migration for Online Scheduling".

© Sebastian Berndt, Leah Epstein, Klaus Jansen, Asaf Levin, Marten Maack, and Lars Rohwedder;
licensed under Creative Commons License CC-BY

1 Introduction

Online algorithms aim to maintain a competitive solution without knowing future parts of the input. The competitive ratio of such an algorithm (for a maximization problem) is thus defined as the worst-case ratio between the value of an optimal solution produced by an offline algorithm knowing the complete input and the value of the solution produced by the online algorithm. Furthermore, once a decision is made by these algorithms, this decision is fixed and irreversible. While a surprisingly large number of problems do have such algorithms, the complete irreversibility requirement is often too strict, leading to high competitive ratios. Furthermore, if the departure of objects from the instance is also allowed, irreversible online algorithms are rarely able to be competitive at all. From a practical point of view, this is quite alarming, as the departure of objects is part of many applications. We call such a problem *dynamic* and the version with only insertions *static*.

A number of different scenarios to loosen the strict requirement of irreversibility – called *semi-online* scenarios – have been developed over time in order to find algorithms that achieve good competitive ratios for some of the scenarios with bounded reversibility. In the last few years, the concept of the *migration factor* has been studied intensively [3, 10, 11, 12, 13, 16, 19, 21, 22]. Roughly speaking, a migration factor of β allows to reverse a total size of $\beta \cdot s(o)$ decisions, where $s(o)$ denotes the *size* of the newly arrived object o. For a packing problem, this means that the algorithm is allowed to repack objects with a total size of $\beta \cdot s(o)$. This notion of reversibility is very natural, as it guarantees that a small object can only lead to small changes in the solution structure. Furthermore, algorithms with bounded migration factor often show a very clear-cut tradeoff between their migration and the competitive ratio: Many algorithms in this setting have a bounded migration factor which can be defined as a function $f(\varepsilon)$ (growing with $\frac{1}{\varepsilon}$) and a small competitive ratio $g(\varepsilon)$ (growing with ε), where the functions f and g can be defined for all $\varepsilon > 0$ [3, 10, 13, 16, 19, 21, 22]. Such algorithms are called *robust*, as the amount of reversibility allowed only depends on the solution guarantee that one wants to achieve. Such robust algorithms thus serve as evidence for the possibility for sensitivity analysis in approximated settings.

Many different problems have been studied in online and semi-online scenarios, but two problems that have been considered in nearly every scenario are classical scheduling problems: The *bin packing* problem and the *makespan minimization* problem. Both of these problems have been studied intensively in the migration model [3, 10, 12, 13, 19, 21, 22]. Both of these problems also have corresponding dual maximization variants. The dual version of the makespan minimization problem, often called the *Santa Claus* or *machine covering* problem, has also been studied with migration [16, 22]. In contrast, the dual version of bin packing, called *bin covering* has not yet been studied in this model. The aim of this paper is to remedy this situation by taking a look at this classical scheduling problem in the migration model.

Formal Problem Statement. In the *bin covering problem*, a set of items Γ with sizes $s\colon \Gamma \to (0, 1]$ is used to cover as many unit sized bins as possible, that is, Γ has to be partitioned maximizing the number of partitions with summed up item size of at least one. An instance of the problem will usually be denoted as I and is given as a sequence of entries $(i, s(i))$ where i is the identifier of the item and $s(i)$ is the size of the item. A *solution* to such an instance I with items Γ is a partition $P\colon \Gamma \to \mathbb{N}$ and a set $B = P^{-1}(k)$ with $B \neq \emptyset$ is called a *bin* and we say that the items in B are packed into the k-th bin. A bin B is *covered* if $s(B) := \sum_{i \in B} s(i) \geq 1$, where $s(B)$ is called the load of B, and the goal is to maximize the number of such covered bins. The optimal (maximum) number of covered bins of instance I is denoted as $\textsc{opt}(I)$.

We also use the following notation throughout our work: The smallest size of an item in bin B is defined as $s_{\min}(B) := \min_{i \in B}\{s(i)\}$ and the largest size is defined as $s_{\max}(B) := \max_{i \in B}\{s(i)\}$. If $\mathcal{B}$ is a set of bins, we also define its total size $s(\mathcal{B}) := \sum_{B \in \mathcal{B}} s(B)$, its minimal size $s_{\min}(\mathcal{B}) := \min_{B \in \mathcal{B}} s_{\min}(B)$, and its maximal size $s_{\max}(\mathcal{B}) := \max_{B \in \mathcal{B}} s_{\max}(B)$. Furthermore, we define $s_{\min}(\emptyset) = +\infty$ and $s_{\max}(\emptyset) = 0$.

We consider variants of static and dynamic online bin covering in which algorithms are allowed to reassign a bounded amount of previously assigned items. In particular, an algorithm has a *migration factor* of β, if the total size of items that it reassigns upon arrival or departure of an item of size s is bounded by βs. Moreover, it has an *amortized migration factor* of β, if at any time the total size of items that have been reassigned by the algorithm in total is bounded by βS, where S is the total size of all items that arrived before. Intuitively, an item of size s creates a migration potential of size βs upon arrival, and this potential may be used by an algorithm to reassign items right away (non-amortized) or anytime from then on (amortized). Note that if an algorithm has a non-amortized migration factor of β, it also has an amortized migration factor of at most β. Thus, we study four variants in this work.

Offline bin covering is NP-hard and therefore there is little hope for a polynomial time algorithm solving the problem to optimality, and in the online setting there is no algorithm that can maintain an optimal solution regardless of its running time. We prove that this non-existence of algorithms that maintain an optimal solution holds also for (static or dynamic) algorithms with bounded amortized migration factor (and thus also for algorithms with bounded non-amortized migration factor).

Hence, algorithms satisfying some performance guarantee are studied. In particular an offline algorithm ALG for a maximization problem has an *asymptotic performance guarantee* of $\alpha \geq 1$, if $\text{OPT}(I) \leq \alpha \cdot \text{ALG}(I) + c$, where $\text{OPT}(I)$ and $\text{ALG}(I)$ are the objective values of an optimal solution or the one produced by ALG respectively for some instance I, and c is an input independent constant. If $c = 0$ holds, α is called *absolute* rather than asymptotic. An online algorithm has a (asymptotic or absolute) *competitive ratio* of α, if after each arrival or departure an (asymptotic or absolute) performance guarantee of α for the instance of the present items holds. Note that we use the convention of competitive ratios larger than 1 for maximization problems. For minimization problems similar definitions are used but they use the required inequality $\text{ALG}(I) \leq \alpha \cdot \text{OPT}(I) + c$. As we study asymptotic competitive ratios in this paper, we will sometimes omit the word asymptotic (and we always use the word absolute for absolute competitive ratios).

Our Results. We present competitive algorithms using both amortized migration and non-amortized migration and develop nearly matching lower bounds (up to an arbitrarily small additive term of ε). These bounds show the optimality of all of our algorithms for both the static and the dynamic version of the bin covering problem. The main technical contribution of our work is an algorithm with competitive ratio $3/2 + \varepsilon$ and non-amortized migration of $\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$ for the dynamic bin covering problem where items arrive and depart. A major obstacle in the design of competitive algorithms for dynamic problems is the impossibility of moving large items on the arrival or departure of small items. We overcome this obstacle by developing a delicate technique to combine the packing of large and small items. The main results of this work are summarized in the following table. Note that the lower bound of 1 in the third row indicates that there is no online algorithm that maintains an optimal solution with amortized migration factor $\mathcal{O}(1)$. All of our algorithms run in polynomial time. Curiously, we achieve a polynomial migration factor, while most known migration factors are exponential (e. g. for the makespan minimization problem [21]) with the exception of bin packing [3, 18].

Amortization	Departures	Lower Bound	Competitive Ratio	Migration
✖	✖	3/2	$3/2+\varepsilon$	$\mathcal{O}(1/\varepsilon)$
✖	✔	3/2	$3/2+\varepsilon$	$\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$
✔	✖	1	$1+\varepsilon$	$\mathcal{O}(1/\varepsilon)$
✔	✔	3/2	$3/2+\varepsilon$	$\mathcal{O}(1/\varepsilon^5 \cdot \log^2(1/\varepsilon))$

Due to space constraints, we focus on the non-amortized case with only insertions (described in the first row) and prove most of the corresponding results here. The remaining results are shortly described and a full presentation of them is given in the appendix.

Related Results

Bin Covering. The offline bin covering problem was first studied by Assmann et al. [1]. It was shown that a simple greedy strategy achieves approximation ratio 2. For the online version of the bin covering problem, Csirik and Totik showed in [6] that this simple greedy algorithm also works in the online setting and that the competitive ratio of 2 reached by this algorithm is the best possible. Csirik, Johnson, and Kenyon presented an asymptotic polynomial time approximation scheme (APTAS) with approximation ratio $1+\varepsilon$ in [5]. This was improved to an asymptotic fully polynomial time approximation scheme (AFPTAS) by Jansen and Solis-Oba in [20]. Many different variants of this problem have also been investigated: If a certain number of classes needs to be part of each bin [9, 15]; if items are drawn probabilistically [14, 15]; if bins have different sizes [7, 23]; if the competitiveness is not measured with regard to an optimal offline algorithm [4, 8]. More variants are e. g. discussed in [17] and lower bounds for several variants are studied in [2].

Makespan Minimization and Santa Claus. The migration factor model was introduced by Sanders, Sivadasan, and Skutella in [21]. The paper investigated several algorithms for the makespan minimization problem and also presents an approximation scheme with absolute competitive ratio $1+\varepsilon$ and non-amortized migration factor $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Skutella and Verschae [22] studied a dynamic setting with amortized migration, where jobs may also depart from the instance. They achieved the same absolute competitive ratio, but their algorithm needs an amortized migration of $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$. Their algorithm also works for the Santa Claus (or machine covering) problem, for which they show that even in the static setting no algorithm has absolute competitive ratio $1+\varepsilon$ and a bounded migration factor. If one aims for a polynomial migration factor for the Santa Claus problem, Gálvez, Soto, and Verschae presented an online variant of the LPT (longest processing time) algorithm achieving an absolute competitive ratio of $4/3+\varepsilon$ with non-amortized migration factor $O(1/\varepsilon^3 \log(1/\varepsilon))$ [16]. For the makespan minimization problem with preemption, Epstein and Levin showed in [12] that an optimal algorithm with a non-amortized migration factor of $1-1/m$ is achievable and best possible.

Bin Packing. Epstein and Levin presented an approximation scheme with the same ratio $1+\varepsilon$ and the same non-amortized migration factor $2^{O(1/\varepsilon \log^2(1/\varepsilon))}$ as in the makespan minimization for the bin packing problem in [10]. This result was improved by Jansen and Klein in [18], who drastically reduced the migration factor to $O(1/\varepsilon^4)$. Berndt, Jansen, and Klein used a similar approach to also handle the dynamic bin packing problem, where items may also depart over time [3]. They also showed that a non-amortized migration factor of $\Omega(1/\varepsilon)$ is needed for this. A generalized model, where an item i has arbitrary

movement costs c_i – not necessarily linked to the size of an item – was studied by Feldkord et al. [13]. They showed that for $\alpha \approx 1.387$ and every $\varepsilon > 0$, a competitive ratio of $\alpha + \varepsilon$ is achievable with migration $O(1/\varepsilon^2)$, but no algorithm with migration $o(n)$ and ratio $\alpha - \varepsilon$ exists. Strengthening the lower bound of [3], they also showed an amortized migration factor of $\Omega(1/\varepsilon)$ is needed for the standard model, where movement costs c_i equal items sizes s_i, if one wants to achieve competitive ratio $1 + \varepsilon$. A generalization of bin packing – packing d-dimensional cubic items into unit size cubes – was studied by Epstein and Levin [11].

2 Non-amortized Migration in the Static Case

We begin our study by analyzing the static case with non-amortized migration. We will first present a lower bound showing that no algorithm with constant non-amortized migration factor can have a competitive ratio below 3/2. Then, we present an algorithm that achieves for all $\varepsilon > 0$ a competitive ratio of $3/2 + \varepsilon$ with non-amortized migration factor $O(1/\varepsilon)$.

We start with a simple lower bound on the asymptotic competitive ratio of all algorithms with constant non-amortized migration factor. This lower bound can also be proved for a different definition of the asymptotic competitive ratio α, where we require $\textsc{opt}(I) \leq \alpha \cdot \textsc{alg}(I) + o(\textsc{opt}(I))$.

▶ **Proposition 1.** *There is no algorithm for static online bin covering with a constant non-amortized migration factor and an asymptotic competitive ratio smaller than* 3/2*.*

Proof sketch. Fix a migration factor β and an integer N. First, insert $6N$ items of size $1 - \varepsilon$ and then $6N$ items of size ε, where $\varepsilon = \min\{(2\beta + 2)^{-1}\}$. ◀

We will now give our algorithm ALG for this scenario. In addition to the instance I, a parameter $\varepsilon > 0$ is also given that regulates the asymptotic competitive ratio and the used migration. The assumption $\varepsilon \leq 0.5$ is justified as for $\varepsilon \geq 0.5$, the result follows by the online algorithm with an asymptotic competitive ratio of 2 presented in [6], or by using the algorithm below with $\varepsilon = \frac{1}{2}$.

▶ **Theorem 2.** *For each $\varepsilon \in (0, 0.5]$, there is an algorithm* ALG *for static online bin covering with polynomial running time, an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant* 3*, and a non-amortized migration factor of $\mathcal{O}(1/\varepsilon)$.*

The algorithm distinguishes between big, medium and small items. For each item, it calls a corresponding insertion procedure based on this classification into three classes. An item i is called *big* if $s(i) \in (0.5, 1]$, *medium* if $s(i) \in (\varepsilon, 0.5]$, and *small* otherwise. For a bin B, let $\text{small}(B)$ be the set of small items of B. We define $\text{medium}(B)$ and $\text{big}(B)$ accordingly and also extend these notions to sets of bins $\mathcal{B}$. Furthermore, we call a covered bin *barely covered*, if removing the biggest item of the smallest class of items, i. e. big, medium or small, contained in the bin, results in the bin not being covered anymore. For instance, consider a bin containing four items with sizes 0.65, 0.3, ε and 0.25ε, for $\varepsilon \geq 0.04$. This bin contains items of all three classes if $\varepsilon < 0.3$. In this case, the biggest item of the smallest class has size ε, and if $\varepsilon = 0.1$, the bin is indeed barely covered. However, if $\varepsilon = 0.22$, the bin is not barely covered. If $\varepsilon = 0.3$, the bin only has items of two classes, and it is not barely covered, since removing one item of size 0.3 results in a total size above 1. Note that showing that removing an arbitrary item of the smallest class of items for a given covered bin results in an uncovered bin is sufficient for showing that it is barely covered.

Let B be a barely covered bin. If B contains at most one big item, its load is bounded from above by 1.5, and if B additionally contains no medium item the bound is reduced to $1 + \varepsilon$. This holds due to the following. Since the big item has size below 1, the bin contains at

least one medium or small item. If the bin has no small items, removing the largest medium item reduces the load to below 1, and together with the medium item the load is below 1.5. If it has a small item, a similar calculation shows that the load is below $1 + \varepsilon$.

The last two types of bins are benign in the sense that they allow analysis using arguments that are based on sizes. This is not the case for bins containing two big items. Such bins could have a size arbitrarily close to 2 (even if they have no other items). Bins of this type are needed for instances with many big items (for an example we refer to the construction of the lower bound in Proposition 1). However, they cause problems not only because they are wastefully packed, but also because they exclusively contain big items that should only be moved if suitably large items arrive in order to bound migration. The basic idea of the algorithm is to balance the number of bins containing two big items and the number of bins containing one big item and no medium items. The two numbers will be roughly the same, which is obtained using migration on arrivals of big and medium items. As described in the previous paragraph, the guarantees of these bins that are based on loads cancel each other out in the sense that an average load not exceeding $1.5 + \varepsilon/2$ can be achieved. In order to keep the number of bins with two big items in check, our algorithm will only produce very few bin types and we will maintain several invariants. This structured approach allows us also to bound the migration needed. We elaborate on the details of the algorithm.

Bin Types and Invariants

We distinguish different types of bins packed by the algorithm.

The bins are partitioned into bins containing two big items (and no other items) $\mathtt{BB}$; barely covered bins containing one big item and some medium items $\mathtt{BM}$; *barely covered* bins containing one big item and some small items $\mathtt{BSC}$; bins that are not covered (*partially covered*) and contain one big item and no medium items (but it could contain small items) $\mathtt{BSP}$; and bins that are at most barely covered (they are barely-covered, or not covered) and exclusively contain small or medium items $\mathtt{S}$ or $\mathtt{M}$ respectively. Furthermore, let $\mathtt{MC} \subseteq \mathtt{M}$ and $\mathtt{SC} \subseteq \mathtt{S}$ be the corresponding subsets of barely covered bins (while $M \setminus MC$ and $S \setminus SC$ are sets of bins that are not covered). We denote the (disjoint) union of $\mathtt{BSC}$ and $\mathtt{BSP}$ as $\mathtt{BS}$. The set of bins packed by the algorithm (covered or not covered) is denoted as $\mathtt{Bins}$. All bins covered by the algorithm are in fact barely covered, and no bin (covered or not) contains items of all three classes. Among bins that are not covered, there are no bins containing a big item and a non-empty set of medium items.

We will now introduce the invariants needed. The first invariant of the algorithm ensures that this bin structure is maintained and the second invariant was already indicated above ($A \dot\cup B$ denotes the disjoint union of A and B):

I1 The solution has the proposed bin type structure, i. e. $\mathtt{Bins} = \mathtt{BB} \dot\cup \mathtt{BM} \dot\cup \mathtt{BS} \dot\cup \mathtt{M} \dot\cup \mathtt{S}$ and $\mathtt{BS} = \mathtt{BSC} \dot\cup \mathtt{BSP}$.

I2 The sets $\mathtt{BB}$ and $\mathtt{BS}$ are balanced in size, i. e. $\big||\mathtt{BB}| - |\mathtt{BS}|\big| \leq 1$.

Therefore we have $\textsc{alg}(I) = |\mathtt{BB}| + |\mathtt{BM}| + |\mathtt{BSC}| + |\mathtt{MC}| + |\mathtt{SC}|$. Furthermore, we have several invariants concerning the distribution of items to different bin types. The intuition behind these invariants is always the same: We have to ensure that no other algorithm is able to use the small and medium items to cover too many bins.

I3 The big items contained in $\mathtt{BM}$ are at least as big as the ones in $\mathtt{BSC}$ which in turn are at least as big as the ones in $\mathtt{BSP}$, and the smallest big items are placed in $\mathtt{BB}$, i. e. $s(i) \geq s(i')$ for each (i) $i \in \text{big}(\mathtt{BM})$ and $i' \in \text{big}(\mathtt{BS} \cup \mathtt{BB})$; each (ii) $i \in \text{big}(\mathtt{BM} \cup \mathtt{BSC})$ and $i' \in \text{big}(\mathtt{BSP} \cup \mathtt{BB})$; each (iii) $i \in \text{big}(\mathtt{BM} \cup \mathtt{BS})$ and $i' \in \text{big}(\mathtt{BB})$. Informally, $\mathtt{BB} \leq \mathtt{BSP} \leq \mathtt{BSC} \leq \mathtt{BM}$.

I4 The items in M cannot be used to cover a bin together with a big item from BS or BB, i. e. $\mathtt{BS} \cup \mathtt{BB} \neq \emptyset \implies s(\mathtt{M}) < 1 - s_{\max}(\mathtt{BS} \cup \mathtt{BB})$.

I5 If a bin containing only small items exists, all bins in BS are covered, i. e. $|\mathtt{S}| > 0 \implies |\mathtt{BSP}| = 0$.

Lastly, there are some bin types with bins that are not covered, and we have to make sure that they are not wastefully packed:

I6 If there are small items in BSP, they are all included in the bin containing the biggest item in BSP.

I7 Both S and M each contain at most one bin that is not covered.

This concludes the definition of all invariants. It is easy to see that the invariants all hold in the beginning when no item has arrived yet. Next, we describe the insertion procedures and argue that the invariants are maintained.

Insertion Procedures

We start with the definition of two simple auxiliary procedures used in the following:

- GreedyPush$(i, \mathcal{B})$ is given an item i and a set of bins $\mathcal{B}$. If all the bins contained in $\mathcal{B}$ are covered, it creates a new bin containing item i, and otherwise it inserts i into the most loaded bin that is not covered.
- GreedyPull$(B, \mathcal{B})$ is given bin B and a set of bins $\mathcal{B}$. It successively removes a largest non-big item from a least loaded bin from $\{B' \in \mathcal{B} \,|\, \text{small}(B') \cup \text{medium}(B') \neq \emptyset\}$ and inserts it into B. This is repeated until B is covered or $\mathcal{B}$ does not contain non-big items.

Consider one application of GreedyPull such that B already has a big item. The total size of moved items is smaller than 1. Both procedures are used to insert and repack non-big items. Note that calling GreedyPush for a small item and bin set BSP or S, or a medium item and bin set M, the last two invariants I6 and I7 are maintained. For BSP, the most loaded bin always contains the largest big item, and if there is at least one small item, such a bin is unique. It could happen that as a result of inserting a small item into this bin of BSP the bin is covered and moves to BSC.

For each insertion procedure, we will argue that the invariants are maintained and focus on the critical ones, that is, in each context the invariants, that are not discussed explicitly, trivially hold. For example, we do not discuss I1 in the following, because it will always be easy to see that it is maintained.

Insertion of Small Items. If the arriving item i^* is small, we call GreedyPush$(i^*, \mathtt{BSP})$, if $\mathtt{BSP} \neq \emptyset$, and GreedyPush$(i^*, \mathtt{S})$ otherwise. Insertion into a bin of BSP (the most loaded one) may lead to a covered bin, in which case the bin becomes a bin of BSC (but remains in BS). It is easy to verify, that all invariants, and I5, I6 and I7 in particular, are maintained by this. As mentioned above, I3 holds, as the most loaded bin in BSP always contains the largest big item. Furthermore, there is no migration in this case. The insertion of a medium or big item, however, is more complicated.

Insertion of Big Items. In the case that a big item i^* arrives, we have to be careful where we place it exactly, because, on the one hand, the distributions of big and medium items, that is, invariants I3 and I4, have to be maintained, and, on the other hand, we have to balance out BS and BB (I2). We consider placing the item in BM, BS or BB in this order, i. e. we first try to insert i^* into BM, then into BS and finally into BB. Figure 1 illustrates this.

Insertion into $\mathtt{BM}$**.** We insert i^* into $\mathtt{BM}$, if either $s(i^*) + s(\mathtt{M}) \geq 1$ or $s(i^*) > s_{\min}(\text{big}(\mathtt{BM}))$. Note that the first condition implies $s(i^*) \geq s_{\max}(\text{big}(\mathtt{BB} \cup \mathtt{BS}))$, because of I4, and therefore the insertion of i^* into $\mathtt{BM}$ maintains I3 in both situations. The second condition implies $\mathtt{BM} \neq \emptyset$, because we set $s_{\min}(\emptyset) = +\infty$. In either of these cases, we create a new bin $B^* = \{i^*\}$ and call $\mathsf{GreedyPull}(B^*, \mathtt{M})$, thereby ensuring that I4 is maintained if the new bin is covered. If the first condition did hold, B^* is covered afterwards and we do nothing else. Otherwise, there is a bin $B \in \mathtt{BM}$ containing a big item i with $s(i) = s_{\min}(\text{big}(\mathtt{BM})) < s(i^*)$, and we have $\mathtt{M} = \emptyset$. We remove i from B, yielding $\mathtt{M} = \{B\}$, and call $\mathsf{GreedyPull}(B^*, \mathtt{M})$ a second time. Afterwards, B^* is covered, because $s(i^*) > s(i)$. Furthermore, $s(i) + s(\mathtt{M}) < 1$, because B was barely covered before and the biggest medium item was removed from B due to the second call of $\mathsf{GreedyPull}$. This ensures I4, since by I3, item i is no smaller than any big item packed in $\mathtt{BS}$ or $\mathtt{BB}$. The item i is reinserted using a recursive call to the procedure of inserting a big item. However, item i will not be considered for insertion into $\mathtt{BM}$, because neither the first nor second condition holds for this item, and the other insertion options have no recursive calls for insertion into $\mathtt{BM}$. It is easy to verify that the distribution of medium items in $\mathtt{M}$ (I7) is maintained.

Insertion into $\mathtt{BS}$**.** This step is possible only for item i^* that satisfies $s(i^*) + s(\mathtt{M}) < 1$ and $s(i^*) \leq s_{\min}(\text{big}(\mathtt{BM}))$. Thus, $\mathtt{BM}$ will have the largest big items as required in Invariant I3 after the insertion is performed. In this case there is no recursive call for inserting a big item.

We insert i^* into $\mathtt{BS}$, if either $s(i^*) > s_{\min}(\text{big}(\mathtt{BS}))$ or the following two conditions hold: $s(i^*) \geq s_{\max}(\text{big}(\mathtt{BB}))$ and $|\mathtt{BS}| \leq |\mathtt{BB}|$. Note that $s(i^*) \geq s_{\max}(\text{big}(\mathtt{BB}))$ trivially holds, if $\mathtt{BB} = \emptyset$. Inserting i^* into $\mathtt{BS}$ under these conditions already ensures the correct distribution of big items (I3) with respect to $\mathtt{BS}$ and $\mathtt{BB}$, but we still have to be careful concerning the distribution within the two subsets of $\mathtt{BS}$. The procedure is divided into three simple steps. As a first step, we create a new bin $B^* = \{i^*\}$ and call $\mathsf{GreedyPull}(B^*, \mathtt{S})$. No matter whether B^* is now covered or not, Invariant I5 is satisfied as either B^* is covered and therefore $\mathtt{BSP} = \emptyset$ both before and after the call, or B^* is not covered but now $\mathtt{S} = \emptyset$ (it is possible that both will hold). Note that all properties of the invariants are satisfied, if B^* is already covered. In particular, Invariant I3 holds within $\mathtt{BS}$ because $\mathtt{BSP} = \emptyset$. In the remainder of the second step of the insertion into $\mathtt{BS}$ algorithm we deal with the case that B^* is not covered.

Let $\mathtt{XB}$ denote the set of bins $B \in \mathtt{BS}$ that include small items as well as a big item i with $s(i) < s(i^*)$. Recall that any bin of $\mathtt{BSC}$ has at least one small item, while at most one bin of $\mathtt{BSP}$ has small items.

First, assume that $\mathtt{XB} = \emptyset$ but B^* is not covered. There are two cases. In the first case, at least one item of $\mathtt{S}$ was moved. In this case before we started dealing with i^*, the set $\mathtt{BSP}$ was empty, and now B^* is the unique bin of $\mathtt{BSP}$, and its big item is not larger than those of $\mathtt{BSC}$ (if the last set is not empty) so the invariants I3 and I6 are maintained. In the second case, $\mathtt{S}$ was empty, and B^* is now a bin of $\mathtt{BSP}$ with only a big item. Since $\mathtt{XB} = \emptyset$, adding B^* to $\mathtt{BSP}$ maintains the invariants I3 and I6. Thus, it is left to deal with the case $\mathtt{XB} \neq \emptyset$. In the remainder of the second step of the insertion into $\mathtt{BS}$ algorithm we deal with the case that $\mathtt{XB}$ is not empty.

As B^* is not yet covered, we now have $\mathtt{S} = \emptyset$, and this might have been the case before the call of $\mathsf{GreedyPull}$, in particular if we had $\mathtt{BSP} \neq \emptyset$. Due to the existence of a big item that is smaller than i^* in $\mathtt{BS}$ (such items exist in all bins of $\mathtt{XB}$), we have to be careful in order to maintain the correct distribution of big and small items inside of $\mathtt{BS}$ (I3 and I6).

In the second step, we construct a set of bins $\tilde{\mathtt{B}} \subseteq \mathtt{BS}$ from which small items are removed in order to cover B^*. If $\mathtt{BSP} \cap \mathtt{XB} \neq \emptyset$, this set has exactly one bin (containing small items) by Invariant I6. If such a bin exists, we denote it by B_1. If $\mathtt{BSC} \cap \mathtt{XB} \neq \emptyset$, the set $\mathtt{BSC}$ includes

a bin that contains a big item i' with $s_{\min}(\text{big}(\texttt{BSC})) = s(i') < s(i^*)$ and we denote one such bin (with a big item of minimum size in BSC) by B_2. As $\texttt{XB} \neq \emptyset$, at least one of the bins B_1 or B_2 must exist, but it can also be the case that both exist. Let $\tilde{\texttt{B}}$ be the set of cardinality 1 or 2, which contains these bins. The next operation of the second step is to call GreedyPull($B^*, \tilde{\texttt{B}}$). It is easy to see that no matter whether B^* is covered or not after this operation, the invariants I3 and I6 hold. Specifically, if B_2 does not exist, B^* is not necessarily covered, but I3 and I6 hold as all big items of BSC are not smaller than i^* (as every such bin has at least one small item). If B_2 exists, then B^* keeps receiving items coming first from B_1 and then possibly also from B_2, until it is covered. As the total size of small items of B_2 is sufficient for covering B^* since the big item of B_2 is smaller than i^*, B^* will be covered, so all big items of BSP are not larger than i^*.

Lastly, we describe the third step, which is performed for all cases above, after i^* has been inserted. The insertion of i^* might have violated I2, that is, we now have $|\texttt{BS}| = |\texttt{BB}| + 2$. In this case, we perform the last step, namely, we select two bins $B_3, B_4 \in \texttt{BS}$ with minimal big items, merge the big items into a BB bin and remove and reinsert all small items from B_3 and B_4, using insertion of small items. This yields, $|\texttt{BS}| = |\texttt{BB}| - 1$ and I2 holds.

Insertion into BB. If i^* was not inserted in any of the last steps, it is inserted into BB. In this case, we know from the conditions above that $s(i^*) \leq s_{\min}(\text{big}(\texttt{BS} \cup \texttt{BM}))$, and additionally that $s(i^*) \geq s_{\max}(\text{big}(\texttt{BB}))$ implies $|\texttt{BS}| = |\texttt{BB}| + 1$. We consider two cases.

If $|\texttt{BS}| = |\texttt{BB}| + 1$ (and hence $\texttt{BS} \neq \emptyset$), we select a bin $B \in \texttt{BS}$ with a big item of minimal size. We insert i^* into B to obtain a BB bin and remove and reinsert all small items from B. This yields $|\texttt{BS}| = |\texttt{BB}| - 1$ and I2 holds.

If $|\texttt{BS}| < |\texttt{BB}| + 1$, we have $s(i^*) < s_{\max}(\text{big}(\texttt{BB}))$ (and hence $\texttt{BB} \neq \emptyset$). In this case, we select a bin $B \in \texttt{BB}$ with a big item i of maximal size, insert i^* into B, and remove and reinsert i. Because of its size and invariant I4, the item i will be inserted into BS. Note that in both cases invariant I3 is maintained.

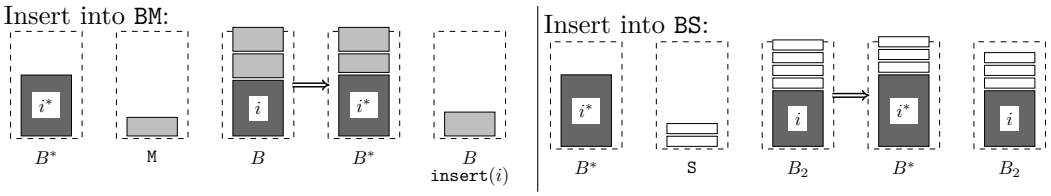

Figure 1 Insertion of a big item i^*. Big items are drawn in dark gray, medium items in light gray, and small items in white.
Insert into BM: open a new bin B^* for i^*; pull M into B^*; pull from bin $B \in \texttt{BM}$ containing the smallest big item i; remove and reinsert i.
Insert into BS: open a new bin B^* for i^*; pull S into B^*; pull from bin $B \in \texttt{BS}$ containing the smallest big item i.

▶ **Lemma 3.** *The overall size of items migrated due to the insertion of a big item i^* is upper bounded by* 11.

Proof sketch. First, note that an insertion into BS can not trigger the reinsertion of a big item. The insertion into BB can only trigger the reinsertion of a single big item into BS and the insertion into BM can only trigger the reinsertion into BB or BS. Hence, each insertion of a big item can trigger at most two other insertions in total and thus only move a total size of 2 this way. The direct reassignments are bounded by 5. ◀

Insertion of Medium Items. If a medium item i^* arrives, GreedyPush$(i^*, \mathtt{M})$ is called. Afterwards, the invariant I4 may be infringed and if this happens, we have $\mathtt{BS} \cup \mathtt{BB} \neq \emptyset$ and $s(\mathtt{M}) \geq 1 - s_{\max}(\mathtt{BS} \cup \mathtt{BB})$, and we continue as follows. We will now describe how to pack a barely covered bin using the items from M and a largest big item from $\mathtt{BS} \cup \mathtt{BB}$ to maintain I4.

If $\mathtt{BS} = \emptyset$, I2 implies that BB contains a single bin B including two items i and i' with $s_{\max}(\mathtt{BS} \cup \mathtt{BB}) = s(i) \geq s(i')$. We remove i' from B, and call GreedyPull$(B, \mathtt{M})$ to create a BM bin. Afterwards, $s(\mathtt{M})$ and $s_{\max}(\mathtt{BS} \cup \mathtt{BB})$ are at most as big as they were before i^* arrived as the first item we pulled from M is at least as big as i^*, and therefore I4 holds. Furthermore $\mathtt{BS} = \mathtt{BB} = \emptyset$ and I2 still holds. Lastly, we reinsert the big item i'.

If, on the other hand, $\mathtt{BS} \neq \emptyset$, the corresponding big item i with $s_{\max}(\mathtt{BS} \cup \mathtt{BB}) = s(i)$ is contained in a bin $B \in \mathtt{BS}$, because of I3. In this case, we remove the small items from B and call GreedyPull$(B, \mathtt{M})$. Afterwards I4 holds, but I2 may be infringed due to the removal of a bin from BS, i. e. $|\mathtt{BB}| = |\mathtt{BS}| + 2$. In this case, we remove the two biggest items i_1 and i_2 from the bins $B_1, B_2 \in \mathtt{BB}$ and if $B_1 \neq B_2$ merge the two bins. This yields $|\mathtt{BB}| = |\mathtt{BS}| + 1$ and I2 holds. Afterwards, we reinsert the two items i_1 and i_2, which both will be inserted in BS due to their sizes. No matter whether we had to rebalance $|\mathtt{BB}|$ and $|\mathtt{BS}|$ or not, we reinsert the removed small items from B as a last step. Figure 2 contains an illustration of this process.

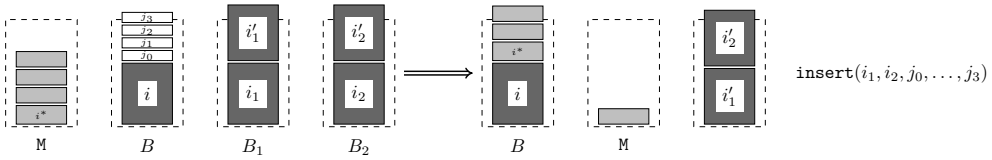

Figure 2 Insertion of a medium item i^*. Big items are drawn in dark gray, medium items in light gray, and small items in white.

▶ **Lemma 4.** *The overall size of items migrated due to the insertion of a medium item i^* is upper bounded by* 27*.*

Analysis. The migration bound stated in Theorem 2 or more precisely $\frac{27}{\varepsilon}$ is already implied by Lemma 3 and Lemma 4, as a medium item has size above ε. It is easy to see that:

▶ Remark 5. The presented algorithm for static bin covering has polynomial running time.

Hence, the only thing left to show is the stated asymptotic competitive ratio:

▶ **Lemma 6.** *The presented algorithm has an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant* 3*.*

Proof. First, we consider the case $\mathtt{BSP} = \emptyset$. In this case, the claim holds because the bins on average have not too much excess size. More precisely, we obviously have $\mathrm{OPT}(I) \leq s(I)$, and invariants I1 and I7 imply $s(I) < 2|\mathtt{BB}| + (1+\varepsilon)|\mathtt{BS}| + 1.5|\mathtt{BM}| + 1.5|\mathtt{MC}| + (1+\varepsilon)|\mathtt{SC}| + 2$. Furthermore, we have $0.5|\mathtt{BB}| \leq 0.5(|\mathtt{BS}|+1)$, due to Invariant I2, and $|\mathtt{BS}| = |\mathtt{BSC}|$, as $\mathtt{BSP} = \emptyset$ holds in the case we are currently considering. Hence $\mathrm{OPT}(I) < (1.5+\varepsilon)(|\mathtt{BB}| + |\mathtt{BSC}| + |\mathtt{BM}| + |\mathtt{MC}| + |\mathtt{SC}|) + 2.5 < (1.5+\varepsilon)\,\mathrm{ALG}(I) + 3$.

A similar argument holds, if $\mathtt{BSP} \neq \emptyset$ but $\mathtt{BB} = \emptyset$. In this case, we have $|\mathtt{BSP}| = |\mathtt{BS}| = 1$, because of invariant I2; and $\mathtt{S} = \emptyset$, because of Invariant I5. Hence $\mathrm{OPT}(I) \leq s(I) < 1.5|\mathtt{BM}| + 1.5|\mathtt{MC}| + 2 = 1.5\,\mathrm{ALG}(I) + 2$.

Next, we consider the case $\texttt{BSP} \neq \emptyset$ and $\texttt{BB} \neq \emptyset$. Here, we have $\texttt{MC} = \emptyset$, because of Invariant I4, and $\texttt{S} = \emptyset$, because of Invariant I5. Note that every bin of $\texttt{BSC} \cup \texttt{BM}$ has at least one item that is not big, since big items have sizes below 1, and these bins are covered. Let $\xi = s_{\max}(\texttt{BSP})$ be the size of a big item from BSP with maximal size. Then all items in $\texttt{BB} \cup \texttt{BSP}$ have size at most ξ (I3) and $\xi > 0.5$. We construct a modified instance I^* as follows:

1. The size of each big item with size below ξ is increased to ξ.
2. Every big item of size larger than ξ is split into a big item of size ξ and a medium or small item, such that the total size of these two items is equal to the size of the original item. Let X be the set of items with sizes of ξ, which we will call ξ-items in the instance after these transformations (X includes also items whose sizes were ξ in I).
3. For each bin from $\texttt{BSC} \cup \texttt{BM}$, select the largest item of I that is not big and call it *special*. By increasing item sizes if necessary, change the sizes of all special items to 0.5. Let Y be the set of special items (whose sizes are now all equal to 0.5). Let Z be the set of the remaining items not belonging to X or Y (in the instance I^* after the transformations, so there may be items that did not exist in I resulting from splitting a big item).

The set of items in I^* is just $X \cup Y \cup Z$. For instance I^*, any bin of BB contains only two items of X. Any bin of BSP has an item of X, and one of these bins may also have small items of Z, but it is not covered. Any bin of $\texttt{BSC} \cup \texttt{BM}$ has one item of X, one item of Y, and possibly items of Z. There may be one uncovered bin of M, containing items of Z.

Note that $\textsc{opt}(I) \leq \textsc{opt}(I^*)$, since any packing for I can be used as a packing for I^* with at least the same number of covered bins. Next, we investigate the relationship between $\textsc{opt}(I^*)$ and the packing of the algorithm for the original instance I. For some optimal solution for I^* without overpacked bins (more than barely covered), let k_2, k_1 and k_0 be the number of covered bins with 2, 1 and 0 items from $X \cup Y$, respectively. Then we have $\textsc{opt}(I^*) = k_2 + k_1 + k_0$ and due to counting $2k_2 + k_1 = |X \cup Y| = (2|\texttt{BB}| + |\texttt{BS}| + |\texttt{BM}|) + (|\texttt{BM}| + |\texttt{BSC}|)$. Since each item in $X \cup Y$ is upper bounded by ξ, we have: $(1-\xi)k_1 + k_0 \leq s(Z)$.

The total size of items (of Z only) packed into the bin of M is below $1 - \xi$ since BSP has a big item of size ξ in I and by Invariant I4, since every item of $\texttt{BS} \cup \texttt{BB}$ is smaller than $1 - s(M)$. For BSP only one bin may contain items of Z by Invariant I6, and this bin has an item of size ξ in I (and it is not covered), so it also has items of Z of total size below $1 - \xi$. Consider a bin of $\texttt{BSC} \cup \texttt{BM}$. The total size of items excluding the special item is the same for I and I^*. Since such a bin is barely covered and for I it has items of one class except for the big item (small or medium), removing the special item results in a load below 1. The total size of items of I^* in such a bin excluding the ξ-item and the special item is below $1 - \xi$.

Therefore, we find that $s(Z) \leq (1-\xi)(|\texttt{BM}| + |\texttt{BSC}| + 2)$. Hence:

$$\begin{aligned} 2\,\textsc{opt}(I) &\leq 2(k_2 + k_1 + k_0) \leq (2k_2 + k_1) + (k_1 + (1-\xi)^{-1}k_0) \\ &\leq (2|\texttt{BB}| + |\texttt{BS}| + |\texttt{BM}| + |\texttt{BM}| + |\texttt{BSC}|) + (|\texttt{BM}| + |\texttt{BSC}| + 2) \\ &\underbrace{\leq}_{\text{Invariant I2}} \quad 3|\texttt{BB}| + 3|\texttt{BM}| + 2|\texttt{BSC}| + 3 \leq 3\,\textsc{alg}(I) + 3. \end{aligned}$$

◀

Extending Our Results

Non-amortized Migration in the Dynamic Case. We are able to extend the result of the static case and show that we can also handle the case of departing items.

▶ **Theorem 7.** *For each $\varepsilon < 1$ with $1/\varepsilon \in \mathbb{Z}$, there is an algorithm* ALG *for dynamic online bin covering with polynomial running time, an asymptotic competitive ratio of $1.5 + \varepsilon$ with additive constant $\mathcal{O}(\log 1/\varepsilon)$, and a non-amortized migration factor of $\mathcal{O}((1/\varepsilon)^5 \log^2(1/\varepsilon))$.*

This is the most elaborate result of the paper and its proof can be found in the long version of the paper (see appendix). In the following, we briefly discuss this the result and give some intuition for the developed techniques.

The main challenge in the dynamic case arises from small items: Let N be some positive integer. Consider the case that N^2 items of size $1/N$ arrived and were placed into N bins, covering each of them perfectly. Next, one item from each bin leaves yielding a solution without any covered bin while the optimum is still $N-1$. Hence, a migration strategy for the small items is needed. Now, coming up with such a strategy to deal with the present example is rather simple, since all the items are of the same size, but in principle small items may differ in size by arbitrary factors. Still, the case with only small items can be dealt with adapting a technique for online bin packing with migration [3]. The basic idea is to sort the items non-increasingly and maintain a packing that corresponds to a partition of this sequence into barely covered bins. If an item arrives, it is inserted into the correct bin and excess items are pushed to the right, that is, to the neighboring bin containing the next items in the ordering, and this process is repeated until the packing is restored. Correspondingly, if an item departs, items are pulled in from the next bin to the right. In this process the arrival or departure of a small item can only cause movements of items at most as big as the original one. While this is useful, it does not suffice to bound migration: Too many bins have to be repacked. In order to deal with this, the bins are partitioned into *chains* of appropriate constant length with a *buffer bin* at the end, which is used to interrupt the migration process. This technique can be applied for the bins $\mathtt{S}$ containing only small items, but for bins $\mathtt{BS}$ containing big items as well problems arise. The main reason for this is that in order to adapt our analysis, we need to cover the bins in $\mathtt{BS}$ containing larger big items with higher priority and furthermore guarantee that there are no (or only few) bins contained in $\mathtt{S}$ if there are bins containing big items that are not covered, i.e., $\mathtt{BSP} \neq \emptyset$. It is not hard to see that spreading one sequence of chains out over the bins of $\mathtt{BS}$ and $\mathtt{S}$ will not suffice.

To overcome these problems, we develop a novel technique: We partition the bins of $\mathtt{BS}$ into few, that is, $\mathcal{O}(\log 1/\varepsilon)$ many, *groups*. Each of the groups is in turn partitioned into *parallel chains* of length $\mathcal{O}(1/\varepsilon)$. The groups are defined such that they comply with a non-increasing ordering of both the big and the small items: the first group contains the largest big and small items, the next group the remaining largest, and so on. A similar ordering holds for each single parallel chain, but no such structure is maintained in between the parallel chains of the same group. Now, whenever a buffer bin of a parallel chain becomes overfilled, items are pushed directly into the next group. However, to maintain the described structure, these have to be the smallest items of the group, and to guarantee this we introduce some additional structure for the buffer bins in each group. While there may be a chain reaction caused by such a push or pull, the migration can still be bounded, because there are only few groups. We are able to guarantee that there is at most one group $\mathtt{G}$ containing uncovered bins and that that all bins of $\mathtt{BS}$ are barely covered, if $\mathtt{S} \neq \emptyset$. These are the essential properties we need in order to adapt our approach to the dynamic case.

Amortized Migration. First of all, we can strengthen the lower bound of Theorem 1 if items can depart and show that the same bound also holds for amortized migration in this case.

▶ **Proposition 8.** *There is no algorithm for dynamic online bin covering with a constant amortized migration factor β and an asymptotic competitive ratio smaller than $3/2$.*

If items never depart, we can use the amortization by repacking the completely instance every once in a while with the help of an AFPTAS of Jansen and Solis-Oba [20]. We can also show that we need to be contended with a non-optimal solution by making use of a highly non-trivial construction.

▶ **Theorem 9.** *For every $\varepsilon > 0$, there is an algorithm for static bin covering with polynomial running time, asymptotic competitive ratio $1 + \varepsilon$, and amortized migration factor $\mathcal{O}(1/\varepsilon)$. Additionally, there is no (possibly exponential time) algorithm for static online bin covering that maintains an optimal solution with constant amortized migration factor β.*

References

1. Susan F. Assmann, David S. Johnson, Daniel J. Kleitman, and Joseph Y.-T. Leung. On a Dual Version of the One-Dimensional Bin Packing Problem. *J. Algorithms*, 5(4):502–525, 1984. doi:10.1016/0196-6774(84)90004-X.
2. János Balogh, Leah Epstein, and Asaf Levin. Lower bounds for online bin covering-type problems. *Journal of Scheduling*, pages 1–11, 2018.
3. Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. *Mathematical Programming*, 2018. doi:10.1007/s10107-018-1325-x.
4. Marie G. Christ, Lene M. Favrholdt, and Kim S. Larsen. Online bin covering: Expectations vs. guarantees. *Theor. Comput. Sci.*, 556:71–84, 2014. doi:10.1016/j.tcs.2014.06.029.
5. János Csirik, David S. Johnson, and Claire Kenyon. Better approximation algorithms for bin covering. In S. Rao Kosaraju, editor, *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 557–566. ACM/SIAM, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365533.
6. János Csirik and V. Totik. Online algorithms for a dual version of bin packing. *Discrete Applied Mathematics*, 21(2):163–167, 1988. doi:10.1016/0166-218X(88)90052-2.
7. Leah Epstein. Online Variable Sized Covering. *Inf. Comput.*, 171(2):294–305, 2001. doi:10.1006/inco.2001.3087.
8. Leah Epstein, Lene M. Favrholdt, and Jens S. Kohrt. Comparing online algorithms for bin packing problems. *J. Scheduling*, 15(1):13–21, 2012. doi:10.1007/s10951-009-0129-5.
9. Leah Epstein, Csanád Imreh, and Asaf Levin. Class Constrained Bin Covering. *Theory Comput. Syst.*, 46(2):246–260, 2010. doi:10.1007/s00224-008-9129-7.
10. Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119(1):33–49, 2009. doi:10.1007/s10107-007-0200-y.
11. Leah Epstein and Asaf Levin. Robust Approximation Schemes for Cube Packing. *SIAM Journal on Optimization*, 23(2):1310–1343, 2013. doi:10.1137/11082782X.
12. Leah Epstein and Asaf Levin. Robust Algorithms for Preemptive Scheduling. *Algorithmica*, 69(1):26–57, 2014. doi:10.1007/s00453-012-9718-3.
13. Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-Dynamic Bin Packing with Little Repacking. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 51:1–51:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.51.
14. Carsten Fischer and Heiko Röglin. Probabilistic Analysis of the Dual Next-Fit Algorithm for Bin Covering. In Evangelos Kranakis, Gonzalo Navarro, and Edgar Chávez, editors, *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, volume 9644 of *Lecture Notes in Computer Science*, pages 469–482. Springer, 2016. doi:10.1007/978-3-662-49529-2_35.
15. Carsten Fischer and Heiko Röglin. Probabilistic Analysis of Online (Class-Constrained) Bin Packing and Bin Covering. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 461–474. Springer, 2018. doi:10.1007/978-3-319-77404-6_34.

16 Waldo Gálvez, José A. Soto, and José Verschae. Symmetry Exploitation for Online Machine Covering with Bounded Migration. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.32.

17 Teofilo F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420010749.

18 Klaus Jansen and Kim-Manuel Klein. A Robust AFPTAS for Online Bin Packing with Polynomial Migration,. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013. doi:10.1007/978-3-642-39206-1_50.

19 Klaus Jansen, Kim-Manuel Klein, Maria Kosche, and Leon Ladewig. Online Strip Packing with Polynomial Migration. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh Srinivas Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.13.

20 Klaus Jansen and Roberto Solis-Oba. An asymptotic fully polynomial time approximation scheme for bin covering. *Theor. Comput. Sci.*, 306(1-3):543–551, 2003. doi:10.1016/S0304-3975(03)00363-3.

21 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online Scheduling with Bounded Migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.

22 Martin Skutella and José Verschae. Robust Polynomial-Time Approximation Schemes for Parallel Machine Scheduling with Job Arrivals and Departures. *Math. Oper. Res.*, 41(3):991–1021, 2016. doi:10.1287/moor.2015.0765.

23 Gerhard J. Woeginger and Guochuan Zhang. Optimal on-line algorithms for variable-sized bin covering. *Oper. Res. Lett.*, 25(1):47–50, 1999. doi:10.1016/S0167-6377(99)00023-1.