View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Fully Dynamic Bin Packing Revisited*

Sebastian Berndt[1], **Klaus Jansen**†[2], **and Kim-Manuel Klein**[2]

1 **Institute of Theoretical Computer Science, Universität zu Lübeck, Germany,** `berndt@tcs.uni-luebeck.de`
2 **Department of Computer Science, Christian-Albrechts-University to Kiel, Germany,** `{kj,kmk}@informatik.uni-kiel.de`

Abstract

We consider the *fully dynamic bin packing* problem, where items arrive and depart in an online fashion and repacking of previously packed items is allowed. The goal is, of course, to minimize both the number of bins used as well as the amount of repacking. A recently introduced way of measuring the repacking costs at each timestep is the *migration factor,* defined as the total size of repacked items divided by the size of an arriving or departing item. Concerning the trade-off between number of bins and migration factor, if we wish to achieve an asymptotic competitive ratio of $1+\epsilon$ for the number of bins, a relatively simple argument proves a lower bound of $\Omega(1/\epsilon)$ on the migration factor. We establish a fairly close upper bound of $O(1/\epsilon^4 \log 1/\epsilon)$ using a new dynamic rounding technique and new ideas to handle small items in a dynamic setting such that no amortization is needed. The running time of our algorithm is polynomial in the number of items n *and* in $1/\epsilon$. The previous best trade-off was for an asymptotic competitive ratio of $5/4$ for the bins (rather than $1+\epsilon$) and needed an amortized number of $O(\log n)$ repackings (while in our scheme the number of repackings is independent of n and non-amortized).

1998 ACM Subject Classification G.2.1 Combinatorial algorithms

Keywords and phrases online, bin packing, migration factor, robust, AFPTAS

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2015.136

1 Introduction

For the classical bin packing problem, we are given a set I of items with a size function $s\colon I \to (0,1]$ and need to pack them into as few unit sized bins as possible. In practice, the complete instance is often not known in advance, which has lead to the definition of a variety of *online* versions of the bin packing problem. First, in the classical *online bin packing* [35], items arrive over time and have to be packed on arrival. Second, in *dynamic bin packing* [8], items also depart over time. This dynamic model is often used for instance in

- the placement and movement of virtual machines onto different servers for cloud computing [3, 4, 22, 23, 32, 36],
- the development of guaranteed quality of service channels over certain multi-frequency time division multiple access systems [28],
- the placement of processes, which require different resources, onto physical host machines [33, 34],
- the resource allocation in a cloud network where the cost depends upon different parameters [9, 26].

---

[*] A full version of this paper is available under `http://arxiv.org/abs/1411.0960`.

[†] Supported by DFG Project "Entwicklung und Analyse von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme," Ja 612/14-1.

© Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein;
licensed under Creative Commons License CC-BY

18th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'15) / 19th Int'l Workshop on Randomization and Computation (RANDOM'15).
Editors: Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim; pp. 135–151
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Name	Deletion	Repacking
Online Bin Packing	✗	✗
Relaxed Online Bin Packing	✗	✓
Dynamic Bin Packing	✓	✗
Fully Dynamic Bin Packing	✓	✓

Figure 1 Overview of online models.

Third and fourth, we may allow already packed items to be slightly rearranged, leading to online bin packing with repacking (known as *relaxed online bin packing*) [14] and dynamic bin packing with repacking (known as *fully dynamic bin packing*) [16]. See Figure 1 for a short overview on the different models.

The amount of repacking can be measured in different ways. We can either count the total number of moved items at each timestep or the sum of the sizes of the moved items at each timestep. If one wants to count the number of moved items, one typically counts a group of tiny items as a single move. A *shifting move* [14] thus involves either a single large item or a bundle of small items in the same bin of total size s with $1/10 \leq s \leq 1/5$. Such a bundle may consists of up to $\Omega(n)$ (very small) items. If an algorithm measures the repacking by shifting moves, an occurring tiny item may lead to a large amount of repacking. In order to guarantee that a tiny item i with size $s(i)$ only leads to a small amount of repacking, one may allow to repack items whose size adds up to at most $\beta \cdot s(i)$. The term β is called the *migration factor* [29]. Note that shifting moves and migration factor are incomparable in the sense that a small migration factor does not imply a small number of shifting moves and vice versa.

In order to measure the quality of an online algorithm, we compare the costs incurred by an online algorithm with the costs incurred by an optimal offline algorithm. An *online algorithm* receives as input a *sequence* of items $I = (i_1, i_2, i_3, \ldots)$ and decides at each timestep t, where to place the item i_t without knowing future items $i_{t+1}, i_{t+2}, \ldots$. We denote by $I(t) = (i_1, i_2, \ldots, i_t)$ the instance containing the first t items of the instance I and by $\textsc{opt}(I(t))$ the minimal number of bins needed to pack all items in $I(t)$. Note that the packings corresponding to $\textsc{opt}(I(t))$ and $\textsc{opt}(I(t+1))$ may differ significantly, as those packings do not need to be consistent. For an online algorithm A, we denote by $A(I(t))$ the number of bins generated by the algorithm on the input sequence $I(t)$. Note that A must make its decision online, while $\textsc{opt}(I(t))$ is the optimal value of the offline instance. The quality of an algorithm for the online bin packing problem is typically measured by its *asymptotic competitive ratio*. An online algorithm A is called an *asymptotic α-competitive algorithm*, if there is a function f with $\lim_{n\to\infty} \sup \left\{ \frac{f(I)}{\textsc{opt}(I)} \mid \textsc{opt}(I) = n \right\} = 0$ such that $A(I(t)) \leq \alpha\, \textsc{opt}(I(t)) + f(I(t))$ for all instances I and all $t \leq |I|$. The minimum α such that A is an asymptotic α-competitive algorithm is called the *asymptotic competitive ratio of A*, denoted by $r^{\mathrm{on}}_{\infty}(A)$, i. e., the ratio is defined as $r^{\mathrm{on}}_{\infty}(A) = \min\{\alpha \mid A \text{ is an asymptotic } \alpha\text{-competitive algorithm}\}$. The online algorithm A thus has a double disadvantage: It does not know future items and we compare its quality to the optimal offline algorithm which may produce arbitrary different packings at time t and time $t+1$. In order to remedy this situation, one may also compare the solution generated by A to a non-repacking optimal offline algorithm. This non-repacking optimal offline algorithm knows the complete instance, but is not allowed to repack, i.e., the solutions at time t and time $t+1$ must be consistent.

In this work, we present new results in fully dynamic bin packing where we measure the quality of an algorithm against a repacking optimal offline algorithm and achieve an asymptotic competitive ratio of $1+\epsilon$. The amount of repacking is bounded by $\mathcal{O}(1/\epsilon^4 \log(1/\epsilon))$. While we measure the amount of repacking in terms of the migration factor, we also prove that our algorithm uses at most $\mathcal{O}(1/\epsilon^4 \log(1/\epsilon))$ shifting moves. Our algorithm runs in time polynomial in the instance size and in $1/\epsilon$.

1.1 Previous Results on Online Variants of Bin Packing

Online Bin Packing

The classical version of the online bin packing problem was introduced by Ullman [35]. In this classical model items arrive over time and have to be packed at their arrival, while *one is not allowed to repack already packed items.* Ullman gave the very first online algorithm FirstFit for the problem and proved that it its absolute competitive ratio is at most 2. The next algorithm NextFit was given by Johnson [19], who proved that its absolute competitive is also at most 2. The analysis of FirstFit was refined by Johnson, Demers, Ullman, Garey and Graham [20], who proved that its asymptotic competitive ratio is at most $17/10$. A revised version of FirstFit, called Revised FirstFit was shown to have asymptotic competitive ratio of at most $5/3$ by Yao [39]. A series of developments of so called *harmonic algorithms* for this problem was started by Lee and Lee [25] and the best known algorithm of this class which has asymptotic competitive ratio at most 1.58889 was given by Seiden [30].

The lower bound on the absolute approximation ratio of $3/2$ also holds for the asymptotic competitive ratio as shown by Yao [39]. This lower bound was first improved independently by Brown [5] and Liang [27] to 1.53635 and subsequently to 1.54014 by van Vliet [37] and finally to 1.54037 by Balogh, Békési and Galambos [1].

Relaxed Online Bin Packing Model

In contrast to the classical online bin packing problem, Gambosi, Postiglione and Talamo [14] considered the online case where one is *allowed to repack items.* They called this model the *relaxed online bin packing model* and proved that the lower bound on the competitive ratio in the classical online bin packing model can be beaten. They presented an algorithm that uses 3 *shifting moves* and has an asymptotic competitive ratio of at most $3/2$, and an algorithm that uses at most 7 shifting moves and has an asymptotic competitive ratio of $4/3$. In another work, Ivković and Lloyd [15] gave an algorithm that uses $\mathcal{O}(\log n)$ *amortized* shifting moves and achieves an asymptotic competitive ratio of $1+\epsilon$. In this amortized setting, shifting moves can be saved up for later use and the algorithm may repack the whole instance sometimes. Epstein and Levin [11] used the measure of the migration factor to give an algorithm that has an asymptotic competitive ratio of $1+\epsilon$ and a migration factor of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$. This result was improved by Jansen and Klein [18] who achieved polynomial migration of $\mathcal{O}(1/\epsilon^4)$ to achieve an asymptotic competitive ratio of $1+\epsilon$.

Concerning lower bounds on the migration factor, Epstein and Levin [11] showed that no optimal solution can be maintained while having a constant migration factor (independent of $1/\epsilon$). Furthermore, Balogh, Békési, Galambos and Reinelt [2] proved that a lower bound on the asymptotic competitive ratio of 1.3877 holds, if one is only allowed to repack a *constant number of items.*

Dynamic Bin Packing

An extension to the classical online bin packing model was given by Coffman, Garey and Johnson [8], called the *dynamic bin packing* model. In addition to the insertion of items,

items also depart over time. *No repacking is allowed* in this model. It is easily seen that no algorithm can achieve a constant asymptotic competitive ratio in this setting. In order to measure the performance of an online algorithm A in this case, they compared the *maximum number of bins used by* A with the *maximum number of bins used by an optimal offline algorithm*, i. e., an algorithm A in this dynamic model is called an *asymptotic* α*-competitive algorithm*, if there is a function f with $\lim_{n\to\infty}\sup\left\{\frac{f(I)}{\text{max-OPT}(I)} \mid \text{max-}\textsc{opt}(I)=n\right\}=0$ where $\text{max-}\textsc{opt}(I)=\max_t \textsc{opt}(I(t))$ such that $\max_t A(I(t)) \leq \alpha \cdot \max_t \textsc{opt}(I(t)) + f(I)$ for all instances I. The minimum of all such α is called the *asymptotic competitive ratio of* A. Coffman, Garey and Johnson modified the FirstFit algorithm and proved that its asymptotic competitive ratio is at most 2.897. Furthermore, they showed a lower bound of 2.5 on the asymptotic competitive ratio when the performance of the algorithm is compared to a repacking optimal offline algorithm, i. e., $\max_t \textsc{opt}(I(t))$.

In the case that the performance of the algorithm is compared to an optimal non-repacking offline algorithm, Coffman, Garey and Johnson showed a lower bound of 2.388. This bound on the non-repacking optimum was improved to 2.428 by Chan, Lam and Wong [6], to 2.5 by Chan, Wong and Yung [7] and finally to $8/3 \approx 2.666$ by Wong, Yung and Burcea [38].

Fully Dynamic Bin Packing

We consider the dynamic bin packing when repacking of already packed items is allowed. This model was first investigated by Ivković and Lloyd [16] and is called *fully dynamic bin packing*. In this model, items arrive and depart in an online fashion and limited repacking is allowed. The quality of an algorithm is measured by the asymptotic competitive ratio as defined in the classical online model (no maximum is taken as in the dynamic bin packing model). Ivković and Lloyd developed an algorithm that uses amortized $\mathcal{O}(\log n)$ many shifting moves (see definition above) to achieve an asymptotic competitive ratio of $5/4$.

Related Results on the Migration Factor

Since the introduction of the migration factor, several problems were considered in this model and different algorithms for these problems have been developed. Following the terminology of Sanders, Sivadasan and Skutella [29] we sometimes use the term *approximation ratio* instead of competitive ratio. Hence, we also use the terms asymptotic polynomial time approximation scheme (APTAS) and asymptotic fully polynomial time approximation scheme (AFPTAS) in the context of online algorithms. If the migration factor of an algorithm A only depends upon the approximation ratio ϵ and not on the size of the instance, we call *A robust*.

In the case of online bin packing, Epstein and Levin [11] developed the first robust APTAS for the problem using a migration factor of $2^{\mathcal{O}((1/\epsilon^2)\log(1/\epsilon))}$. They also proved that there is no online algorithm for this problem that has a constant migration factor and that maintains an optimal solution. The APTAS by Epstein and Levin was later improved by Jansen and Klein [18], who developed a robust AFPTAS for the problem with migration factor $\mathcal{O}(1/\epsilon^4)$. In their paper, they developed new linear program (LP)/integer linear program (ILP) techniques, which we make use of to obtain polynomial migration. It was shown by Epstein and Levin [12] that their APTAS for bin packing can be generalized to packing d-dimensional cubes into unit cubes. Sanders, Sivadasan and Skutella [29] developed a robust polynomial time approximation scheme (PTAS) for the scheduling problem on identical machines with a migration factor of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$. Skutella and Verschae [31] studied the problem of maximizing the minimum load given n jobs and m identical machines. They also considered a dynamic setting, where jobs may also depart. They showed that there is no robust PTAS for this machine covering problem with constant migration. The main reason for the nonexistence is due to very small jobs. By using an amortized migration factor, they developed a PTAS for the problem with amortized migration of $2^{\mathcal{O}((1/\epsilon)\log^2(1/\epsilon))}$.

1.2 Our Contributions

Main Result

In this work, we investigate the *fully dynamic bin packing* model. We measure the amount of repacking by the *migration factor*; but our algorithm uses a bounded number of shifting moves as well. Since the work of Ivković and Lloyd from 1998 [16], no progress was made on the fully dynamic bin packing problem concerning the asymptotic competitive ratio of $5/4$. It was also unclear whether the number of shifting moves (respectively migration factor) must depend on the number of packed items n. In this paper we give positive answers for both of these concerns. We develop an algorithm that provides at each time step t an approximation guarantee of $(1+\epsilon)\,\mathrm{OPT}(I(t)) + \mathcal{O}(1/\epsilon \log(1/\epsilon))$. The algorithm uses a migration factor of $\mathcal{O}(1/\epsilon^4 \cdot \log(1/\epsilon))$ by repacking at most $\mathcal{O}(1/\epsilon^3 \cdot \log(1/\epsilon))$ bins. Hence, the generated solution can be arbitrarily close to the optimum solution, and for every fixed ϵ the provided migration factor is constant (it does not depend on the number of packed items). The running time is polynomial in n and $1/\epsilon$. In case that no deletions are used, the algorithm has a migration factor of $\mathcal{O}(1/\epsilon^3 \cdot \log(1/\epsilon))$, which beats the best known migration factor of $\mathcal{O}(1/\epsilon^4)$ by Jansen and Klein [18]. Since the number of repacked bins is bounded, so is the number of shifting moves as it requires at most $O(1/\epsilon)$ shifting moves to repack a single bin. Furthermore, we prove that there is no asymptotic approximation scheme for the online bin packing problem with a migration factor of $o(1/\epsilon)$ even in the case that no items depart (and even if $\mathcal{P} = \mathcal{NP}$).

Technical Contributions

We use the following techniques to achieve our results:

- In order to obtain a lower bound on the migration factor in Section 2, we construct a series of instances that provably need migration of $\Omega(1/\epsilon)$ in order to have an asymptotic approximation ratio of $1+\epsilon$.
- In Section 3, we show how to handle large items in a fully dynamic setting. The fully dynamic setting involves more difficulties in the rounding procedure, in contrast to the setting where large items may not depart, treated in [18]. A simple adaption of the dynamic techniques developed in [18] does not work (see introduction of Section 3). We modify the offline rounding technique by Karmarkar and Karp [24] such that a feasible rounding structure can be maintained when items are inserted or removed. This way, we can make use of the LP-techniques developed in Jansen and Klein [18].
- In Section 4, we explain how to deal with small items in a dynamic setting. In contrast to the setting where departure of items is not allowed, the fully dynamic setting provides major challenges in the treatment of small items. An approach is thus developed where small items of similar size are packed near each other. We describe how this structure can be maintained as new items arrive or depart. Note that the algorithm of Ivković and Lloyd [16] relies on the ability to manipulate up to $\Omega(n)$ very small items in constant time. See also their updated work for a thorough discussion of this issue [17].
- In order to unify the different approaches for small and large items in Section 4.2, we develop an advanced structure for the packing. We give novel techniques and ideas to manage this mixed setting of small and large items. The advanced structure makes use of a potential function, which bounds the number of bins that need to be reserved for incoming items.

Several proofs in this extended abstract are removed due to space constraints. The full version[1] contains all of these proofs.

2 Lower Bound

We start our investigation by analyzing the connection between the approximation ratio and the migration factor of an algorithm. Intuitively, one would expect that a small migration factor c leads to a worse approximation ratio r_c. Whether the dependency between those term is logarithmic, linear, quadratic or something completely different is unclear. A simple argument shows that the dependency is at least linear, i. e., there is no robust asymptotic approximation scheme for bin packing with migration factor of $o(1/\epsilon)$, even if $\mathcal{P} = \mathcal{NP}$. This improves upon the lower bound given by Epstein and Levin [11], which states that no algorithm for online bin packing, that maintains an optimal solution can have constant migration.

▶ **Theorem 1.** *For a fixed migration factor $\gamma > 0$, there is no robust approximation algorithm for bin packing with asymptotic approximation ratio better than $1 + \frac{1}{6\lceil\gamma\rceil+5}$.*

▶ **Corollary 2.** *There is no robust asymptotic approximation scheme for bin packing with a migration factor $\gamma \leq 1/6(1/\epsilon - 11) = \Theta(1/\epsilon)$.*

3 Dynamic Rounding

The goal of this section is to give a robust AFPTAS for the case that only large items arrive and depart. In the first subsection we present a general rounding structure. In the second subsection we give operations that modify the rounding in a way that the general structure is preserved. We give the final algorithm for the insertion and departure of large items in Section 3.3. Finally, the correctness is proved by using the LP/ILP techniques developed in [18].

In [18], the last two authors developed a dynamic rounding technique based on an offline rounding technique from Fernandez de la Vega and Lueker [13]. However, a simple adaption of these techniques does not work in the dynamic case where items may also depart. In the case of the offline rounding by Fernandez de la Vega and Lueker, items are sorted and then collected in groups of the same cardinality. As a new item arrives in an online fashion, this structure can be maintained by inserting the new item to its corresponding group. By shifting the largest item of each group to the left, the cardinality of each group (except for the first one) can be maintained. However, shifting items to the right whenever an item departs leads to difficulties in the LP/ILP techniques. As the rounding for a group may increase, patterns of the existing LP/ILP solution might become infeasible. We overcome these difficulties by developing a new dynamic rounding structure and operations based on the offline rounding technique by Karmarkar and Karp [24]. We felt that this dynamic rounding technique is easier to analyze since the structure can essentially be maintained by shifting items, instead of the use of multiple complex operations as in [18].

A bin packing instance consists of a set of *items* $I = \{i_1, i_2, \ldots, i_n\}$ with *size function* $s : I \to [0, 1] \cap \mathbb{Q}$. A feasible solution is a partition $B^1, \ldots, B^k$ of I such that $\sum_{i \in B^j} s(i) \leq 1$ for $j = 1, \ldots, k$. We call a partition $B^1, \ldots, B^k$ a *packing* and a single set B^j is called a *bin*. The goal is to find a solution with a minimal number of bins. If the item i is packed into the bin B^j, we write $B(i) = j$. The smallest value $k \in \mathbb{N}$ such that a packing with k bins exists is denoted by $\text{OPT}(I, s)$ or if the size function is clear by $\text{OPT}(I)$. A trivial, yet useful lower bound on $\text{OPT}(I, s)$ is given by the value $\text{SIZE}(I, s) = \sum_{i \in I} s(i)$.

---

[1] The full version is available at `http://arxiv.org/abs/1411.0960`.

3.1 Rounding

First, we divide the set of items into *small* ones and *large* ones. An item i is called *small* if $s(i) < \epsilon/14$, otherwise it is called *large*. Instance I is partitioned accordingly into a set of large items I_L and a set of small items I_S. We treat small items and large items differently. Small items can be packed using an algorithm presented in Section 4.1 while large items will be assigned using an ILP. In this section we discuss how to handle large items.

To obtain a LP formulation of fixed (independent of $|I|$) dimension, we use a rounding technique based on the offline AFPTAS by Karmarkar and Karp [24]. Here, large items are put into categories such that each item in category ℓ has size $\in (2^{-(\ell+1)}, 2^{-\ell}]$. In each of those categories, the items are sorted by their size and the first $2^\ell \cdot k$ items are put into the first group, the second $2^\ell \cdot k$ items into the second group and so on. The size of every item in a group is rounded to the maximal size in this group. Depending on a suitable choice of k, there are at most $\mathcal{O}(1/\epsilon \log(1/\epsilon))$ groups and the additive error produced by the rounding is bounded by ϵ.

In order to use this rounding technique for our dynamic setting, we generalize their rounding by having groups of size $2^\ell \cdot k$ (the A-block) and groups of size $2^\ell(k-1)$ (the B-block). This generalized rounding has a certain structure that is maintained throughout the algorithm and guarantees an approximate solution for the original instance. As in [24], we characterize the set of large items more precisely by their sizes. We say that an item i is in *size category* ℓ if $s(i) \in (2^{-(\ell+1)}, 2^{-\ell}]$. Denote the set of all size categories by W. As every large item has size at least $\epsilon/14$, the number of size categories is bounded by $|W| \leq \log(1/\epsilon) + 5$. Next, items of the same size category are characterized by their *block*, which is either A or B and their *position* $r \in \mathbb{N}$ in this block. Therefore, we partition the set of large items into a set of groups $G \subseteq W \times \{A, B\} \times \mathbb{N}$. A group $g \in G$ thus consists of a triple (ℓ, X, r) with size category $\ell \in W$, block $X \in \{A, B\}$ and position $r \in \mathbb{N}$. The *rounding function* $R : I_L \mapsto G$ maps each large item $i \in I_L$ to a *group* $g \in G$. By $g[R]$ we denote the set of items being mapped to the group g, i. e., $g[R] = \{i \in I_L \mid R(i) = g\}$.

Let $q(\ell, X)$ be the maximal $r \in \mathbb{N}$ such that $|(\ell, X, r)[R]| > 0$, i.e., $q(\ell, X)$ is the last position in block X (with respect to the size category ℓ). If (ℓ, X_1, r_1) and (ℓ, X_2, r_2) are two different groups, we say that (ℓ, X_1, r_1) is *left* of (ℓ, X_2, r_1), if $X_1 = A$ and $X_2 = B$ or $X_1 = X_2$ and $r_1 < r_2$. We say that (ℓ, X_1, r_1) is *right* of (ℓ, X_2, r_2) if it is not left of it.

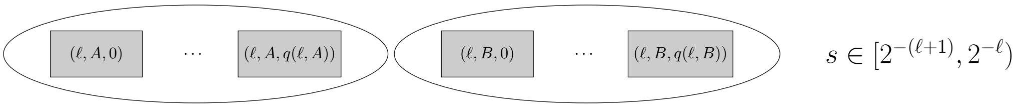


Figure 2 Grouping in $(\ell, A, \cdot)$ and $(\ell, B, \cdot)$.

Given an instance (I, s) and a rounding function R, we define the rounded size function s^R by rounding the size of every large item i up to the size of the largest item in its group, i.e. $s^R(i) = \max_{i'} \{s(i') \mid R(i') = R(i)\}$. We denote by $\text{OPT}(I, s^R)$ the value of an optimal solution of the rounded instance (I, s^R).

Depending on a parameter k, we define the following properties for a rounding function R. Property (a) guarantees that the items are categorized correctly according to their sizes.

Property (b) guarantees that items of the same size category are sorted by their size and properties (c) and (d) define the number of items in each group.

(a) For each $i \in (\ell, X, r)[R]$ we have $2^{-(\ell+1)} < s(i) \leq 2^{-\ell}$.

(b) For each $i \in (\ell, X, r)[R]$ and each $i' \in (\ell, X, r')[R]$ and $r < r'$, we have $s(i) \geq s(i')$.

(c) For each $\ell \in W$ and $1 \leq r \leq q(\ell, A)$ we have that $|(\ell, A, r)[R]| = 2^{\ell}k$ and $|(\ell, A, 0)[R]| \leq 2^{\ell}k$.

(d) For each $\ell \in W$ and $0 \leq r \leq q(\ell, B) - 1$ we have $|(\ell, B, r)[R]| = 2^{\ell}(k-1)$ and $|(\ell, B, q(\ell, B))[R]| \leq 2^{\ell}(k-1)$.

The following lemma shows that the number of groups is bounded and the rounding function does in fact yield a $(1+\epsilon)$-approximation for a suitable choice of k.

▶ **Lemma 3.** *If* $\textsc{size}(I_L, s) > {}^{8}\!/\!_{\epsilon} \cdot (\lceil \log({}^{1}\!/\!_{\epsilon}) \rceil + 5)$ *and* $k = \left\lfloor \frac{\textsc{size}(I_L, s) \cdot \epsilon}{2(\lfloor \log(1/\epsilon) \rfloor + 5)} \right\rfloor$*, the number of non-empty groups is bounded by* $\mathcal{O}({}^{1}\!/\!_{\epsilon} \log({}^{1}\!/\!_{\epsilon}))$.

▶ **Lemma 4.** *Given an instance* (I_L, s) *with items greater than* $\epsilon/14$ *and a rounding function* R *fulfilling properties (a) to (d), then* $\textsc{opt}(I_L, s^R) \leq (1+\epsilon) OPT(I_L, s)$.

3.2 Rounding Operations

Let us consider the case where large items arrive and depart in an online fashion. Formally this is described by a sequence of pairs $(i_1, A_1), \ldots, (i_n, A_n)$ where $A_i \in \{\text{Insert}, \text{Delete}\}$. At each time $t \in \{1, \ldots, n\}$ we need to pack the item i_t into the corresponding packing of $i_1, \ldots, i_{t-1}$ if $A_i = \text{Insert}$ or remove the item i_t from the corresponding packing of $i_1, \ldots, i_{t-1}$ if $A_i = \text{Delete}$. We will denote the instance $i_1, \ldots, i_t$ at time t by $I(t)$ and the corresponding packing by B_t. We will also round our items and denote the rounding function at time t by R_t. The large items of $I(t)$ are denoted by $I_L(t)$. At time t we are allowed to repack several items with a total size of $\beta \cdot s(i_t)$ but we intend to keep the migration factor β as small as possible. The term $\text{repack}(t) = \sum_{i, B_{t-1}(i) \neq B_t(i)} s(i)$ denotes the sum of the items which are moved at time t, the *migration factor* β of an algorithm is then defined as $\max_t \{{}^{\text{repack}(t)}\!/\!_{s(i_t)}\}$. As the value of $\textsc{size}$ will also change over the time, we define the value $\kappa(t)$ as

$$\kappa(t) = \frac{\textsc{size}(I_L(t), s) \cdot \epsilon}{2(\lfloor \log({}^{1}\!/\!_{\epsilon}) \rfloor + 5)}.$$

As shown in Lemma 3, we will make use of the value $k(t) := \lfloor \kappa(t) \rfloor$.

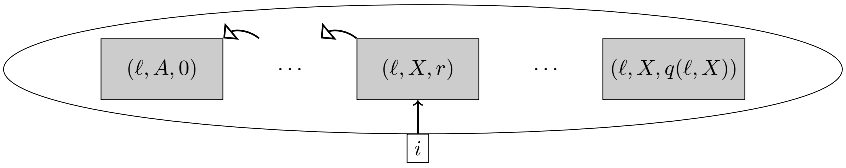
We present operations that transform the current rounding R_t, the current packing B_t with its corresponding LP/ILP solutions into a new rounding R_{t+1}, a new packing B_{t+1} and new corresponding LP/ILP solutions for the new instance $I(t+1)$. At every time t the rounding R_t maintains properties (a) to (d). Therefore the rounding provides an asymptotic approximation ratio of $1+\epsilon$ (Lemma 4) while maintaining only $\mathcal{O}({}^{1}\!/\!_{\epsilon} \log({}^{1}\!/\!_{\epsilon}))$ many groups (Lemma 3). We will now present a way how to adapt this rounding to a dynamic setting, where items arrive or depart online.

Our rounding R_t is manipulated by different *operations*, called the *insert, delete, shiftA* and *shiftB* operation. Some ideas behind the operations are inspired by the operations described by Epstein and Levin [11]. The insert operation is performed whenever a large item arrives and the delete operation is performed whenever a large item departs. The shift operations are used to modify the number of groups that are contained in the A and B block. As we often need to filter the largest items of a group g in a rounding R, we denote this item by $\lambda(g, R)$. Due to space constraints we do not describe how the LP/ILP solutions are modified by the operations (see full version). Intuitively, an insert operation finds the group

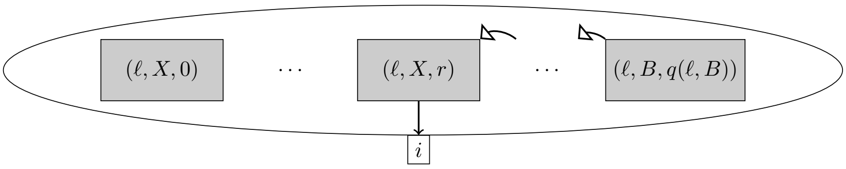
g, where the new item needs to be placed and inserts it into g. In order to maintain the size of g, the largest item from g is shifted to its left neighbour g'. The largest item of g' is then shifted to its left neighbour and so on. The deletion algorithm removes the item from its group g. In order to maintain the size of g, the largest item from its right neighbour g' is shifted into g. The largest item of the right neighbour of g' is then shifted into g' and so on.

- Insert: To insert item i_t, find the corresponding group (ℓ, X, r) with
  - $s(i_t) \in (2^{-(\ell+1)}, 2^{-\ell}]$,
  - $\min_i \{s(i) \mid i \in (\ell, X, r-1)\} > s(i_t)$ and
  - $s(\lambda((\ell, X, r+1), R)) \leq s(i_t)$.

  We will then insert i_t into (ℓ, X, r) and get the rounding R' by shifting the largest element of (ℓ, X, r) to $(\ell, X, r-1)$ and the largest item of $(\ell, X, r-1)$ to $(\ell, X, r-2)$ and so on until the first group $(\ell, A, 0)$ is reached.



(a) Insert i into $(\ell, X, \cdot)$



(b) Delete i from $(\ell, X, \cdot)$

Figure 3 Insert and Delete.

- Delete: To delete an item i_t that is in group (ℓ, X, r), we remove i_t from this group and move the largest item from $(\ell, X, r+1)$ into (ℓ, X, r) and the largest item from $(\ell, X, r+2)$ into $(\ell, X, r+1)$ and so on until the last group $(\ell, B, q(\ell, B))$ is reached.

To control the number of groups in A and B we introduce operations shiftA and shiftB that increase or decrease the number of groups in A respectively B. An operation shiftA increases the number of groups in A by 1 and decreases the number of groups in B by 1 by shifting a group from block B to block A. The operation shiftB moves a group From block A to block B. The ability to modify the blocks is needed later on, as the value k used by the rounding changes over time.

- shiftA: In order to move a group from B to A, shift exactly 2^ℓ items from $(\ell, B, q(\ell, B))$ to $(\ell, B, q(\ell, B) - 1)$. Then shift exactly 2^ℓ items from $(\ell, B, q(\ell, B) - 1)$ to $(\ell, B, q(\ell, B) - 2)$ and so on until $(\ell, B, 0)$ is reached. The group $(\ell, B, 0)$ has now the same size as the groups in $(\ell, A, \cdot)$. We transfer $(\ell, B, 0)$ to block A. Note that the total size of the 2^ℓ items is bounded by 1.
- shiftB: In order to move a group from A to B, shift 2^ℓ items from $(\ell, A, q(\ell, A))$ to $(\ell, A, q(\ell, A) - 1)$. Then shift exactly 2^ℓ items from $(\ell, A, q(\ell, A) - 1)$ to $(\ell, A, q(\ell, A) - 2)$ and so on until $(\ell, A, 0)$ is reached. The group $(\ell, A, q(\ell, A))$ has now the same size as the groups in $(\ell, B, \cdot)$. We transfer $(\ell, A, q(\ell, A))$ to block B.

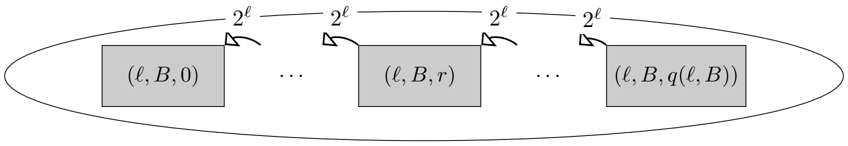


Figure 4 shiftA.

The next lemma shows that all of these operations maintain the desired properties of the rounding.

▶ **Lemma 5.** *Let R be a rounding function fulfilling properties (a) to (d). Applying any of the operations insert, delete, shiftA or shiftB on R results in a rounding function R' fulfilling properties (a) to (d).*

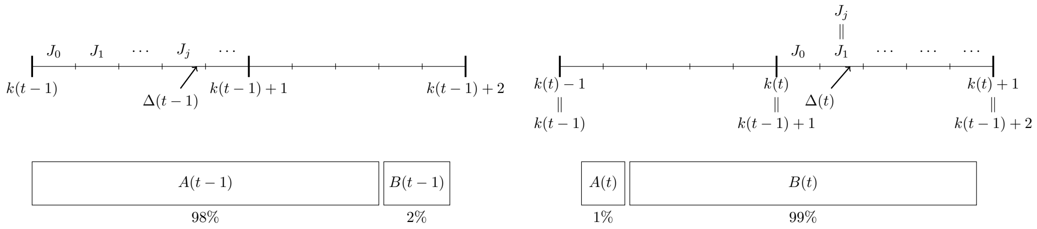
According to Lemma 3 the rounded instance (I, s^R) has $\mathcal{O}(1/\epsilon \log(1/\epsilon))$ different item sizes (given a suitable k). Using the LP formulation of Eisemann [10], the resulting LP called $LP(I, s^R)$ has $m = \mathcal{O}(1/\epsilon \log(1/\epsilon))$ constraints. We say a packing B *corresponds* to a rounding R and an integral solution y of the ILP if all items in (I, s^R) are packed by B according to y. The operations also modify these ILP solutions.

▶ **Lemma 6.** *Applying any of the operations insert, delete, shiftA or shiftB on a rounding R and ILP solution y with corresponding packing B defines a new rounding R' and a new integral solution y'. The solution y' is a feasible solution of the linear program $LP(I, s^{R'})$.*

3.3 Algorithm for Dynamic Bin Packing

We will use the operations from the previous section to obtain a dynamic algorithm for bin packing with respect to large items. The operations insert and delete are designed to process the input depending of whether an item is to be inserted or removed. Keep in mind that the parameter $k(t) = \lfloor \kappa(t) \rfloor = \left\lfloor \frac{\text{SIZE}(I_L(t)) \cdot \epsilon}{2(\lfloor \log(1/\epsilon) \rfloor + 5)} \right\rfloor$ changes over time as $\text{SIZE}(I_L(t))$ may increase or decrease. In order to fulfill the properties (c) and (d), we need to adapt the number of items per group whenever k changes. The shiftA and shiftB operations are thus designed to manage the dynamic number of items in the groups as k changes. Note that a group in the A-block with parameter k has by definition the same number of items as a group in the B-block with parameter $k-1$ if they are in the same size category. If k increases, the former A block is treated as the new B block in order to fulfill the properties (c) and (d) while a new empty A block is introduced. To be able to rename the blocks, the B block needs to be empty. Accordingly the A block needs to be empty if k decreases in order to treat the old B block as new A block. Hence we need to make sure that there are no groups in the B-block if k increases and vice versa, that there are no groups in the A-block if k decreases.

We denote the number of all groups in the A-blocks at time t by $A(t)$ and the number of groups in B-blocks at time t by $B(t)$. To make sure that the B-block (respectively the A-block) is empty when k increases (decreases) the ratio $\frac{A(t)}{A(t)+B(t)}$ needs to correlate to the fractional digits of $\kappa(t)$ at time t denoted by $\Delta(t)$. For example, if $\Delta(t) = 0.98$, nearly every group must be in an A-block, as the B-blocks need to be empty if $k(t)$ increases (see Figure 5(a) for a sketch of the situation). Note that the term $\frac{A(t)}{A(t)+B(t)}$ is 0 if the A-block is empty and the term is 1 if the B-block is empty. This way, we can make sure that as soon as $k(t)$ increases, the number of B-blocks is close to 0 and as soon as $k(t)$ decreases, the number of A-blocks is close to 0. Therefore, the blocks can be renamed whenever $k(t)$



(a) Before Insert **(b)** After Insert

Figure 5 Comparison of the situation before and after an Insert Operation.

changes. Hence we partition the interval $[0,1)$ into exactly $A(t)+B(t)$ smaller intervals $J_i = \left[\frac{i}{A(t)+B(t)}, \frac{i+1}{A(t)+B(t)}\right)$. We will make sure that $\Delta(t) \in J_i$ iff $\frac{A(t)}{A(t)+B(t)} \in J_i$.

The algorithm inserts an item via the insert operation and then uses shiftA and shiftB operations to adjust the number of A- and B-blocks. Recall that a shiftA operation reduces the number of groups in the B-block by 1 and increases the number of groups in the A-block by 1 (shiftB works vice versa).

In the following algorithm we also make use of an algorithm called IMPROVE, which was developed in [18] to reduce the number of used bins. Using IMPROVE(x) on a packing B with approximation guarantee $\max_i B(i) \leq (1+\bar{\epsilon})\,\mathrm{OPT} + C$ for some $\bar{\epsilon} = \mathcal{O}(\epsilon)$ and some additive term C yields a new packing B' with approximation guarantee $\max_i B(i) \leq (1+\bar{\epsilon})\,\mathrm{OPT} + C - x$. We use the operations in combination with IMPROVE to obtain a fixed approximation guarantee. The similar deletion algorithm can be found in the full version.

▶ **Algorithm 1** (AFPTAS for large items).

```
Algorithm: Insertion
if SIZE(I(t)) $< (m+2)(1/\delta + 2)$ or SIZE(I(t)) $< 8(1/\delta + 1)$ then
    use offline Bin Packing;
else
    IMPROVE(2); insert(i);
    // Shifting to the correct interval
    Let $J_i$ be the interval containing $\Delta(t)$;
    Let $J_j$ be the interval containing $\frac{A(t)}{A(t)+B(t)}$;
    Set $d = i - j$;
    if $k(t) > k(t-1)$ then // Modulo $A(t)+B(t)$ when $k$ increases
        $d = d + (A(t)+B(t))$;
    for $p := 0$ to $|d| - 1$ do // Shifting $d$ groups from $B$ to $A$
        if $i+p = A(t) + B(t)$ then
            Rename$(A, B)$;
        IMPROVE(1); shiftA;
```

Note that as exactly $d = i - j$ groups are shifted from A to B (or B to A) we have by definition that $\Delta(t) \in \left[\frac{A(t)}{A(t)+B(t)}, \frac{A(t)+1}{A(t)+B(t)}\right)$ at the end of the algorithm. The following lemmas prove that the algorithm works as expected by moving only a constant number of groups.

▶ **Lemma 7.** *At most* 11 *groups are shifted from A to B (or B to A) in Algorithm 1.*

▶ **Lemma 8.** *Every rounding function* R_t *produced by Algorithm 1 fulfills properties* (a) *to* (d) *with parameter* $k(t) = \left\lfloor \frac{\textsc{size}(I_L(t)) \cdot \epsilon}{2(\lceil \log(1/\epsilon) \rceil + 5)} \right\rfloor$.

Using analysing techniques developed in [18] we prove the following theorem.

▶ **Theorem 9.** *Algorithm 1 is an AFPTAS with migration factor* $\mathcal{O}(\frac{1}{\epsilon^3} \cdot \log(1/\epsilon))$ *for the fully dynamic bin packing problem with respect to large items.*

If no deletions are present, we can use a simple FirstFit algorithm (as described by Jansen and Klein [18]) to pack the small items into the bins. This does not change the migration factor or the running time of the algorithm and we obtain a robust AFPTAS with $\mathcal{O}(\frac{1}{\epsilon^3} \cdot \log(1/\epsilon))$ migration for the case that no items depart. This improves the best known migration factor of $\mathcal{O}(\frac{1}{\epsilon^4})$ [18].

4 Handling Small Items

In this section we present methods for dealing with arbitrary small items in a dynamic setting. First, we present a robust AFPTAS with migration factor of $\mathcal{O}(1/\epsilon)$ for the case that only small items arrive and depart. In Section 4.2 we generalize these techniques to a setting where small items arrive into a packing, that already contains large items which can not be rearranged. Finally we state the AFPTAS for the general fully dynamic bin packing problem. In a robust setting without departing items, small items can easily be treated by packing them greedily via the classical FirstFit algorithm of Johnson et al. [21] (see Epstein and Levin [11] or Jansen and Klein [18]). However, in a setting where items may also depart, small items need much more attention. We show in the full version that the FirstFit algorithm does not work in this dynamic setting.

▶ **Lemma 10.** *Using the* FirstFit *algorithm to pack small items may lead to an arbitrarily bad approximation.*

4.1 Only Small Items

We consider a setting where only small items exist, i.e., items with a size less than $\epsilon/14$. One way to overcome the problematic instances, like those in Lemma 10, is to make sure that the small items remain ordered. If one has m bins $b_1, b_2, \ldots, b_m$, one can make sure that the larger small items are in bins with lower indices as the smaller ones. A possible way to maintain such a property works as follows: Whenever a small items i arrives, find the bin with the smallest index that contains a set J of small items such that $\sum_{j \in J} s(j) \geq s(i)$. Remove those items in J from the bin and add i in this place. Insert all of the items in J in the same way. The deletion algorithm is very similar, but searches for a set J from smaller sizes, that replaces the departing item i. In order to bound the migration of this operation, we declare every $1/\epsilon$-th bin as buffer bin and terminate the procedure at the buffer bin. If the moved items do not fit into the buffer bin, we declare the buffer bin as normal and open a new buffer bin containing the remaining items. As only a fraction of ϵ of the bins are buffer bins, this worsen the approximation ratio only by ϵ. The migration remains bounded as at most $1/\epsilon$ bins are changed.

Formally, we divide the set of small items into different size intervals S_j where $S_j = \left[\frac{\epsilon}{2^{j+1}}, \frac{\epsilon}{2^j}\right)$ for $j \geq 1$. Let $b_1, \ldots, b_m$ be the used bins of our packing. We say a size category S_j is bigger than a size category S_k if $j < k$, i.e., the item sizes contained in S_j are larger (note that a size category S_j with large index j is called small). We say a bin b is filled completely

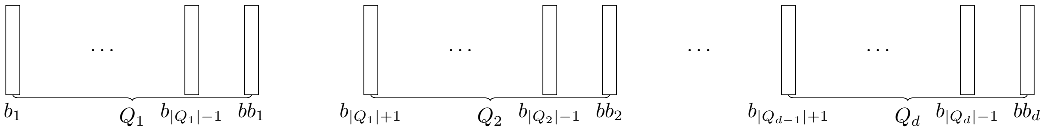


Figure 6 Distribution of bins with small items into queues.

if it has less than $\frac{\epsilon}{2^j}$ remaining space, where S_j is the biggest size category appearing in b, i.e., no item from the categories $S_1, S_2, \ldots, S_j$ fit into b. Furthermore we label bins b as *normal* or as *buffer bins* and partition all bins $b_1, \ldots, b_m$ into *queues* $Q_1, \ldots, Q_d$. A queue is a consecutive subsequence of bins $b_i, b_{i+1} \ldots, b_{i+c}$ where bins $b_i, \ldots, b_{i+c-1}$ are normal bins and bin b_{i+c} is a buffer bin. We denote the number of bins in Q_i by $|Q_i|$. The buffer bin of queue Q_i is denoted by bb_i. See Figure 6 for a sketch of the situation.

We will maintain a special form for the packing of small items such that the following properties are always fulfilled. For the sake of simplicity, we assume that $1/\epsilon$ is integral.

1. For every item $i \in b_d$ with size $s(i) \in S_j$ for some $j, d \in \mathbb{N}$, there is no item $i' \in b_{d'}$ with size $s(i') \in S_{j'}$ such that $d' > d$ and $j' > j$. This means: Items are ordered from left to right by their size intervals.
2. Every normal bin is filled completely.
3. The length of each queue is at least $1/\epsilon$ and at most $2/\epsilon$ except for the last queue Q_d.

Note that property (1) implies that all items in the same size interval S_j are packed into consecutive bins $b_x, b_{x+1}, \ldots, b_{x+c}$. Items in the next smaller category S_{j+1} are then packed into bins $b_{x+c}, b_{x+c+1}, \ldots$ and so on. We denote by $b_{S(j)}$ the last bin in which an item of category S_j appears.

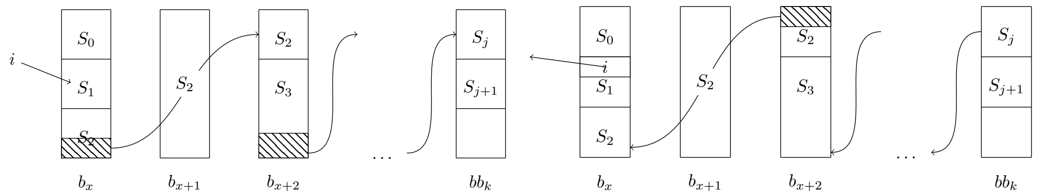
The following lemma guarantees that a packing with properties (1) to (3) is close to the optimum solution.

▶ **Lemma 11.** *If properties* (1) *to* (3) *hold, then at most* $(1 + \mathcal{O}(\epsilon))\,\text{OPT}(I, s) + 2$ *bins are used in the packing for every* $\epsilon \leq 1/3$.

We will now describe the operations that are applied whenever a small item is inserted or removed from the packing. The operations are designed such that properties (1) to (3) are maintained. Lemma 11 thus guarantees a good approximation ratio at every step of the algorithm. The operations are applied recursively such that some items from each size interval are shifted from left to right (insert) or right to left (delete). The recursion halts if the first buffer bin is reached. Therefore, the free space in the buffer bins will change over time. Since the recursion always halts at the buffer bin, the algorithm is applied on a single queue Q_k.

The following Insert/Delete operation is defined for a whole set $J = \{i_1, \ldots, i_n\}$ of items. If an item i of size interval S_ℓ has to be inserted or deleted, the algorithm is called with Insert$(\{i\}, b_{S(\ell)}, Q_k)$ respectively Delete$(\{i\}, b_x, Q_k)$, where b_x is the bin containing item i and Q_k is the queue containing bin $b_{S(\ell)}$ or b_x. Recall that $S_j = \left[\frac{\epsilon}{2^{j+1}}, \frac{\epsilon}{2^j}\right)$ is a fixed interval for every $j \geq 1$ and define $S_{\leq j}, S_{>j}$ as $S_{\leq j} = \bigcup_{i=1}^{j} S_i$ and $S_{>j} = \bigcup_{i>j} S_i$.

Figure 7a shows an example call of Insert$(\{i\}, b_x, Q_k)$. Item i with $s(i) \in S_1$ is put into the corresponding bin b_x into the size interval S_1. As b_x now contains too many items, some items from the smallest size interval S_2 (marked by the dashed lines) are put into the last bin b_{x+2} containing items from S_2. Those items in turn push items from the smallest size interval S_3 into the last bin containing items of this size and so on. This process terminates if either no items need to be shifted to the next bin or the buffer bin bb_k is reached.



(a) Insert($\{i\}$,b_x,Q_k) with $s(i) \in S_1$ **(b)** Delete($\{i\}$,b_x,Q_k) with $s(i) \in S_1$

Figure 7 Example calls of Insert and Delete.

▶ **Algorithm 2** (Insert/Delete: only small items).

- ***Insert***(J, b_x, Q_k)***:***
  - *For* $J = \{i_1, \ldots, i_n\}$*, find the smallest* ℓ *such that* $s(i_j) \in S_{\leq \ell}$ *for all* $j \geq 1$ *and insert those items into bin* b_x*. (By Lemma 12 the total size of* J *is bounded by* $\mathcal{O}(1/\epsilon)$ *times the size of the item which triggered the first Insert operation.)*
  - *Remove just as many items* $J' = \{i'_1, \ldots, i'_m\}$ *of the smaller size intervals* $S_{>\ell}$ *appearing in bin* b_x *(starting by the smallest) such that all items* J *fit into the bin* b_x*. If there are not enough items of smaller categories to insert all items from* J*, insert the remaining items from* J *into* b_{x+1}*.*
  - *Partition* J' *into* $J'_{\ell+1}, J'_{\ell+2}, \ldots$ *such that* $J_{\ell+i}$ *contains the items in the respective size interval* $S_{\ell+i}$*. Put* $J'_{\ell+i}$ *recursively into bin* $b_{S(\ell+i)}$ *(i. e., call Insert(*$J'_{\ell+i}, b_{S(\ell+i)}, Q_k$*) for each* $i \geq 1$*). If the buffer bin* bb_k *is left of* $b_{S(\ell+i)}$ *call Insert(*$J'_{\ell+i}, bb_k, Q_k$*) instead.*
- ***Delete***(J, b_x, Q_k)***:***
  - *For* $J = \{i_1, \ldots, i_n\}$*, find the smallest* ℓ *such that* $s(i_j) \in S_{\leq \ell}$ *for all* $j \geq 1$ *and remove those from bin* b_x *(By Lemma 12 the total size of* J *is bounded by* $\mathcal{O}(1/\epsilon)$ *times the size of the item which triggered the first Delete operation.)*
  - *Let* $S_{\ell'}$ *be the smallest size interval appearing in* b_x*. Insert as many items* $J' = \{i'_1, \ldots, i'_m\}$ *from* $b_{S(\ell')}$ *such that* b_x *is filled completely. If there are not enough items from the size category* $S_{\ell'}$*, choose items from the next size category* $S_{\geq \ell'+1}$ *in bin* b_{x+1}*.*
  - *Partition* J' *into* $J'_{\ell+1}, J'_{\ell+2}, \ldots$ *such that* $J_{\ell+i}$ *contains the items in the respective size interval* $S_{\ell+i}$*. Remove items* $J'_{\ell+i}$ *from bin* $b_{S(\ell+i)}$ *recursively (i. e., call Delete(*$J'_{\ell+i}, b_{S(\ell+i)}, Q_k$*) for each* $i \geq 1$*). If the buffer bin* bb_k *is left of* $b_{S(\ell+i)}$*, call Delete(*$J'_{\ell+i}, bb_k, Q_k$*) instead.*

Using the above operations, the normal bins are filled completely. However, the size of the items in the buffer bins changes. In the following we describe how to handle buffer bins that are being emptied or filled completely. If a buffer bin is filled completely, we add a new empty buffer bin and split the queue if it contains too many bins. Similarly, if a buffer bin is emptied completely, we remove it and label the last bin of the queue as buffer bin. If the queue now contains too few bins, we merge the queue with its successor. There is thus no need in introducing a new buffer bin, as one can simply use the buffer bin of the successor as the new buffer bin.

▶ **Algorithm 3** (Handle filled/emptied buffer bins).

- **Case 1: The buffer bin of Q_i is filled completely by an insert operation.**
  - *Label the filled bin as a normal bin and add a new empty buffer bin to the end of* Q_i*.*
  - *If* $|Q_i| > 2/\epsilon$*, split* Q_i *into two queues* Q'_i, Q''_i *with* $|Q''_i| = |Q'_i| + 1$*. The buffer bin of* Q''_i *is the newly added buffer bin. Add an empty buffer bin to* Q'_i *such that* $|Q'_i| = |Q''_i|$*.*
- **Case 2: The buffer bin of Q_i is being emptied due to a delete operation.**

- *Remove the now empty bin.*
- *If* $|Q_i| \geq |Q_{i+1}|$ *and* $|Q_i| > 1/\epsilon$*, choose the last bin of* Q_i *and label it as new buffer bin of* Q_i*.*
- *If* $|Q_{i+1}| > |Q_i|$ *and* $|Q_{i+1}| > 1/\epsilon$*, choose the first bin of* Q_{i+1}*, move it to* Q_i *and label it buffer bin.*
- *If* $|Q_{i+1}| = |Q_i| = 1/\epsilon$*, merge queues* Q_i *and* Q_{i+1}*. As* Q_{i+1} *already contains a buffer bin, there is no need to label another bin as buffer bin for the merged queue.*

Creating and deleting buffer bins this way guarantees that property (3) is never violated since queues never exceed the length of $2/\epsilon$ and never fall below $1/\epsilon$.

It remains to prove that the migration of the operations is bounded and that the properties are invariant under those operations.

▶ **Lemma 12.**
(i) *Let* B *be a packing that fulfills properties* (1) *to* (3)*. Applying Algorithm 2 on* B *yields a packing* B' *that also fulfills properties* (1) *to* (3)*.*
(ii) *The migration factor of a single operation is bounded by* $\mathcal{O}(1/\epsilon)$ *for all* $\epsilon \leq 2/7$*.*

4.2 Handling the General Setting

In the scenario that there are mixed item types (small and large items), we need to be more careful in the creation and the deletion of buffer bins. To maintain the approximation guarantee, we have to make sure that as long as there are bins containing only small items, the remaining free space of all bins can be bounded. Packing small items into empty bins and leaving bins with large items untouched does not lead to a good approximation, as the free space of the bins containing only large items is not used. To tackle this problem we developed new techniques and ideas using an even more refined distribution of the bins. A bin that contains no small items is called a heap bin. The heap bins are then used as buffer bins. In order to measure the number of buffer bins that are about to get filled, we developed a potential function. The potential function is related to the number of heap bins and the free space of the last buffer bin that contains large items. This relation allows us to extend Algorithm 2 to the general case of both small and large items. The developed techniques involve a complex structure of the packing and require an intricate analysis. See the full version for details.

Combining all the results from the current and the previous section, we finally obtain our central result.

▶ **Theorem 13.** *There is a AFPTAS with a migration factor of at most* $\mathcal{O}(1/\epsilon^4 \cdot \log 1/\epsilon)$ *for the fully dynamic bin packing problem.*

Acknowledgements. We would like to thank Till Tantau for his valuable comments and suggestions to improve the presentation of the paper.

References

1 J. Balogh, J. Békési, and G. Galambos. New lower bounds for certain classes of bin packing algorithms. In *Workshop on Approximation and Online Algorithms(WAOA)*, volume 6534 of *LNCS*, pages 25–36, 2010.

2 J. Balogh, J. Békési, G. Galambos, and G. Reinelt. Lower bound for the online bin packing problem with restricted repacking. *SIAM Journal on Computing*, 38(1):398–410, 2008.

3 A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010*, pages 577–578, 2010.

4 N. Bobroff, A. Kochut, and K.A. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Integrated Network Management, IM 2007. 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

5 D.J. Brown. A lower bound for on-line one-dimensional bin packing algorithms. Technical Report R-864, Coordinated Sci Lab Univ of Illinois Urbana, 1979.

6 J.W. Chan, T. Lam, and P.W.H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science*, 409(3):521–529, 2008.

7 J.W. Chan, P.W.H. Wong, and F.C.C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis. *Algorithmica*, 53(2):172–206, 2009.

8 E.G. Coffman, M.R. Garey, and D.S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12(2):227–258, 1983.

9 Khuzaima D., Shahin K., and Alejandro L. On the online fault-tolerant server consolidation problem. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'14*, pages 12–21, 2014.

10 K. Eisemann. The Trim Problem. *Management Science*, 3(3):279–284, 1957.

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

12 L. Epstein and A. Levin. Robust approximation schemes for cube packing. *SIAM Journal on Optimization*, 23(2):1310–1343, 2013.

13 W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

14 G. Gambosi, A. Postiglione, and M. Talamo. Algorithms for the relaxed online bin-packing model. *SIAM Journal on Computing*, 30(5):1532–1551, 2000.

15 Z. Ivković and E.L. Lloyd. Partially dynamic bin packing can be solved within $1+\epsilon$ in (amortized) polylogarithmic time. *Information Processing Letter*, 63(1):45–50, 1997.

16 Z. Ivković and E.L. Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM Journal on Computing*, 28(2):574–611, 1998.

17 Z. Ivković and E.L. Lloyd. Fully dynamic bin packing. In *Fundamental Problems in Computing*, pages 407–434. Springer, 2009.

18 K. Jansen and K. Klein. A robust AFPTAS for online bin packing with polynomial migration. In *International Colloquium on Automata, Languages, and Programming(ICALP)*, pages 589–600, 2013.

19 D.S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

20 D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

21 D.S. Johnson, A.J. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

22 G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *2008 International Conference on Autonomic Computing, ICAC 2008, June 2-6, 2008, Chicago, Illinois, USA*, pages 23–32, 2008.

23 G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *Middleware 2009,* *ACM/IFIP/USENIX, 10th International Middleware Conference, Proceedings*, pages 163–183, 2009.

24 N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 312–320. IEEE Computer Society, 1982.

25 C.C. Lee and D. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

26 Y. Li, X. Tang, and W. Cai. On dynamic bin packing for resource allocation in the cloud. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'14*, pages 2–11, 2014.

27 F.M. Liang. A lower bound for on-line bin packing. *Information processing letters*, 10(2):76–79, 1980.

28 J.M. Park, Uday R. Savagaonkar, E.K.P. Chong, H.J. Siegel, and S.D. Jones. Efficient resource allocation for qos channels in mf-tdma satellite systems. In *MILCOM 2000. 21st Century Military Communications Conference Proceedings*, volume 2, pages 645–649. IEEE, 2000.

29 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

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

31 M. Skutella and J. Verschae. A robust PTAS for machine covering and packing. In *European Symposium on Algorithms(ESA)*, volume 6346 of *LNCS*, pages 36–47, 2010.

32 S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 10–10, 2008.

33 A.L. Stolyar. An infinite server system with general packing constraints. *Operations Research*, 61(5):1200–1217, 2013.

34 A.L. Stolyar and Y. Zhong. A large-scale service system with packing constraints: Minimizing the number of occupied servers. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 41–52. ACM, 2013.

35 J.D. Ullman. *The Performance of a Memory Allocation Algorithm.* Technical report. Princeton University, 1971.

36 A. Verma, P. Ahuja, and A. Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *Middleware 2008, ACM/IFIP/USENIX 9th International Middleware Conference, Proceedings*, pages 243–264, 2008.

37 A. Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43(5):277–284, 1992.

38 Prudence WH Wong, Fencol CC Yung, and Mihai Burcea. An 8/3 lower bound for online dynamic bin packing. In *Algorithms and Computation*, pages 44–53. Springer, 2012.

39 A.C. Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.