Fully-Dynamic Bin Packing with Little Repacking

Björn Feldkord
Paderborn University, Paderborn, Germany

Matthias Feldotto
Paderborn University, Paderborn, Germany

Anupam Gupta[1]
Carnegie Mellon University, Pittsburgh, USA

Guru Guruganesh[2]
Carnegie Mellon University, Pittsburgh, USA

Amit Kumar[3]
IIT Delhi, New Delhi, India

Sören Riechers
Paderborn University, Paderborn, Germany

David Wajc[4]
Carnegie Mellon University, Pittsburgh, USA

"To improve is to change; to be perfect is to change often."
– Winston Churchill.

Abstract

We study the classic BIN PACKING problem in a fully-dynamic setting, where new items can arrive and old items may depart. We want algorithms with low asymptotic competitive ratio *while repacking items sparingly* between updates. Formally, each item i has a *movement cost* $c_i \geq 0$, and we want to use $\alpha \cdot OPT$ bins and incur a movement cost $\gamma \cdot c_i$, either in the worst case, or in an amortized sense, for α, γ as small as possible. We call γ the *recourse* of the algorithm. This is motivated by cloud storage applications, where fully-dynamic BIN PACKING models the problem of data backup to minimize the number of disks used, as well as communication incurred in moving file backups between disks. Since the set of files changes over time, we could recompute a solution periodically from scratch, but this would give a high number of disk rewrites, incurring a high energy cost and possible wear and tear of the disks. In this work, we present optimal tradeoffs between number of bins used and number of items repacked, as well as natural extensions of the latter measure.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Bin Packing, Fully Dynamic, Recourse, Tradeoffs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.51

Related Version This paper is a merged version of papers [15], `https://arxiv.org/abs/1711.02078`, and [11], `https://arxiv.org/abs/1711.01231`.

[1] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[2] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[3] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[4] Work done in part while the authors were at the Simons Institute for the Theory of Computing.

Acknowledgements The work of Björn Feldkord, Matthias Feldotto and Sören Riechers is supported in part by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901). The work of Anupam Gupta and Guru Guruganesh is supported in part by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790. Amit Kumar and Anupam Gupta are part of the Joint Indo-US Virtual Center for Computation and Uncertainty. The work of David Wajc is supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

1 Introduction

Consider the problem of data backup on the cloud, where multiple users' files are stored on disks (for simplicity, of equal size). This is modeled by the BIN PACKING problem, where the items are files and the bins are disks, and we want to pack the items in a minimum number of bins. However, for the backup application, files are created and deleted over time. The storage provider wants to use a small number of disks, and also keep the communication costs incurred by file transfers to a minimum. Specifically, we want bounded "recourse", i.e., items should be moved sparingly while attempting to minimize the number of bins used in the packing. These objectives are at odds with each other, and the natural question is to give optimal tradeoffs between them.

Formally, an instance $\mathcal{I}$ of the BIN PACKING problem consists of a list of n items of sizes $s_1, s_2, \ldots, s_n \in [0, 1]$. A bin packing algorithm seeks to pack the items of $\mathcal{I}$ into a small number of unit-sized bins; let $OPT(\mathcal{I})$ be the minimum number of unit-sized bins needed to pack all of $\mathcal{I}$'s items. This NP-hard problem has been studied since the 1950s, with hundreds of papers addressing its many variations; see e.g., [18, Chapter 2] and [7] for surveys. Much of this work (e.g. [27, 39, 29, 30, 31, 37, 36, 33, 16, 2]) starting with [35] studies the *online* setting, where items arrive sequentially and are packed into bins immediately and irrevocably. While the offline problem is approximable to within an *additive* term of $\mathcal{O}(\log OPT)$ [17], in the online setting there is a 1.540-*multiplicative gap* between the algorithm and OPT in the worst case, even as $OPT \to \infty$ [2]. Given the wide applicability of the online problem, researchers have studied the problem where a small amount of repacking is allowed upon item additions and deletions. Our work focuses on the bounded recourse setting, and we give *optimal tradeoffs* between the number of bins used and amount of repacking required in the fully-dynamic setting for several settings.

A *fully-dynamic* BIN PACKING *algorithm* $\mathcal{A}$, given a sequence of item insertions and deletions, maintains at every time t a feasible solution to the BIN PACKING instance $\mathcal{I}_t$ given by items inserted and not yet deleted until time t. Every item i has a size s_i and a *movement cost* c_i, which $\mathcal{A}$ pays every time $\mathcal{A}$ moves item i between bins. Different approaches have been developed to measure the amount of repacking, in the context of data backup these measures corresponds to the communication or energy cost. For example, the movement cost c_i may be proportional to the size of the file (which is the communication cost if the files are all large), or may be a constant (if the files are small, and the overhead is entirely in setting up the connections), or may depend on the file in more complicated ways.

Formally, the fully-dynamic BIN PACKING problem is as follows.

▶ **Definition 1.1.** *A fully-dynamic algorithm $\mathcal{A}$ has (i) an* asymptotic competitive ratio *α, (ii)* additive term *β and (iii)* recourse *γ, if at each time t it packs the instance $\mathcal{I}_t$ in at most $\alpha \cdot OPT(\mathcal{I}_t) + \beta$ bins, while paying at most $\gamma \cdot \sum_{i=1}^{t} c_i$ movement cost until time t. If at each time t algorithm $\mathcal{A}$ incurs at most $\gamma \cdot c_t$ movement cost, for c_t the cost of the item updated at time t, we say $\mathcal{A}$ has* worst-case *recourse γ, otherwise we say it has* amortized *recourse γ.*

Any algorithm must pay $\frac{1}{2}\sum_{i=1}^{t} c_i$ movement cost just to insert the items, so an algorithm with γ recourse spends at most 2γ times more movement cost than the absolute minimum. The goal is to design algorithms which simultaneously have low asymptotic competitive ratio (a.c.r), additive term, and recourse. There is a natural tension between the a.c.r and recourse, so we want to find the optimal a.c.r-to-recourse trade-offs.

1.1 Related Work

Based on the classical BIN PACKING Problem [14], different online and dynamic variants have been developed and investigated. Due to space constraints we focus on the scenarios which share the most important properties with our model and only mention the best results for them. In the *Online Bin Packing Problem* [35], the items are unknown to the algorithm at the beginning and appear one after another. Balogh et al. [2] show a lower bound of 1.54037 for this problem and Seiden [33] presents an approximation algorithm with a competitive ratio of 1.58889. In the *Dynamic Bin Packing* setting [28], additionally to arrivals as in the Online Bin Packing Problem departures of items are also allowed. Here, a lower bound of 8/3 by Wong et al. [38] and an upper bound of 2.897 by Coffman et al. [28] exist.

We now turn our attention to the models which allow repacking of items. In the *Relaxed Online Bin Packing Problem*, first studied by Gambosi et al. [12, 13], online arrivals and no departures occur, but repacking of items is allowed. Repacking means that items can be assigned to another bin in the course of the execution, while in a setting without repacking decisions are irrevocable. Gambosi et al. gave a 4/3-a.c.r algorithm for the insertion-only setting which moves each item $\mathcal{O}(1)$ times, which in terms of recourse translates into $\mathcal{O}(1)$ amortized recourse for general movement costs. Following Sanders et al. [32], who studied makespan minimization with recourse, Epstein et al. [9] re-introduced the dynamic BIN PACKING problem with the movement costs c_i equaling the size s_i (the *size cost* setting, where worst-case recourse is also called *migration factor*). Epstein et al. gave a solution with a.c.r $(1+\varepsilon)$ and bounded (exponential in ϵ^{-1}) recourse for the insertion-only setting. Jansen and Klein [23] improved the recourse to $poly(\varepsilon^{-1})$ for insertion-only algorithms. The best known lower bound for an algorithm which uses only constant recourse in the *unit cost* model (i.e. $c_i = 1$ for all items) is originally given for our model, but it also applies to this setting with 1.3871 by Balogh et al. [4]. From the positive perspective Balogh et al. [5] give an approximation algorithm based on the Harmonic Fit Algorithm [29] for which they achieve a competitive ratio of $3/2$ and $\mathcal{O}(\varepsilon^{-1})$ movements per update. Since our algorithms are also applicable to this setting we improve this result to also close the gap between the lower and upper bound for this problem.

Our setting is the most powerful model among the presented ones, the *Fully-Dynamic Bin Packing* [21], in which we allow arrivals and departures of items as well as repacking. Ivković and Lloyd [21, 22] introduced the model of Fully-Dynamic Bin Packing and developed an algorithm called *Mostly Myopic Packing (MMP)* which achieves a $\frac{5}{4}$-competitive ratio. Their algorithm is based on an offline algorithm by Johnson [25, 26] and utilizes a technique whereby the packing of an item is done with a total disregard for already packed items of a smaller size. In contrast to our work, they use the concept of bundling smaller elements in their analysis, i.e. a group of items smaller than ε may be moved as a single item for unit costs. They can show that the number of single items or bundles of very small elements that need to be repacked is bounded by a constant. Additionally, Ivković [19] also gives a slightly simpler version of this algorithm, called *Myopic Packing (MP)*. It uses similar ideas but ignores one step of MMP that results in a much easier analysis and a competitive ratio of $\frac{4}{3}$. Berndt et al. [6] consider exactly the same setting, also allowing the bundling of very small elements in the analysis. Their algorithm achieves a $1+\varepsilon$ approximation ratio and a bound of $\mathcal{O}\left(1/\varepsilon^4 \log\left(1/\varepsilon\right)\right)$ for the recourse in both the size costs and unit costs model.

In addition to the positive algorithmic results, researchers also explored lower bounds for this setting. All results assume that no bundling is allowed in the unit cost model (otherwise there can only be the trivial lower bound of one [6]), hence allowing only a constant number of shifted items per time step. Ivković and Lloyd [20] show a lower bound of $\frac{4}{3}$. Their construction uses the inability of an algorithm to react to insertions and deletions of items with size slightly larger than $\frac{1}{2}$ when the remaining items may be of arbitrarily small size. Balogh et al. [3, 4] improve this bound to roughly 1.3871. They extend the technique of the previous lower bound by constructing multiple lists of large items whose sizes are chosen through the construction of a linear program. Their results are the inspiration for some of the parameter choices in this work. For size costs, Berndt et al. [6] showed that any $(1+\varepsilon)$ a.c.r algorithm must have *worst-case recourse* $\Omega(\varepsilon^{-1})$. While these give nearly-tight results for "size costs" $c_i = s_i$, the unit cost ($c_i = 1$) and general cost cases were not so well understood prior to this work.

1.2 Our Results

We give (almost) tight characterizations for the recourse-to-asymptotic competitive ratio trade-off for fully-dynamic BIN PACKING under (a) unit movement costs, (b) general movement costs and (c) size movement costs. Our results are summarized in the following theorems. (See Tables 1 and 2 for a tabular listing of our results contrasted with the best previous results.) Note that an amortized recourse bound is a weaker (resp. stronger) upper (resp. lower) bound. In the context of sensitivity analysis (see [32]), our bounds provide a tight characterization of the *average* change between *approximately*-optimal solutions of slightly modified instances.

Unit Costs. Consider the most natural movement costs, *unit costs*, where $c_i = 1$ for all items i. Here we give tight upper and lower bounds. Let $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.3871$ (here W_{-1} is the lower real branch of the Lambert W-function [8]). Balogh et al. [4] showed that α is a lower bound on the a.c.r with constant recourse. We present an alternative and simpler proof of this lower bound, also giving tighter bounds: doing better than α requires either *polynomial* additive term or recourse. Moreover, we give two matching algorithms proving α is tight for this problem: The first one uses directly the insights from the lower bound while the second one drives deeper into the structural insights of the current problem instance and reaches a slightly better result.

▶ **Theorem 1.2** (Unit Costs Tight Bounds). *For any $\varepsilon > 0$, there exists fully-dynamic* BIN PACKING *algorithms with amortized competitive ratio $(\alpha+\varepsilon)$ with worst case recourse $\mathcal{O}(\varepsilon^{-2})$ under unit movement costs and additive term $\mathcal{O}(\varepsilon^{-1})$. Conversely, any algorithm with a.c.r $(\alpha-\varepsilon)$ has additive term and amortized recourse whose product is $\Omega(\varepsilon^4 \cdot n)$ under unit movement costs.*

General Costs. Next, we consider the most general problem, with arbitrary movement costs. Theorem 1.2 showed that in the unit cost model, we can get a better a.c.r than for online BIN PACKING without repacking, whose optimal a.c.r is at least 1.540 ([2]). Alas, the fully-dynamic BIN PACKING problem with the general costs is not easier than the arrival-only online problem (with no repacking).

Table 1 Fully-dynamic bin packing with limited recourse: Positive results (Big-$\mathcal{O}$ notation dropped for notational simplicity)

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
Unit	$1.5+\varepsilon$	ϵ^{-1}	ϵ^{-1}	✓	insertions only	Balogh et al. [5]
	$\alpha+\varepsilon$	ϵ^{-1}	ε^{-2}	✓	$\alpha \approx 1.387$	**Theorem 2.11**
General	1.589	1	1	✗		**Theorem 3.5**
	1.333	1	1	✗	insertions only	Gambosi et al. [13]
Size	$1+\varepsilon$	1	$\varepsilon^{-O(\epsilon^{-2})}$	✓	insertions only	Epstein & Levin [10]
	$1+\varepsilon$	ϵ^{-2}	ϵ^{-4}	✓	insertions only	Jansen & Klein [23]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-3}\log(\varepsilon^{-1})$	✓	insertions only	Berndt et al. [6]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-4}\log(\varepsilon^{-1})$	✓		Berndt et al. [6]
	$1+\varepsilon$	ϵ^{-1}	ϵ^{-2}	✗		**Fact 4.1**

Table 2 Fully-dynamic bin packing with limited recourse: Negative results.

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
Unit	1.333	$o(n)$	1	✓		Ivković & Lloyd [20]
	$\alpha-\varepsilon$	$o(n)$	1	✓	$\alpha \approx 1.387$	Balogh et al. [4]
	$\alpha-\varepsilon$	$o(\epsilon^2 \cdot n^\delta)$	$\Omega(\varepsilon^2 \cdot n^{1-\delta})$	✗	for all $\delta \in (0,1/2]$	**Theorem 2.1**
General	1.540	$o(n)$	∞	✗	as hard as online	**Theorem 3.1**
Size	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✓		Berndt et al. [6]
	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✗		**Theorem 4.2**

▶ **Theorem 1.3** (Fully Dynamic as Hard as Online)**.** *Any fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs has a.c.r at least as high as that of any online* BIN PACKING *algorithm. Given current bounds ([2]), this is at least* 1.540*.*

Given this result, is it conceivable that the fully-dynamic model is *harder* than the arrival-only online model, even allowing for recourse? We show this is likely not the case, as we can almost match the a.c.r of the current-best algorithm for online BIN PACKING.

▶ **Theorem 1.4** (Fully Dynamic Nearly as Easy as Online)**.** *Any algorithm in the Super Harmonic family of algorithms can be implemented in the fully-dynamic setting with constant recourse under general movement costs. This implies an algorithm with a.c.r* 1.589 *using [33].*

The current best online BIN PACKING algorithm [1] is not from the Super Harmonic family but is closely related to it. It has an a.c.r of 1.578, so our results for fully-dynamic BIN PACKING are within a hair's width of the best bounds known for online BIN PACKING. It remains an open question as to whether our techniques can be extended to achieve the improved a.c.r bounds while maintaining constant recourse. While we are not able to give a black-box reduction from fully-dynamic algorithms to online algorithms, we conjecture that such a black-box reduction exists and that these problems' a.c.r are equal.

Size Costs. Finally, we give an extension of the already strong results known for the size cost model (where $c_i = s_i$ for every item i). We show that the lower bound known in the worst-case recourse model extends to the amortized model as well, for which it is easily shown to be tight.

▶ **Theorem 1.5** (Size Costs Tight Bounds). *For any $\varepsilon > 0$, there exists a $(1+\varepsilon)$-a.c.r algorithm with $\mathcal{O}(\varepsilon^{-2})$ additive term and $\mathcal{O}(\varepsilon^{-1})$ amortized recourse under size costs. Conversely, for infinitely many $\varepsilon > 0$, any $(1+\varepsilon)$-a.c.r algorithm with $o(n)$ additive term requires $\Omega(\varepsilon^{-1})$ amortized recourse under size costs.*

The hardness result above was previously only known for *worst-case* recourse [6]; this previous lower bound consists of a hard instance which effectively disallowed any recourse, while lower bounds against amortized recourse can of course not do so.

1.3 Techniques and Approaches

Unit Costs. For unit costs, our lower bound is based on a natural instance consisting of small items, to which large items of various sizes (greater than $\frac{1}{2}$) are repeatedly added/removed [4]. The near-optimal solutions oscillate between either having most bins containing both large and small items, or most bins containing only small items. Since small items can be moved rarely, this effectively makes them static, giving us hard instances. We can optimize for the lower bound arising thus via a *gap-revealing* LP, similar to [4]. However, rather than bound this LP precisely, we exhibit a near-optimal dual solution showing a lower bound of $\alpha - \varepsilon$.

For the first upper bound, the same LP now changes from a gap-revealing LP to a *factor-revealing* LP. The LP solution shows how the small items should be packed in order to "prepare" for arrival of large items, to ensure $(\alpha + \epsilon)$-competitiveness. An important building block of our algorithms is the ability to deal with (sub)instances made up solely of small items. In particular, we give fully-dynamic BIN PACKING algorithms with a.c.r $(1+\varepsilon)$ and only $\mathcal{O}(\varepsilon^{-2})$ *worst-case* recourse if all items have size $\mathcal{O}(\varepsilon)$. The ideas we develop are versatile, and are used, e.g., to handle the small items for general costs – we discuss them in more detail below. We then extend these ideas to allow for (approximately) packing items according to a "target curve", which in our case is the solution to the above LP. At this point the LP makes a re-appearance, allowing us to analyze a simple packing of large items "on top" of the curve (which we can either maintain dynamically using the algorithm of Berndt et al. [6] or recompute periodically in linear time); in particular, using this LP we are able to prove this packing of large items on top of small items has an optimal a.c.r of $\alpha + \mathcal{O}(\varepsilon)$. This implies that lazily repacking the large items near-optimally and packing them on top of the small items easily yields $\mathcal{O}(\varepsilon^{-2})$ amortized recourse. Relying on the fully-dynamic $poly(\varepsilon^{-1})$ migration factor AFPTAS algorithms of [6] with some additional ideas allows us to leverage this LP to obtain worst-case recourse bounds.

In the second upper bound, we take a different, more recourse-efficient method to pack the large items, which results in improved worst-case recourse. Specifically, we utilize an algorithm called Myopic Packing (MP) by Ivković [19]. This algorithm has a competitive ratio of $^4/_3$ (it is below the lower bound for our model since it uses bundling, i.e., it allows the repacking of a group of small items as one) and modifies only a constant number of bins per time step. Applying this algorithm to only items of a size of at least $^\varepsilon/_{15}$ restricts the amount of repacking to $\mathcal{O}(^1/_\varepsilon)$ per time step. We develop a new view of this algorithm to derive structural properties of the solution which are needed for the combination of major and minor items. Small items are handled similarly to the previous algorithm, however, we provide a fixed solution to the LP from before in order to use it in the analysis of the combined solution. Finally, the bins with small items are merged with the bins with large items by utilizing a greedy-like approach, where small chunks of reserved space and a big cumulative size of major items is prioritized in order to guarantee an efficient utilization of the reserved space. The combination has two main challenges: Firstly, we ensure that this greedy process only has to modify $\mathcal{O}(1/\varepsilon)$ bins per time step. Secondly, we guarantee a space-efficient combination resulting in an overall good solution quality. The analysis carefully utilizes the structural insights about the solution for major items to estimate the solution quality within the bins that did not get merged.

General Costs. To transform any online lower bound to a lower bound for general costs even with recourse, we give the same sequence of items but with movement costs that drop sharply. This prohibits worst-case recourse, but since we allow amortized recourse, the large costs c_i for early items can still be used to pay for movement of the later items (with smaller c_i values). Hence, if the online algorithm moves an element e to some bin j, we delete all the elements after e and proceed with the lower bound based on assigning $e \mapsto j$. Now a careful analysis shows that for some sub-sequence of items the dynamic algorithm effectively performs no changes, making it a recourse-less online algorithm – yielding the claimed bound.

For the upper bound, we show how to emulate any algorithm in the *Super Harmonic* class of algorithms [33] even in the fully-dynamic case. We observe that the analysis for SH essentially relies on maintaining a stable matching in an appropriate compatibility graph. Now our algorithm for large items uses a subroutine similar to the Gale-Shapley algorithm. Our simulation also requires a solution for packing similarly-sized items. The idea here is to sort items by their cost (after rounding costs to powers of two), such that any insertion or deletion of items of this size can be "fixed" by moving at most one item of each smaller cost, yielding $\mathcal{O}(1)$ worst-case recourse.

The final ingredient of our algorithm is packing of small items into $(1-\epsilon)$-full bins. Unlike the online algorithm which can just use FirstFit, we have to extend the ideas of Berndt et al. [6] for the size-cost case. Namely we group bins into buckets of $\Theta(1/\epsilon)$ bins, such that all but one bin in a bucket are $1-\mathcal{O}(\epsilon)$ full, guaranteeing these bins are $1-\mathcal{O}(\epsilon)$ full on average. While for the size-cost case, bucketing bins and sorting the small items by size readily yields an $\mathcal{O}(\epsilon^{-1})$ recourse bound, this is more intricate for general case where size and cost are not commensurate. Indeed, we also maintain the small items in sorted order according to their size-to-movement-cost ratio (i.e., their *Smith ratio*), and only move items to/from a bin once it has $\Omega(\epsilon)$'s worth of volume removed/inserted (keeping track of erased, or "ghost" items). This gives an amortized $\mathcal{O}(\epsilon^{-2})$ recourse cost for the small items.

Size Costs. The technical challenge for size costs is in proving the lower bound with *amortized* recourse that matches the easy upper bound. Our proof uses item sizes which are roughly the reciprocals of the Sylvester sequence [34]. While this sequence has been used often in the context of BIN PACKING algorithms, our sharper lower bound explicitly relies on its divisibility properties (which have not been used in similar contexts, to the best of our knowledge). We show that for these sizes, any algorithm $\mathcal{A}$ with a.c.r $(1+\epsilon)$ must, on an instance containing N items of each size, have $\Omega(N)$ many bins with one item of each size. In contrast, on an instance containing N items of each size *except the smallest size* ε, algorithm $\mathcal{A}$ must have some (smaller) $\mathcal{O}(N)$ many bins with more than one distinct size in them. Repeatedly adding and removing the N items of size ε, the algorithm must suffer a high amortized recourse.

We give a more in-depth description of our algorithms and lower bounds for unit, general and size costs, highlighting some salient points of their proofs in §2,§3 and §4, respectively. Due to space constraints we defer most proofs to the full versions of this paper ([15, 11]).

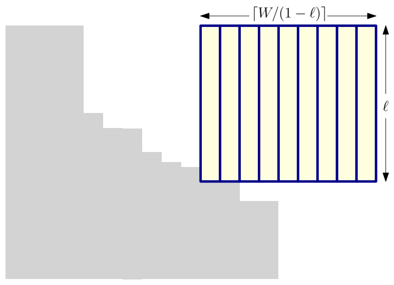

Figure 1 A packing of instance $\mathcal{I}_\ell$. The ℓ-sized items (in yellow) are packed "on top" of the small items (in grey).

2 Unit Movement Costs

We consider the natural unit movement costs model. First, we show that an a.c.r better than $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.387$ implies either polynomial additive term or recourse. Next, we give tight $(\alpha + \epsilon)$-a.c.r algorithms, with both additive term and recourse polynomial in ϵ^{-1}. Our key idea is to use an LP that acts both as a *gap-revealing LP* for a lower bound, and also as a *factor-revealing LP* (as well as an inspiration) for our algorithm.

2.1 Impossibility results

Alternating between two instances, where both have $2n - 4$ items of size $1/n$ and one instance also has 4 items of size $1/2 + 1/2n$, we get that any $\left(\frac{3}{2} - \varepsilon\right)$-competitive online BIN PACKING algorithm must have $\Omega(n)$ recourse. However, this only rules out algorithms with a *zero* additive term. Balogh et al. [4] showed that any $\mathcal{O}(1)$-recourse algorithm (under unit movement costs) with $o(n)$ additive term must have a.c.r at least $\alpha \approx 1.387$. We strengthen both this impossibility result by showing the need for a large additive term or recourse to achieve any a.c.r below α. Specifically, arbitrarily small polynomial additive terms imply near-linear recourse. Our proof is shorter and simpler than in [4]. As an added benefit, the LP we use to bound the competitive ratio will inspire our algorithm in the next section.

▶ **Theorem 2.1** (Unit Costs: Lower Bound)**.** *For any $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, for any algorithm $\mathcal{A}$ with a.c.r $(\alpha - \varepsilon)$ with additive term $o(\varepsilon \cdot n^\delta)$, there exists an dynamic bin packing input with n items on which $\mathcal{A}$ uses recourse at least $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement cost.*

The Instances: The set of instances is the a natural one, and was also considered by [4]. Let $1/2 > \delta > 0$, $c = 1/\delta - 1$, and let $W \geq 1/\varepsilon$ be a large integer. Our hard instances consist of *small* items of size $1/W^c$, and *large* items of size ℓ for all sizes $\ell \in \mathcal{S}_\varepsilon \triangleq \{\ell = 1/2 + i \cdot \varepsilon \mid i \in \mathbb{N}_{>0}, \ell \leq 1/\alpha\}$. Specifically, input $\mathcal{I}_s$ consists of $\lfloor W^{c+1} \rfloor$ small items, and for each $\ell \in \mathcal{S}_\varepsilon$, the input $\mathcal{I}_\ell$ consists of $\lfloor W^{c+1} \rfloor$ small items followed by $\lfloor \frac{W}{1-\ell} \rfloor$ size-ℓ items. The optimal bin packings for $\mathcal{I}_s$ and $\mathcal{I}_\ell$ require precisely $OPT(\mathcal{I}_s) = W$ and $OPT(\mathcal{I}_\ell) = \lceil \frac{W}{1-\ell} \rceil$ bins respectively. Consider any fully-dynamic bin packing algorithm $\mathcal{A}$ with limited recourse and bounded additive term. When faced with input $\mathcal{I}_s$, algorithm $\mathcal{A}$ needs to distribute the small items in the available bins almost uniformly. And if this input is extended to $\mathcal{I}_\ell$ for some $\ell \in \mathcal{S}_\varepsilon$, algorithm $\mathcal{A}$ needs to move many small items to accommodate these large items (or else use many new bins). Since $\mathcal{A}$ does not know the value of t beforehand, it cannot "prepare" simultaneously for all large sizes $\ell \in \mathcal{S}_\varepsilon$.

As a warm-up we show that the linear program (LP_ε) below gives a lower bound on the *absolute* competitive ratio α_ε of any algorithm $\mathcal{A}$ with no recourse. Indeed, instantiate the variables as follows. On input $\mathcal{I}_s$, let N_x be the number of bins in which $\mathcal{A}$ keeps free space in the range $[x, x+\varepsilon)$ for each $x \in \{0\} \cup \mathcal{S}_\varepsilon$. Hence the total volume packed is at most $N_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x) \cdot N_x$. This must be at least $Vol(\mathcal{I}_s) \geq W - 1/W^c$, implying constraint ($\mathrm{Vol}_\varepsilon$). Moreover, as $OPT(\mathcal{I}_s) = W$, the α_ε-competitiveness of $\mathcal{A}$ implies constraint ($\mathrm{small}_\varepsilon$). Now if instance $\mathcal{I}_s$ is extended to $\mathcal{I}_\ell$, since $\mathcal{A}$ moves no items, these ℓ-sized items are placed either in bins counted by N_x for $x \geq \ell$ or in new bins. Since $\mathcal{A}$ is α_ε-competitive and $OPT(\mathcal{I}_\ell) \leq \lceil \frac{W}{1-\ell} \rceil$ we get constraint (CR_ε). Hence the optimal value of (LP_ε) is a valid lower bound on the competitive ratio α_ε.

$$\begin{array}{llllr}
\text{minimize } \alpha_\varepsilon & & & & (\mathrm{LP}_\varepsilon) \\
\text{s.t.} & N_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x) \cdot N_x & \geq W - 1/W^c & & (\mathrm{Vol}_\varepsilon) \\
& N_0 + \sum_{x \in \mathcal{S}_\varepsilon} N_x & \leq \alpha_\varepsilon \cdot W & & (\mathrm{small}_\varepsilon) \\
& N_0 + \sum_{x \in \mathcal{S}_\varepsilon, x \leq \ell - \varepsilon} N_x + \left\lceil \frac{W}{1-\ell} \right\rceil & \leq \alpha_\varepsilon \cdot \left\lceil \frac{W}{1-\ell} \right\rceil & \forall \ell \in \mathcal{S}_\varepsilon & (\mathrm{CR}_\varepsilon) \\
& N_x \geq 0 & & &
\end{array}$$

The claimed lower bound on the a.c.r of recourse-less algorithms follows from Lemma 2.2.

▶ **Lemma 2.2.** *The optimal value $\alpha^\star_\varepsilon$ of* (LP_ε) *satisfies $\alpha^\star_\varepsilon \in [\alpha - \mathcal{O}(\varepsilon), \alpha + \mathcal{O}(\varepsilon)]$.*

To extend the above argument to the fully-dynamic case, we observe that any solution to (LP_ε) defined by packing of $\mathcal{I}_s$ as above must satisfy some constraint (CR_ε) for some $\ell \in \mathcal{S}_\varepsilon$ with equality, implying a competitive ratio of at least α_ϵ. Now, to beat this bound, a fully-dynamic algorithm must move at least ϵ volume of small items from bins which originally had less than $x - \epsilon$ free space. As small items have size $1/W^c = 1/n^\delta$, this implies that $\Omega(\epsilon n^\delta)$ small items must be moved for every bin affected by such movement. This argument yields Lemma 2.3, which together with Lemma 2.2 implies Theorem 2.1.

▶ **Lemma 2.3.** *For all $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\alpha^\star_\varepsilon$ is the optimal value of* (LP_ε)*, then any fully-dynamic* BIN PACKING *algorithm $\mathcal{A}$ with a.c.r $\alpha^\star_\varepsilon - \varepsilon$ and additive term $o(\varepsilon^2 \cdot n^\delta)$ has recourse $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement costs.*

2.2 Matching Algorithmic Results

As mentioned earlier, LP_ε also guides our algorithm. For the rest of this section, items smaller than ε are called *small*, and the rest are *large*. Items of size $s_i > 1/2$ are *huge*.

To begin, imagine a total of W volume of small items come first, followed by large items. Inspired by the LP analysis above, we pack the small items such that an N_ℓ/W fraction of bins have ℓ free space for all $\ell \in \{0\} \cup \mathcal{S}_\varepsilon$, where the N_ℓ values are near-optimal for (LP_ε). We call the space profile used by the small items the "curve"; see Figure 1. In the LP analysis above, we showed that this packing can be extended to a packing with a.c.r $\alpha + \mathcal{O}(\varepsilon)$ if $W/(1-\ell)$ items of size ℓ are added. But what if large items of *different* sizes are inserted and deleted? In §2.2.1 we show that this approach retains its $(\alpha + \mathcal{O}(\epsilon))$-a.c.r in this case too, and outline a linear-time algorithm to obtain such a packing.

The next challenge is that the small items may also be inserted and deleted. In §2.2.2 we show how to dynamically pack the small items with bounded recourse, so that the number of bins with any amount of free space $f \in \mathcal{S}_\varepsilon$ induce a near-optimal solution to LP_ε.

Finally, in §2.2.3 we combine the two ideas together to obtain our fully-dynamic algorithm.

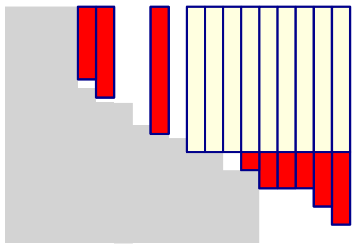

Figure 2 A packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey). Parts of large items not in the instance $\mathcal{I}_\ell^k$ are indicated in red.

2.2.1 LP$_\varepsilon$ as a Factor-Revealing LP

In this section we show how we use the linear program LP$_\varepsilon$ to analyze and guide the following algorithm: pack the small items according to a near-optimal solution to LP$_\varepsilon$, and pack the large items near-optimally "on top" of the small items.

To analyze this approach, we first show it yields a good packing if all large items are huge and have some common size $\ell > 1/2$. Consider an instance $\mathcal{I}_s$ consisting solely of small items with total volume W, packed using N_x bins with gaps in the range $[x, x+\varepsilon)$ for all $x \in \{0\} \cup \mathcal{S}_\varepsilon$, where $\{\alpha_\varepsilon, N_0, N_x : x \in \mathcal{S}_\varepsilon\}$ form a feasible solution for (LP$_\varepsilon$). We say such a packing is *α_ε-feasible*. By the LP's definition, any α_ε-feasible packing of small items can be extended to an α_ε-competitive packing for any extension $\mathcal{I}_\ell$ of $\mathcal{I}_s$ with $\ell \in \mathcal{S}_\varepsilon$, by packing the size ℓ items in the bins counted by N_x for $x \geq \ell$ before using new bins. In fact, this solution satisfies a similar property for any extension $\mathcal{I}_\ell^k$ obtained from $\mathcal{I}_s$ by adding *any* number k of items all of size ℓ. (Note that $\mathcal{I}_\ell$ is the special case of $\mathcal{I}_\ell^k$ with $k = \lfloor W/(1-\ell) \rfloor$.)

▶ **Lemma 2.4** (Huge Items of Same Size). *Any α_ε-feasible packing of small items of $\mathcal{I}_s$ induces an α_ε-competitive packing for all extensions $\mathcal{I}_\ell^k$ of $\mathcal{I}_s$ with $\ell > 1/2$ and $k \in \mathbb{N}$.*

Now, to pack the large items of the instance $\mathcal{I}$, we first create a similar new instance $\mathcal{I}'$ whose large items are all huge items. To do so, we first need the following observation.

▶ **Observation 2.5.** *For any input $\mathcal{I}$ made up of solely large items and function $f(\cdot)$, a packing of $\mathcal{I}$ using at most $(1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins has all but at most $2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 3$ of its bins containing either no large items or being more than half-filled by large items.*

Now, consider a packing $\mathcal{P}$ of the large items of $\mathcal{I}$ using at most $(1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins. By Observation 2.5, at most $2\varepsilon \cdot OPT(\mathcal{I}) + \mathcal{O}(f(\varepsilon^{-1}))$ bins in $\mathcal{P}$ are at most half full. We use these bins when packing $\mathcal{I}$. For each of the remaining bins of $\mathcal{P}$ we "glue" all large items occupying the same bin into a single item, yielding a new instance $\mathcal{I}'$ with only huge items, with any packing of $\mathcal{I}'$ "on top" of the small items of $\mathcal{I}$ trivially inducing a similar packing of $\mathcal{I}$ with the same number of bins. We pack the huge items of $\mathcal{I}'$ on top of the curve greedily, repeatedly packing the smallest such item in the fullest bin which can accommodate it. See §2.3.3 for a description of an implementation of this idea with limited recourse. Now, if we imagine we remove and decrease the size of some of these huge items, this results in a new (easier) instance of the form $\mathcal{I}_\ell^k$ for some k and $\ell > 1/2$, packed in no more bins than $\mathcal{I}'$ (see Figure 2). By Lemma 2.4, our packing of $\mathcal{I}'$ (and of $\mathcal{I}$) uses at most $\alpha_\varepsilon \cdot OPT(\mathcal{I}_\ell^k) \leq \alpha_\varepsilon \cdot OPT(\mathcal{I}') \leq (\alpha_\varepsilon + \mathcal{O}(\epsilon)) \cdot OPT(\mathcal{I}) + \mathcal{O}(f(\varepsilon^{-1}))$ bins. Performing the same while only near-optimally packing large items of size exceeding $1/4$, and packing the large items of size in the range $(\varepsilon, 1/4]$ so that we only open a new bin if all bins are at least $3/4$ full (in which case we would obtain a $4/3 < \alpha$ a.c.r) allows us to obtain even better recourse bounds. These ideas underlie the following theorem (a more complete proof of which appears in [15]).

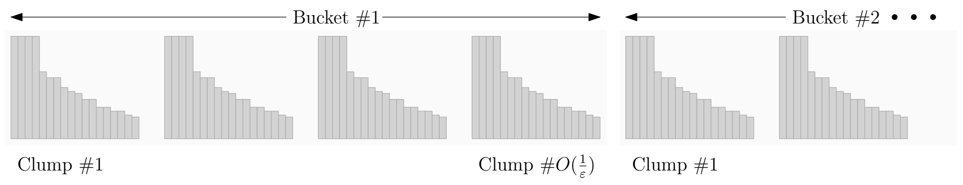

Figure 3 Buckets have $\mathcal{O}(\varepsilon^{-1})$ clumps, clumps have T bins.

▶ **Theorem 2.6.** *An α_ε-feasible packing of the small items of an instance $\mathcal{I}$ can be extended into a packing of $\mathcal{I}$ using at most $(\alpha_\epsilon + \mathcal{O}(\varepsilon)) \cdot OPT(\mathcal{I}) + \mathcal{O}(\varepsilon^{-2})$ bins in linear time. Moreover, given a fully-dynamic $(1+\varepsilon) \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ packing of items of size greater than $1/4$ of $\mathcal{I}_t$ and an α_ε-feasible packing of its small items, one can maintain a packing using $(\alpha_\epsilon + \mathcal{O}(\varepsilon)) \cdot OPT(\mathcal{I}_t) + \mathcal{O}(\max\{\varepsilon^{-2}, f(\varepsilon^{-1})\})$ bins with $\mathcal{O}(\varepsilon^{-1})$ worst-case recourse per item move in the near-optimal dynamic packing of the items of size exceeding $1/4$.*

It remains to address the issue of maintaining an α_ε-feasible packing of small items dynamically using limited recourse.

2.2.2 Dealing With Small Items: "Fitting a Curve"

We now consider the problem of packing ε-small items according to an approximately-optimal solution of (LP$_\varepsilon$). We abstract the problem thus.

▶ **Definition 2.7** (Bin curve-fitting). *Given a list of bin sizes $0 \le b_0 \le b_1 \le \ldots, b_K \le 1$ and relative frequencies $f_0, f_1, f_2, \ldots, f_K$, such that $f_x \ge 0$ and $\sum_{x=0}^{K} f_x = 1$, an algorithm for the* bin curve-fitting problem *must pack a set of m of items with sizes $s_1, \ldots, s_m \le 1$ into a minimal number of bins N such that for every $x \in [0, K]$ the number of bins of size b_x that are used by this packing lie in $\{\lfloor N \cdot f_x \rfloor, \lceil N \cdot f_x \rceil\}$.*

If we have $K = 0$ with $b_0 = 1$ and $f_0 = 1$, we get standard BIN PACKING. We want to solve the problem only for (most of the) small items, in the fully-dynamic setting. We consider the special case with relative frequencies f_x being multiples of $1/T$, for $T \in \mathbb{Z}$; e.g., $T = \mathcal{O}(\varepsilon^{-1})$. Our approach is inspired by the algorithm of [23], and maintains bins in increasing sorted order of item sizes. The number of bins is always an integer product of T. Consecutive bins are aggregated into *clumps* of exactly T bins each, and clumps aggregated into $\Theta(\varepsilon^{-1})$ *buckets* each. Formally, each clump has T bins, with $f_x \cdot T \in \mathbb{N}$ bins of size b_x for $x = 0, \ldots, K$. The bins in a clump are ordered according to their capacity b_x, so each clump looks like its target curve. Each bucket except the last consists of some $s \in [1/\varepsilon, 3/\varepsilon]$ consecutive clumps (the last bucket may have fewer than $1/\epsilon$ clumps). See Figure 3. For each bucket, all bins except those in the last clump are full to within an additive ε.

Inserting an item adds it to the correct bin according to its size. If the bin size becomes larger than the target size for the bin, the largest item overflows into the next bin, and so on. Clearly this maintains the invariant that we are within an additive ε of the target size. We perform $\mathcal{O}(T/\epsilon)$ moves in the same bucket; if we overflow from the last bin in the last clump of the bucket, we add a new clump of T new bins to this bucket, etc. If a bucket contains too many clumps, it splits into two buckets, at no movement cost. An analogous (reverse) process happens for deletes. Loosely, the process maintains that on average the bins are full to within $\mathcal{O}(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $\mathcal{O}(\varepsilon)$ fraction of bins have no guarantees whatsoever.

We now relate this process to the value of LP$_\varepsilon$. We first show that setting $T = \mathcal{O}(\varepsilon^{-1})$ and restricting to frequencies which are multiples of ε does not hurt us. Indeed, for us, $b_0 = 1$, and $b_x = (1 - x)$ for $x \in \mathcal{S}_\varepsilon$. Since (LP$_\varepsilon$) depends on the total volume W of small items, and f_x may change if W changes, it is convenient to work with a normalized version normalized by W, which is essentially (LP$_\varepsilon$) with $W = 1$. Now $n_x = N_x/W$ can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$ to the fraction of bins of size b_x in our solution. However, we also need each f_x to be an integer multiple of $1/T$ for some integer $T = \mathcal{O}(\varepsilon^{-1})$. We achieve this by slightly modifying the LP solution, obtaining the following.

▶ **Lemma 2.8** (Multiples of ε). *For any optimal solution $\{n_x\}$ to (LP$_\varepsilon$) (with $W = 1$) with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + \mathcal{O}(\varepsilon)$.*

Using a near-optimal solution to (LP$_\varepsilon$) as in the above lemma, we obtain a bin curve-fitting instance with $T = \mathcal{O}(\varepsilon^{-1})$. Noting that if we ignore the last bucket's $\mathcal{O}(T/\varepsilon^{-1}) = \mathcal{O}(\varepsilon^{-2})$ bins in our solution, we obtain an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing (with additive term $\mathcal{O}(\varepsilon^{-2})$) of the remaining items by our algorithm above, using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. We obtain the following.

▶ **Lemma 2.9** (Small Items Follow the LP). *Let $\varepsilon \leq 1/6$. Using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $\mathcal{O}(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing.*

2.2.3 Our Algorithm

From Lemma 2.8 and Lemma 2.9, we can maintain an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing of the small items (that is, a packing of inducing a solution to (LP$_\varepsilon$) with objective value $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$), all while using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. From Theorem 2.6, using a $(1+\varepsilon)$-a.c.r packing of the large items one can extend such a packing of the small items into an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-approximate packing for $\mathcal{I}_t$, where $\alpha^\star_\varepsilon \leq \alpha + \mathcal{O}(\varepsilon)$, by Lemma 2.2. It remains to address the recourse incurred by extending this packing to also pack the large items.

Amortized Recourse. Here we periodically recompute in linear time the extension guaranteed by Theorem 2.6. Dividing the time into epochs and lazily addressing updates to large items (doing nothing on deletion and opening new bins on insertion) for $\varepsilon \cdot N$ steps, where N is the number of bins we use at the epoch's start, guarantees a $(\alpha + \mathcal{O}(\varepsilon))$-a.c.r throughout the epoch. Finally, as the number of large items at the end of an epoch of length $\epsilon \cdot N$ is at most $\mathcal{O}(N/\epsilon)$, repacking them incurs $\mathcal{O}(N/\varepsilon)/(\epsilon \cdot N) = \mathcal{O}(\epsilon^{-2})$ amortized recourse.

Worst Case Recourse. To obtain worst-case recourse we rely on the fully-dynamic $(1+\varepsilon)$-a.c.r algorithm of Berndt al. [6] to maintain a $(1+\varepsilon)$-approximate packing of the sub-instance made of items of size exceeding $1/4$ using $\tilde{O}(\varepsilon^{-3})$ size cost recourse, and so at most $\tilde{O}(\varepsilon^{-3})$ item moves (as these items all have size $\Theta(1)$). By Theorem 2.6, our $(\alpha + \mathcal{O}(\varepsilon))$-feasible solution for the small items of $\mathcal{I}_t$ can be extended dynamically to an $(\alpha + (\mathcal{O}(\varepsilon)))$-a.c.r packing of all of $\mathcal{I}_t$, using $\mathcal{O}(\varepsilon^{-1})$ worst-case recourse per item change in the dynamic packing of the items of size exceeding $1/4$ of $\mathcal{I}_t$, yielding an $\tilde{O}(\varepsilon^{-4})$ worst-case recourse bound overall.

From the above discussions we obtain this section's main result – tight algorithms for unit costs with polynomial additive term and recourse.

▶ **Theorem 2.10** (Unit Costs: Upper Bound). *There is a polytime fully-dynamic* BIN PACKING *algorithm which achieves $\alpha + \mathcal{O}(\varepsilon)$ a.c.r, additive term $\mathcal{O}(\varepsilon^{-2})$ and $\mathcal{O}(\varepsilon^{-2})$ amortized recourse, or additive term poly(ε^{-1}) and $\tilde{O}(\varepsilon^{-4})$ worst-case recourse, under unit movement costs.*

2.3 An Algorithm with Improved Worst-Case Recourse

In this section, we present a second algorithm for the Unit Costs model, which uses only $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. The previous algorithm mainly used the lower bound as borders, and then applied two different algorithms for small and large items, combining the two in a greedy fashion. A near-optimal packing of the large items "on top" of the small items yields the claimed competitive ratio. In this section, we show that a simple fully-dynamic $4/3$-competitive algorithm of Ivković [19] which has recourse only $\mathcal{O}(\varepsilon^{-1})$ for the large items (and only modifies a constant number of bins) combined in the same way yields the same $\alpha + \varepsilon$ a.c.r, but with better worst-case recourse.

For the remainder of the section, we refer to items of size at most $\delta = \varepsilon/15$ as *minor* items and items larger than this size as *major* items. In §2.3.1 we outline our packing of the minor items, which differs slightly from that of §2.2.2 (in particular it only incurs an $\mathcal{O}(\varepsilon^{-1})$, rather than $\mathcal{O}(\varepsilon^{-2})$, additive term). In §2.3.2 we outline Ivković's algorithm and some of its structural properties we rely on for our analysis. Finally, in §2.3.3 we prove that the combination of these solutions in a greedy manner as discussed in §2.2.1 yields an a.c.r of $\alpha + \mathcal{O}(\varepsilon)$. The full proofs of this section are deferred to [11].

The result of this section is summarized in the following theorem:

▶ **Theorem 2.11** (Unit Cost: Improved Upper Bound). *For each $\varepsilon \in (0,1)$, there exists an algorithm for the Fully-Dynamic* BIN PACKING *Problem with a.c.r. $(1+\varepsilon)\cdot\alpha$, additive term $\mathcal{O}(\varepsilon^{-1})$ and $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse, under unit movement costs.*

2.3.1 Prospective Packing of Minor Items

For our improved recourse bound, our packing of small items here is essentially identical to that of 2.2.2, so we only highlight the differences here. As before, we pack the small items in bins which are sorted by item sizes, with bins partitioned into buckets of $\mathcal{O}(\varepsilon^{-1})$ clumps of size $\mathcal{O}(\varepsilon^{-1})$, with a prescribed height for each bin in the clump, and one "buffer" clump in each bucket which is potentially empty (with none of its bins exceeding its target filling height). Now, unlike the solution of 2.2.2, the number of bins of each target height will be the same in each clump, which will imply that the fraction of bins of type at most some j (i.e., filling height at most some w_j) is at most some $\mathcal{O}(\varepsilon)$ times the value assigned to it by the LP solution. This is sufficient to obtain the required competitive ratio for instances comprised solely of small items. The advantage of each clump having the same number of bins of each type however has the added benefit that it implies that once a particular clump is "full", (has all its bin of height at least its target height minus $\varepsilon/15$), we can begin packing major items "on top" of the obtained "curve". In particular this implies an additive term of $\mathcal{O}(\varepsilon^{-1})$, rather than $\mathcal{O}(\varepsilon^{-2})$. As before, the worst-case recourse is at most $\mathcal{O}(\varepsilon^{-2})$.

Choice of Parameters

What remains open in the description of the algorithm is the concrete assignment of bin types and the choice of the parameters k and T. Our choice of filling heights is inspired by the parameters from the lower bound by Balogh et al. [4] (or equivalently, of §2.1), but in order to get the desired upper bound instead, each filling height is essentially replaced by the next smaller filling height from the lower bound. Let $\alpha := 1 - 1/(W_{-1}(-2/e^3)+1) \approx 1.3871$ be the value of the lower bound.

For each bin clump of size T, we need to take care of the correct fraction of bins of a certain type j, which we implicitly determine by parameters z_j (for notational convenience, we also write $z'_j := \sum_{i=1}^{j} z_i$). For a minor item workload of W, we aim to create roughly

$z'_k \cdot W$ bins, where we choose $z'_k := (1 + \varepsilon/4)\alpha$, hence achieving the desired tight bound for minor items. Note that the frequencies of each bin type as defined in 2.7 can be obtained by $f_i = z_i/z'_k$. The filling height corresponding to bin type k is defined as $w_k := y_k := (z'_k - 1)/z'_k$. Bins of this type have the largest remaining space, which is $1/z'_k$. Intuitively, the reason is that we do not need to reserve space for major items of size at least $1/z'_k$ as packing these items in exclusive bins still results in an approximation ratio of z'_k.

For the other bin types $j < k$, we now choose the parameters y_j according to the geometric series $y_j = \frac{1}{2}(2 \cdot y_k)^{\frac{j-1}{k-1}}$ (see also [4] for the background on why this is a good choice). The filling heights w_j of the different bin types depend almost directly on these parameters: We set $w_j := y_j \ \forall j > 1$ and $w_1 := 1 = y_1 + \frac{1}{2}$. This perceived inconsistency is due to the shift of the other w_j (w.r.t. the lower bound) as explained above.

The remaining values for z_j are set such that $z'_j := y_k/(y_j(1-y_k))$ holds for all j. The values for z'_j are a result of optimizing the number of bins of type $\leq j$ against a class of bad instances where many items of size $1 - y_j + \varepsilon'$ (for some tiny $\varepsilon' > 0$) are inserted. These items can not be packed into the same bins with such types, however fit into bins of type $> j$. Note that the choice of these parameters results in $1/4 < 1 - 1/z'_k = y_k < \ldots < y_1 = 1/2$ and $3/4 < 2 \cdot (z'_k - 1) = z'_1 < \ldots < z'_k = (1 + \varepsilon/4) \cdot \alpha$.

Based on these parameters, a clump of T bins is organized as follows: We choose the size of a bin clump to be $T = \lceil 4z'_k/\varepsilon\alpha \rceil = \lceil 4/\epsilon \rceil + 1$ and the total number of bins of type $\leq j$ to be $\lceil z'_j/z'_k \cdot T \rceil$. Hence, the number of bins of type 1 is $\lceil z_1/z'_k \cdot T \rceil$, whereas for bin types $j > 1$, it is determined by $\lceil z'_j/z'_k \cdot T \rceil - \lceil z'_{j-1}/z'_k \cdot T \rceil$. Note that the rounding implies that the number of bins of some types may be zero.

Finally, we choose the number of bin types to be $k = \lceil 3/\varepsilon \rceil + 1$.

Analysis

The main goal of the upcoming analysis is to bound the number of bins used for a given payload W of minor items. In the following we state properties of the structure of the algorithm's solution.

In order to count the number of bins in our solution, we need the following technical lemma, which essentially resembles the volume constraint Vol_ε of LP_ε. Remember that $\delta = \varepsilon/15$ denotes the maximum size of a minor item.

▶ **Lemma 2.12.** *A set of full bins which consists of at least $z_j \cdot W$ bins of type j for each bin type j contains a workload of at least W of minor items; i.e., $\sum_{j=1}^{k} z_j W \cdot (w_j - \delta) \geq W$.*

▶ **Lemma 2.13.** *For all $1 \leq j \leq k$, the total number of bins of any type $i \leq j$ is at most $(z'_j + \frac{3}{4}\varepsilon\alpha)W$.*

This bound is mainly used for the analysis in §2.3.3, showing that a packing with the parameters as defined above indeed fits the target curve as desired. However, it also directly implies the approximation ratio for instances where only minor items are present (constraint small_ε in LP_ε).

▶ **Corollary 2.14.** *For an instance with only minor items, the algorithm achieves an a.c.r. of $z'_k + \frac{3}{4}\varepsilon\alpha = (1+\varepsilon)\alpha$.*

▶ **Lemma 2.15.** *The recourse (regarding minor items) during an insertion or deletion of a minor item is bounded by $\mathcal{O}(\varepsilon^{-2})$.*

2.3.2 Dealing with Major Items

For major items, i.e. items with a size larger than $\delta = \varepsilon/15$, we use an algorithm called *Myopic Packing* (MP) by Ivković [19] which is a simplified version of the *MMP* algorithm by Ivković and Lloyd [21]. The algorithm is essentially a fully-dynamic variant of Johnson's First Fit Grouping Algorithm [24, 25]. For the sake of completeness, and since [19] is not freely available on the Internet and the algorithm is an essential component of our algorithm, we give a detailed description of this algorithm below.

We divide the major items in four sub-groups, depending on their size: A B (big) item has a size in $\left(\frac{1}{2}, 1\right]$, an L (large) item in $\left(\frac{1}{3}, \frac{1}{2}\right]$, an S (small) item in $\left(\frac{1}{4}, \frac{1}{3}\right]$ and an O (other) item in $\left(\frac{\varepsilon}{15}, \frac{1}{4}\right]$. Ignoring additional O items for now, the following bin types of interest can occur: BL, BS, B, LLS, LL, LSS and SSS. The name of the bin type represents the items of type B, L and S contained in that bin. The additional bin types LS, L, SS, S can only occur at most two times in a packing in total, so they induce an additive constant of at most 2 and thus can be ignored for the analysis. Each different type $\tau \in \mathcal{T} := \{BL, BS, B, LLS, LL, LSS, SSS\}$ is given a priority such that we have a total ordering $<$ on $\mathcal{T}$ which is $BL > BS > B > LLS > LL > LSS > SSS$. Out of the listed bin types above, the MP algorithm utilizes all but bins of type LSS.

The general idea of the algorithm is to insert items in a First Fit manner with disregard of items of a smaller type. This is realized by a recursive procedure which takes items out of a dedicated auxiliary storage and inserts them into the regular packing. When an item is inserted into a bin, items of lower types are removed from this bin and added to the auxiliary storage. It is important that the algorithm maintains a structural property called thoroughness, which reflects the structure of the packing as done by the First Fit Grouping Algorithm by Johnson. This property is formally defined in Lemma 2.16.

In more detail, the algorithm does the following: For an insertion of an item a, it is simply added to the auxiliary storage and a procedure to clear the storage is called. For a deletion of a, all items in the same bin as a are removed from the regular packing and added to the auxiliary storage. Then the procedure to clear the storage is called. At the end of an insert or delete operation, only a constant number of items remains in the auxiliary storage and is packed into at most 2 bins. The procedure to clear the auxiliary storage works as follows:

1. Every B item from the auxiliary storage is inserted into the regular packing. The thoroughness property is maintained by successively searching for fitting L and then S items in bins of a lower type to pair with the new B item. The remaining items from the bins from which the L or S item was removed are moved to the auxiliary storage.
2. L and S items from the auxiliary storage are paired with B bins from the regular packing whenever possible. Bins of the same or higher type are not changed in the process, i.e. an L item can only be paired with a B item of a BS or B bin. Other items from the bin in which these items are inserted are moved to the auxiliary storage.
3. As long as there are at least two L items in the auxiliary storage, they are inserted in a new bin and potentially paired with an S item either from the auxiliary storage or from an existing SSS bin to form an LLS bin. If no fitting S item exists, an LL bin is created. If an S item is taken from a regular bin, the remaining items are moved to the auxiliary storage.
4. As long as there are at least three S items in the auxiliary storage, new bins of type SSS are formed with these items and inserted into the regular packing. At the end of this step, the auxiliary storage contains at most one L item and two S items which can be packed into at most two bins.
5. All remaining O items in the auxiliary storage are moved to the regular packing in a first fit manner. This implies that bins that contain any other item type than O are prioritized over bins that exclusively contain O items.

Properties of the Algorithm

For our analysis of the packing, we mainly use the thoroughness property which is ensured by the algorithm and proven in [19]:

▶ **Lemma 2.16** ([19]). *A bin of type $\tau \in \mathcal{T}$ is* thorough *if there do not exist two bins B_1 and B_2 with lower types $\tau_1, \tau_2 \in \mathcal{T}$, $\tau_1, \tau_2 < \tau$ such that items from B_1 and B_2 can be used to form a bin of type τ. In the solution of the MP algorithm, bins of type BL, BS and LLS are thorough.*

We denote by $\textsc{Alg}_\tau$ and $\textsc{Opt}_\tau$ (e.g. $\textsc{Alg}_{BL}$ and $\textsc{Opt}_{BL}$) the number of bins of type $\tau \in \mathcal{T}$ in $\textsc{Alg}$ and $\textsc{Opt}$, respectively. The same notation is adapted for multiple types of bins (e.g. $\textsc{Alg}_{BL,LLS} = \textsc{Alg}_{BL} + \textsc{Alg}_{LLS}$). Using this notation, we get:
$\textsc{Alg} \leq \textsc{Alg}_{BL} + \textsc{Alg}_{BS} + \textsc{Alg}_{B} + \textsc{Alg}_{LLS} + \textsc{Alg}_{LL} + \textsc{Alg}_{SSS} + 2$, and
$\textsc{Opt} \geq \textsc{Opt}_{BL} + \textsc{Opt}_{BS} + \textsc{Opt}_{B} + \textsc{Opt}_{LLS} + \textsc{Opt}_{LL} + \textsc{Opt}_{SSS} + \textsc{Opt}_{LSS}$.

Note that the only reason these are not equalities is that the number of bins of type LS, L, SS or S is between 0 and 2.

We first argue that we may assume that no O items are part of the input: Consider the case that there is a bin containing only O items in $\textsc{Alg}$. Then every bin in $\textsc{Alg}$ except one is filled with items of a cumulative size of at least $3/4$. Together with Lemma 2.13 we then directly get an approximation factor of $(1+\varepsilon)\alpha$ even if we would not combine our solutions for minor and major items at all. Regarding the case that there is no such O bin, since MP packs items in a myopic manner, it would produce the same solution if the O items were not part of the input. Hence we may compare the solution to an instance of the optimal solution, which does not need to consider any O items in its instance.

In order to ease the analysis of the algorithm, we introduce assumptions to the optimal solution and show that these do not increase the value of the optimal solution.

▶ **Lemma 2.17.** *Let* $\textsc{Alg}$ *be the packing of the MP algorithm of a set of major items. For an optimal solution* $\textsc{Opt}$ *of a packing of the same items, the following properties can be assumed without increasing the number of used bins in* $\textsc{Opt}$*:*

1. $\textsc{Opt}$ *does not pack a BS bin containing a B item that is part of a BL bin in* $\textsc{Alg}$.
2. $\textsc{Opt}$ *does not pack a B bin containing a B item that is part of a BL bin in* $\textsc{Alg}$.

We use the thoroughness of the MP algorithm (cf. Lemma 2.16) to show the following four statements that compare the solutions of $\textsc{Alg}$ and $\textsc{Opt}$ with each other. They will later be used in the analysis of the combination of our two approaches.

▶ **Lemma 2.18.** *Let* $\textsc{Alg}$ *be the packing of the MP algorithm and* $\textsc{Opt}$ *an optimal solution:*

1. $\textsc{Alg}_{BL} + \textsc{Alg}_{BS} + \textsc{Alg}_{B} = \textsc{Opt}_{BL} + \textsc{Opt}_{BS} + \textsc{Opt}_{B}$
2. $\textsc{Alg}_{BL} + \textsc{Opt}_{LSS} \geq 2\,(\textsc{Alg}_{LLS,LL} - \textsc{Opt}_{LLS,LL})$
3. $\textsc{Alg}_{BS} + \textsc{Alg}_{LL} + 2\textsc{Opt}_{LSS} \geq 3\,(\textsc{Alg}_{SSS} - \textsc{Opt}_{SSS})$
4. $\textsc{Alg}_{BS} + \textsc{Opt}_{LLS} + 2\textsc{Opt}_{LSS} \geq 3\,(\textsc{Alg}_{SSS} - \textsc{Opt}_{SSS})$

We finally determine the recourse that occurs during the insertion or deletion of a major item. Note that while considering this, we need to also account for possible O items.

▶ **Lemma 2.19.** *The recourse of the MP algorithm (for major items) is bounded by $\mathcal{O}(\varepsilon^{-1})$.*

2.3.3 Combining Major and Minor Items

The two presented approaches for the packing of minor and major items treat these items independently. These independent solutions now have to be combined into one to reach a good approximation guarantee. Assume we obtain these solutions as two sets of bins, where the bins $(B_1^{\text{min}}, \ldots, B_m^{\text{min}})$ are the result of the algorithm for minor items and the bins $\left(B_1^{\text{maj}}, \ldots, B_n^{\text{maj}}\right)$ are the result of the algorithm for major items. Note that although we temporarily ignored items of a size in $(\varepsilon/15, 1/4]$ in the analysis for the major items, the algorithm that combines the two solutions of course does not ignore the potential O items.

We first describe the structure of the packing we want to achieve and then show how to maintain that structure over time. The goal is to create pairs of bins with one bin from each solution while not modifying too many pairs in each time step. For ease of description, we still refer to two bins B_i^{min} and B_j^{maj} as two different bins even though their contents may be packed into the same bin. In such a case, we say that B_i^{min} is paired with B_j^{maj}.

We want to maintain a greedy-style combination of the two lists of bins, which can be described by the following combination process: The list of bins with minor items is (partially) sorted by their type $(1, \ldots, k)$, i.e. bins potentially filled with more minor items appear earlier (the ordering here is different compared to the ordering in §2.3.1 when the minor items are actually packed). The list of bins with major items is sorted by their filling height in decreasing order (regardless of their type). The process iterates over the k bin types in the solution of minor items starting with type 2 (since there is no reserved space in bins of type 1) in increasing order. For each bin B_i^{min} of type j, we iterate over the bins with major items starting with the bins that have the largest filling height. We pair B_i^{min} with a bin with major items B_ℓ^{maj} that has a filling height of at most $1 - w_j$ and for which ℓ is minimal, i.e., the first bin with major items whose items fit into the reserved space of the respective (minor item) bin type.

Note that this process incorporates all bins of the minor solution that contain at least one item, including the ones that are part of a buffer clump. We do not use bins that contain no minor items at all (even if they are already present in the minor algorithm as part of a buffer clump). Such a greedy-style packing can be maintained while only modifying $\mathcal{O}(k)$ pairings per changed bin in either one of the two solutions. A major reason for this is that for the bins with minor items, only their type is of interest. The pairings of bins need to be changed if one of the following happens:

A change in the solution of major items: For each insertion or deletion of a major item, the solution of major items is modified independently first. The above described greedy process is then used to determine which bins with major items need to be matched with which types of bins of minor items. The combination is then modified to fit the new solution by switching out the major bins where needed, starting with those which are paired with bins of minor items of type 2.

A change in the solution of minor items: As for the major items, the solution of minor items is first modified independently upon insertion or deletion of a minor item. The modification may add or remove at most one bin of minor items. In this case, the above greedy approach is used to recalculate which bins with major items need to be matched with which types of bins of minor items. The solution is modified accordingly, starting with the matching with bins with minor items of type 2.

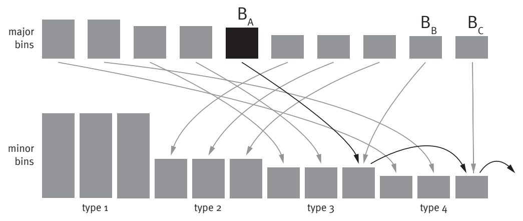

Figure 4 The gray bins represent current bins of the two solutions, the arrows indicate the current combination. The bin B_A is now inserted into the solution for the major items. B_A only fits into minor bins of type 3 or higher. The black arrows indicate the switching process. Bin B_A displaces B_B to a minor bin of type 4. Since there is no bin left for B_C, it is not combined with any minor bin after the changes.

Analysis

Due to the described greedy approach for changes and Lemma 2.15 and 2.19, we can bound the total recourse.

▶ **Lemma 2.20.** *The recourse is bounded by $\mathcal{O}(\varepsilon^{-2})$ for each insertion or deletion.*

▶ **Lemma 2.21.** *Let B^{min} be a bin with minor items of type j. If B^{min} is not combined with a bin of major items, all non-combined bins with major items have a filling height of at least $1 - w_j$.*

It remains to show that our algorithm achieves the desired approximation quality.

▶ **Lemma 2.22.** *Let* $\textsc{Alg}$ *be the number of bins our algorithm uses for an arbitrary input sequence and* $\textsc{Opt}$ *the number of bins used by an optimal solution. Then this yields* $\textsc{Alg} \leq (1+\varepsilon) \cdot \alpha \cdot \textsc{Opt}$.

Proof. We reuse the notation of Sections 2.3.1 and 2.3.2. For the bins with major items of $\textsc{Alg}$, we introduce a collection of bins called *L-S-quartet*, consisting of three LL bins and one SSS bin. The number of such collections is denoted by $Q^{\textsc{Alg}} := \lfloor \min\{\textsc{Alg}_{SSS}, \frac{1}{3}\textsc{Alg}_{LL}\} \rfloor$. We split the number of LL and SSS bins in $\textsc{Alg}$ in the number of bins that can be put in such quartets, denoted by $\textsc{Alg}^{Q}_{LL} := 3Q^{\textsc{Alg}}$ and $\textsc{Alg}^{Q}_{SSS} := Q^{\textsc{Alg}}$, and the respective number of bins that cannot be put in such a collection, denoted by $\textsc{Alg}^{-Q}_{LL}$ and $\textsc{Alg}^{-Q}_{SSS}$. Note that it therefore holds $\textsc{Alg}_{LL} = \textsc{Alg}^{Q}_{LL} + \textsc{Alg}^{-Q}_{LL}$ and $\textsc{Alg}_{SSS} = \textsc{Alg}^{Q}_{SSS} + \textsc{Alg}^{-Q}_{SSS}$. Furthermore, due to the definition of $Q^{\textsc{Alg}}$, it holds $\textsc{Alg}^{-Q}_{LL} \leq 2$ or $\textsc{Alg}^{-Q}_{SSS} = 0$.

Note that we assume that the solution of $\textsc{Alg}$ for the major items does not contain bins with only O items, otherwise the approximation ratio would follow directly as argued in the previous section.

We estimate the optimal solution by adding up all the items which need to be packed. As before, we denote by W the cumulative size of all minor items. We estimate the cumulative size of major items by considering the different types of bins with major items $\tau \in \mathcal{T}$ and their minimal filling height $F(\tau)$ resulting from the minimum size of the respective items (e.g., for bins of type BL, we have $F(BL) = 1/2 + 1/3 = 5/6$). By incorporating the fact that all bins in quartets have an average filling height of $3/4$ we get

$$\textsc{Opt} \geq W + \sum_{\tau \in \mathcal{T}} F(\tau) \cdot \textsc{Alg}^{-Q}_{\tau} + \frac{3}{4}\textsc{Alg}^{Q}_{LL} + \frac{3}{4}\textsc{Alg}^{Q}_{SSS} \text{ as well as} \tag{1}$$

$$\textsc{Opt} \geq W + \sum_{\tau \in \mathcal{T}} F(\tau) \cdot \textsc{Alg}^{-Q}_{\tau} + 3Q^{\textsc{Alg}}. \tag{2}$$

From Lemma 2.21 we get the following: Suppose j is the maximum index such that a bin of type j is not combined with a bin of major items (pick $j = 1$ if such a bin does not exist). Then all remaining bins with major items must have a size of at least $1 - w_j$. Note that if $j = k$, then $\textsc{Alg} \leq (z'_k + \frac{3}{4}\varepsilon\alpha)W + z'_k \cdot W^{maj}$ directly follows, where W^{maj} is the workload of major items, and hence the approximation ratio. We assume $j < k$ in the following. We introduce $\textsc{Alg}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS}$ as the number of bins (with major items) of the given types used by our algorithm, limited to bins with a filling height of at least $1 - w_j$ and excluding all quartets. Hence we have

$$\begin{aligned}\textsc{Alg} &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)W + \textsc{Alg}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS} + \textsc{Alg}^{Q}_{LL,SSS} \\ &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)\textsc{Opt} + \sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \textsc{Alg}^{\geq(1-w_j);-Q}_{\tau} + (1 - \frac{3}{4}z'_j)\textsc{Alg}^{Q}_{LL,SSS} \qquad (3)\end{aligned}$$

from (1) as well as

$$\begin{aligned}\textsc{Alg} &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)W + \textsc{Alg}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS} + 4Q^{\textsc{Alg}} \\ &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)\textsc{Opt} + \sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \textsc{Alg}^{\geq(1-w_j);-Q}_{\tau} + (4 - 3z'_j)Q^{\textsc{Alg}}. \qquad (4)\end{aligned}$$

from (2). We use one of these estimations depending on $Q^{\textsc{Alg}}$.

Let $\hat{z}_j := z'_k - z'_j$. For the two cases with $\textsc{Alg}^{-Q}_{LL} \leq 2$ or $\textsc{Alg}^{-Q}_{SSS} = 0$ we show the following lemmas:

▶ **Lemma 2.23.** *Let* $\textsc{Alg}^{-Q}_{LL} \leq 2$. *Then it holds that*

$$\textsc{Opt} \geq \frac{1}{\hat{z}_j} \cdot \left(\sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \textsc{Alg}^{\geq(1-w_j);-Q}_{\tau} + (1 - \frac{3}{4}z'_j)\textsc{Alg}^{Q}_{LL,SSS}\right).$$

▶ **Lemma 2.24.** *Let* $\textsc{Alg}^{-Q}_{SSS} = 0$. *Then it holds that*

$$\textsc{Opt} \geq \frac{1}{\hat{z}_j} \cdot \left(\sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \textsc{Alg}^{\geq(1-w_j);-Q}_{\tau} + (4 - 3z'_j)Q^{\textsc{Alg}}\right).$$

Hence, from (3) together with Lemma 2.23 or (4) together with Lemma 2.24, we conclude

$$\textsc{Alg} \leq (z'_j + \frac{3}{4}\varepsilon\alpha)\textsc{Opt} + \hat{z}_j \cdot \textsc{Opt} \leq (z'_k + \frac{3}{4}\varepsilon\alpha)\textsc{Opt} = (1 + \varepsilon)\alpha\textsc{Opt}.$$

◀

Finally, Theorem 2.11 now directly follows from our analysis: Lemma 2.22 gives the approximation ratio of $(1 + \varepsilon) \cdot \alpha$ and Lemma 2.20 bounds the worst-case recourse to $\mathcal{O}(\varepsilon^{-2})$.

3 General Movement Costs

We now consider the case of general movement costs, and show a close connection with the (arrival-only) online problem. We first show that the fully-dynamic problem under general movement costs cannot achieve a better a.c.r than the online problem. Next, we match the a.c.r of any SUPER-HARMONIC algorithm for the online problem in the fully-dynamic setting.

3.1 Matching the Lower Bounds for Online Algorithms

Formally, an *adversary process $\mathcal{B}$* for the online BIN PACKING problem is an adaptive process that, depending on the state of the system (i.e., the current set of configurations used to pack the current set of items) either adds a new item to the system, or stops the request sequence. We say that an adversary process shows a lower bound of c for online algorithms for BIN PACKING if for any online algorithm $\mathcal{A}$, this process starting from the empty system always eventually reaches a state where the a.c.r is at least c.

▶ **Theorem 3.1.** *Let $\beta \geq 2$. Any adversary process $\mathcal{B}$ showing a lower bound of c for the a.c.r of any online* BIN PACKING *algorithm can be converted into a fully-dynamic* BIN PACKING *instance with general movement costs such that any fully-dynamic* BIN PACKING *algorithm with amortized recourse at most β must have a.c.r at least c.*

Such a claim is simple for *worst-case* recourse. Indeed, given a recourse bound β, set the movement cost of the i-th item to be $(\beta+\varepsilon)^{n-i}$ for $\varepsilon > 0$. When the i-th item arrives we cannot repack *any* previous item because their movement costs are larger by a factor of $> \beta$. So this is a regular online algorithm. The argument fails, however, if we allow amortization.

To construct our lower bound instance, we start with an adversary process and create a dynamic instance as follows. Each subsequent arriving item has exponentially decreasing (by a factor β) movement cost. When the next item arrives, our algorithm could move certain existing items. These items would have much higher movement cost than the arriving item, and so, this process cannot happen too often. Whenever this happens, we *reset* the state of the process to an earlier time and remove all jobs arriving in the intervening period. This will ensure that the algorithm always behaves like an online algorithm which has not moved any of the existing items. Since it cannot move jobs too often, the set of existing items which have not been moved by the algorithm grow over time. This idea allows us to show: implies:

▶ **Corollary 3.2.** *No fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs and $o(n)$ additive term is better than 1.540-asymptotically competitive.*

3.2 (Nearly) Matching the Upper Bounds for Online Algorithms

We outline some of the key ingredients in obtaining an algorithm with competitive ratio nearly matching our upper bound of the previous section. The first is an algorithm to pack similarly-sized items.

▶ **Lemma 3.3** (Near-Uniform Sizes). *There exists a fully-dynamic* BIN PACKING *algorithm with constant worst-case recourse which given items of sizes $s_i \in [1/k, 1/(k-1))$ for some integer $k \geq 1$, packs them into bins of which all but one contain $k-1$ items and are hence at least $1-1/k$ full. (If all items have size $1/k$, the algorithm packs k items in all bins but one.)*

Lemma 3.3 readily yields a 2-a.c.r algorithm with constant recourse (see [15]). We now discuss how to obtain 1.69-a.c.r based on this lemma and the HARMONIC algorithm [29].

The Harmonic Algorithm. The idea of packing items of nearly equal size together is commonly used in online BIN PACKING algorithms. For example, the HARMONIC algorithm [29] packs large items (of size $\geq \epsilon$) as in Lemma 3.3, while packing small items (of size $\leq \varepsilon$) into dedicated bins which are at least $1-\varepsilon$ full on average, using e.g., FIRSTFIT. This algorithm uses $(1.69 + \mathcal{O}(\varepsilon)) \cdot OPT + \mathcal{O}(\varepsilon^{-1})$ bins [29]. Unfortunately, due to item removals, FIRSTFIT won't suffice to pack small items into nearly-full bins.

To pack small items in a dynamic setting we extend our ideas for the unit cost case, partitioning the bins into *buckets* of $\Theta(1/\epsilon)$ many bins, such that all but one bin in a bucket are $1-\mathcal{O}(\epsilon)$ full, and hence the bins are $1-\mathcal{O}(\epsilon)$ full on average. Since the size and cost are not commensurate, we maintain the small items in sorted order according to their *Smith ratio* (c_i/s_i). However, insertion of a small item can create a large cascade of movements throughout the bucket. We only move items to/from a bin once it has $\Omega(\epsilon)$'s worth of volume removed/inserted (keeping track of erased, or "ghost" items). A potential function argument allows us to show amortized $\mathcal{O}(\epsilon^{-2})$ recourse cost for this approach, implying the following lemma, and by [29], a $(1.69+\mathcal{O}(\varepsilon))$-a.c.r algorithm with $\mathcal{O}(\epsilon^{-2})$ recourse.

▶ **Lemma 3.4.** *For all $\varepsilon \le \frac{1}{6}$ there exists an asymptotically $(1+\mathcal{O}(\varepsilon))$-competitive* BIN PACKING *algorithm with $\mathcal{O}(\varepsilon^{-2})$ amortized recourse if all items have size at most ε.*

Seiden's Super-Harmonic Algorithms. We now discuss our remaining ideas to match the bounds of any Super-Harmonic algorithm [33] in the fully-dynamic setting. A *Super-harmonic* (SH) algorithm partitions the unit interval $[0,1]$ into $K+1$ intervals $[0,\varepsilon],(t_0=\varepsilon,t_1](t_1,t_2],\ldots,(t_{K-1},t_K=1]$. Small items (of size $\le \varepsilon$) are packed into dedicated bins which are $1-\varepsilon$ full. A large item has type i if its size is in the range $(t_{i-1},t_i]$. The algorithm also colors items blue or red. Each bin contains items of at most two distinct item types i and j. If a bin contains only one item type, all its items are colored the same. If a bin contains two item types $i \neq j$, all type i items are colored blue and type j ones are colored red (or vice versa). The SH algorithm is defined by four sequences $(\alpha_i)_{i=1}^K,(\beta_i)_{i=1}^K,(\gamma_i)_{i=1}^K$, and a bipartite *compatibility graph* $\mathcal{G}=(V,E)$. A bin with blue (resp., red) type i items contains at most β_i (resp., γ_i) items of type i, and is *open* if it contains less than this many type i items. The compatibility graph $\mathcal{G}=(V,E)$ has vertex set $V=\{b_i \mid i\in[K]\}\cup\{r_j \mid j\in[K]\}$, with an edge $(b_i,r_j)\in E$ indicating blue items of type i and red items of type j are *compatible* and may share a bin. In addition, an SH algorithm must satisfy the following invariants.

(P1) The number of open bins is $\mathcal{O}(1)$.

(P2) If n_i is the number of type-i items, the number of red type-i items is $\lfloor \alpha_i \cdot n_i \rfloor$.

(P3) If $(b_i,r_j)\in E$ (blue type i items and red type j items are compatible), there is no pair of bins with one containing only blue type i items and one containing only red type j items.

Appropriate choice of $(t_i)_{i=1}^{K+1},(\alpha_i)_{i=1}^K,(\beta_i)_{i=1}^K,(\gamma_i)_{i=1}^K$ and $\mathcal{G}$ allows one to bound the a.c.r of any SH algorithm. (E.g., Seiden gives an SH algorithm with a.c.r 1.589 [33].)

Simulating SH algorithms. In a sense, SH algorithms ignore the exact size of large items, so we can take all items of some type and color. This extends Lemma 3.3 to pack at most β_i or γ_i of them per bin to satisfy Properties 1 and 2. The challenge is in maintaining Property 3: consider a bin with β_i blue type i items and γ_j type-j items, and suppose the type i items are all removed. Suppose there exists an open bin with items of type $i'\neq i$ compatible with j. If the movement costs of both type j and type i' items are significantly higher than the cost of the type i items, we cannot afford to place these groups together, violating Property 3. To avoid such a problem, we use ideas from *stable matchings*. We think of groups of β_i blue type-i items and γ_j red type-j items as nodes in a bipartite graph, with an edge between these nodes if $\mathcal{G}$ contains the edge (b_i,r_j). We maintain a stable matching under updates, with priorities being the value of the costliest item in a group of same-type items packed in the same bin. The stability of this matching implies Property 3; we maintain this stable matching using (a variant of) the Gale-Shapley algorithm. Finally, relying on our solution for packing small items as in §3.2, we can pack the small items in bins which are $1-\varepsilon$ full on average. Combined with the SH bound of Seiden [33], we obtain the following:

▶ **Theorem 3.5.** *There exists a fully-dynamic* BIN PACKING *algorithm with a.c.r* 1.589 *and constant additive term using constant recourse under general movement costs.*

4 Size Movement Costs (Migration Factor)

In this section, we settle the optimal recourse to a.c.r tradeoff for size movement cost (referred to as *migration factor* in the literature); that is, $c_i = s_i$ for each item i. For worst-case recourse in this model (studied in [9, 23, 6]), $poly(\epsilon^{-1})$ upper and lower bounds are known for $(1+\epsilon)$-a.c.r. algorithms [6], though the optimal tradeoff remains elusive. We show that for amortized recourse the optimal tradeoff is $\Theta(\epsilon^{-1})$ recourse for $(1+\epsilon)$-a.c.r.

▶ **Fact 4.1.** *For all* $\epsilon \leq 1/2$*, there exists an algorithm requiring* $(1+\mathcal{O}(\varepsilon))\cdot OPT(\mathcal{I}_t)+\mathcal{O}(\varepsilon^{-2})$ *bins at all times* t *while using only* $\mathcal{O}(\varepsilon^{-1})$ *amortized migration factor.*

This upper bound is trivial – it suffices to repack according to an AFPTAS whenever the volume changes by a multiplicative $(1+\epsilon)$ factor. The challenge here is in showing this algorithm's a.c.r to recourse tradeoff is *tight*. We do so by constructing an instance where addition or removal of small items of size $\approx \varepsilon$ causes *every* near-optimal solution to be far from every near-optimal solution after addition/removal.

▶ **Theorem 4.2.** *For infinitely many* $\epsilon > 0$*, any fully-dynamic* BIN PACKING *algorithm with a.c.r* $(1+\varepsilon)$ *and additive term* $o(n)$ *must have* amortized *migration factor of* $\Omega(\varepsilon^{-1})$*.*

Our matching lower bound relies on the well-known Sylvester sequence [34], given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$, the first few terms of which are $2, 3, 7, 43, 1807, \ldots$ While this sequence has been used previously in the context of BIN PACKING, our proof relies on more fine-grained divisibility properties. In particular, letting c be a positive integer specified later and $\varepsilon := 1/\prod_{\ell=1}^{c} k_\ell$, we use the following properties:

(P1) $\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_c} = 1 - \frac{1}{\prod_{\ell=1}^{c} k_\ell} = 1 - \varepsilon.$

(P2) If $i \neq j$, then k_i and k_j are relatively prime.

(P3) For all $i \in [c]$, the value $1/k_i = \prod_{\ell\in[c]\setminus\{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of ε.

(P4) If $i \neq j \in [c]$, then $1/k_i = \prod_{\ell\in[c]\setminus\{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of $k_j \cdot \epsilon$.

We define a vector of item sizes $\vec{s} \in [0,1]^{c+1}$ in our instances as follows: for $i \in [c]$ we let $s_i = \frac{1}{k_i}\cdot(1-\frac{\varepsilon}{2})$, and $s_{c+1} = \varepsilon \cdot (\frac{3}{2} - \frac{\varepsilon}{2})$. The adversarial input sequence will alternate between two instances, $\mathcal{I}$ and $\mathcal{I}'$. For some large N a product of $\prod_{\ell=1}^{c} k_\ell$, Instance $\mathcal{I}$ consists of N items of sizes s_i for all $i \in [c+1]$. Instance $\mathcal{I}'$ consists of N items of all sizes but s_{c+1}.

Properties 1-4 imply on the one hand that $\mathcal{I}$ can be packed into completely full bins containing one item of each size, while any bin which does not contain exactly one item of each size has at least $\Omega(\varepsilon)$ free space. Similarly, an optimal packing of $\mathcal{I}'$ packs items of the same size in one set of bins, using up exactly $1 - \frac{\epsilon}{2}$ space, while any bin which contains items of at least two sizes has at least ϵ free space. These observations imply the following.

▶ **Lemma 4.3.** *Any algorithm* $\mathcal{A}$ *with* $(1+\varepsilon/7)$*-a.c.r and* $o(n)$ *additive term packs instance* $\mathcal{I}$ *such that at least* $2N/3$ *bins contain exactly one item of each size* s_i*, and packs instance* $\mathcal{I}'$ *such that at least* $N/2$ *bins contain items of exactly one size.*

Theorem 4.2 follows from Lemma 4.3 in a rather straightforward fashion. The full details of this proof and the lemmas leading up to it can be found in [15].

References

1 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728*, 2017.

2 János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theor. Comput. Sci.*, 440-441:1–13, 2012.

3 János Balogh, József Békési, Gábor Galambos, and Mihály Csaba Markót. Improved lower bounds for semi-online bin packing problems. *Computing*, 84(1-2):139–148, 2009. `doi:10.1007/s00607-008-0023-6`.

4 János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. Lower bound for the online bin packing problem with restricted repacking. *SIAM J. Comput.*, 38(1):398–410, 2008. `doi:10.1137/050647049`.

5 János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. On-line bin packing with restricted repacking. *J. Comb. Optim.*, 27(1):115–131, 2014.

6 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. In *APPROX-RANDOM*, volume 40 of *LIPIcs*, pages 135–151. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

7 Edward G Coffman Jr, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

8 Robert M Corless, Gaston H Gonnet, D EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert-W function. *Advances in Computational mathematics*, 5(1):329–359, 1996.

9 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`.

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

11 Björn Feldkord, Matthias Feldotto, and Sören Riechers. A tight approximation for fully dynamic bin packing without bundling. *arXiv preprint arXiv:1711.01231*, 2017.

12 Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. New algorithms for on-line bin packing. In *Algorithms and Complexity, Proceedings of the First Italian Conference*, pages 44–59, 1990.

13 Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. Algorithms for the relaxed online bin-packing model. *SIAM J. Comput.*, 30(5):1532–1551, 2000. `doi:10.1137/S0097539799180408`.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

15 Anupam Gupta, Guru Guruganesh, Amit Kumar, and David Wajc. Fully-dynamic bin packing with limited repacking. *arXiv preprint arXiv:1711.02078*, 2017.

16 Sandy Heydrich and Rob van Stee. Beating the harmonic lower bound for online bin packing. In *43rd International Colloquium on Automata, Languages, and Programming.* Schloss Dagstuhl, 2016.

17 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017.

18 Dorit S Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., 1996.

19 Zoran Ivković. *Fully dynamic approximation algorithms.* PhD thesis, University of Delaware, 1996.

20 Zoran Ivković and Errol L. Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Inf. Process. Lett.*, 59(4):229–232, 1996.

21 Zoran Ivković and Errol L. Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM J. Comput.*, 28(2):574–611, 1998. doi:10.1137/S0097539794276749.

22 Zoran Ivković and Errol L. Lloyd. Fully dynamic bin packing. In *Fundamental Problems in Computing*, pages 407–434. Springer, 2009.

23 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.

24 David S. Johnson. Fast allocation algorithms. In *13th Annual Symposium on Switching and Automata Theory, College Park, Maryland, USA, October 25-27, 1972*, pages 144–154. IEEE Computer Society, 1972. doi:10.1109/SWAT.1972.4.

25 David S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, 1973.

26 David S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974. doi:10.1016/S0022-0000(74)80026-7.

27 David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

28 Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983. doi:10.1137/0212014.

29 Chung C. Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985. doi:10.1145/3828.3833.

30 Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989.

31 Michael B Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203–227, 1991.

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

33 Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49(5):640–671, 2002. doi:10.1145/585265.585269.

34 James J Sylvester. On a point in the theory of vulgar fractions. *American Journal of Mathematics*, 3(4):332–335, 1880.

35 Jeffrey D. Ullman. The performance of a memory allocation algorithm. Technical report, Princeton University. Department of Electrical Engineering. Computer Science Laboratory, 1971.

36 André van Vliet. *Lower and Upper Bounds for On-line Bin Packing and Scheduling Heuristics: Onder-en Bovengrenzen Voor On-line Bin Packing en Scheduling Heuristieken.* Thesis Publ., 1995.

37 Gerhard Woeginger. Improved space for bounded-space, on-line bin-packing. *SIAM Journal on Discrete Mathematics*, 6(4):575–581, 1993.

38 Prudence W. H. Wong, Fencol C. C. Yung, and Mihai Burcea. An 8/3 lower bound for online dynamic bin packing. In *ISAAC*, volume 7676 of *Lecture Notes in Computer Science*, pages 44–53. Springer, 2012.

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