Fully-Dynamic Bin Packing with Limited Repacking*†

Anupam Gupta[1], Guru Guruganesh[1], Amit Kumar[2], and David Wajc[1]

[1]Carnegie Mellon University
[2]IIT Delhi

"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.

*This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.
†An extended abstract of this work, merged with [10], will appear in ICALP 2018.

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 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., [15, Chapter 2] and [6] for surveys. Much of this work (e.g. [2, 13, 18, 20, 21, 22, 24, 27, 28, 29]) starting with [26] 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 $O(\log OPT)$ [14], 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. In our application, 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.

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.

Related Work. Gambosi et al. [11, 12] were the first to study dynamic BIN PACKING, and gave a 4/3-a.c.r algorithm for the insertion-only setting which moves each item $O(1)$ times. This gives amortized $O(1)$ recourse for general movement costs. For unit movement costs, Ivković and Lloyd [16] and Balogh et al. [3, 4] gave lower bounds of 4/3 and 1.3871 on the a.c.r of algorithms with $O(1)$ recourse; Balogh et al. [4] gave an algorithm with a.c.r $3/2 + \varepsilon$ and $O(\varepsilon^{-1})$ movements per update in the insertion-only setting. Following Sanders et al. [23], 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 [17] improved the recourse to $poly(\varepsilon^{-1})$ for insertion-only algorithms, and Berndt et al. [5] gave the same recourse for fully-dynamic algorithms. They also 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.

1.1 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 §A for a tabular listing of our results contrasted with the best previous results.) In the context of sensitivity analysis (see [23]), our bounds provide tight characterization of the *average* change between *approximately*-optimal solutions of slightly modified instances.

Unit Costs: Consider the most natural movement cost: *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 [7]). Balogh et al. [3] showed 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 a matching algorithm proving α is tight for this problem.[1]

Theorem 1.2 (Unit Costs Tight Bounds)**.** *For any $\varepsilon > 0$, there exists a fully-dynamic* BIN PACKING *algorithm with a.c.r $(\alpha+\varepsilon)$, additive term $O(\varepsilon^{-2})$ and amortized recourse $O(\varepsilon^{-2})$, or additive term $poly(\varepsilon^{-1})$ and worst case recourse $\tilde{O}(\varepsilon^{-4})$ under unit movement costs. 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 no easier than the arrival-only online problem (with no repacking).

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 [24].*

[1]Independently of this work, Feldkord et al. [10] provided a different optimal algorithm for unit movement costs. This then inspired us to provide our worst-case recourse algorithm under movement costs using our framework.

The current best online BIN PACKING algorithm [1] is not from the Super Harmonic family but is closely related to them. 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 $O(\varepsilon^{-2})$ additive term and $O(\varepsilon^{-1})$ 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 [5]; 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.

1.2 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 [3]. 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 [3]. However, rather than bound this LP precisely, we exhibit a near-optimal dual solution showing a lower bound of $\alpha - \varepsilon$.

For the 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 $O(\varepsilon^{-2})$ *worst case* recourse if all items have size $O(\varepsilon)$. 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. [5] 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 $\alpha + O(\varepsilon)$.

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 [24] 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 $O(1)$ 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. [5] 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 $1-O(\epsilon)$ full, guaranteeing these bins are $1-O(\epsilon)$ full on average. While for the size-cost case, bucketing bins and sorting the small items by size readily yields an $O(\epsilon^{-1})$ 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 $O(\epsilon^{-2})$ 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 [25]. 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) $O(N)$ 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. Full proofs are deferred to §B,§C and §D, respectively.

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. [3] showed that any $O(1)$-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 [3]. 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 [3]. Let $1/2 > \delta > 0$, $c = 1/\delta - 1$, and let $B \geq 1/\varepsilon$ be a large integer. Our hard instances consist of *small* items of size $1/B^c$, 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 $\lfloor B^{c+1} \rfloor$ small items, and for each $\ell \in \mathcal{S}_\varepsilon$, the input $\mathcal{I}_\ell$ consists of $\lfloor B^{c+1} \rfloor$ small items followed by $\lfloor \frac{B}{1-\ell} \rfloor$ size-ℓ items. The optimal bin packings for $\mathcal{I}_s$ and $\mathcal{I}_\ell$ require precisely $OPT(\mathcal{I}_s) = B$ and $OPT(\mathcal{I}_\ell) = \lceil \frac{B}{1-\ell} \rceil$ 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 $\ell \in \mathcal{S}_\varepsilon$.

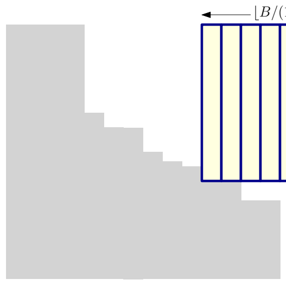


Figure 1: A packing of instance $\mathcal{I}_\ell$. The ℓ-sized items (in yellow) are packed "on top" of the small items (in grey).

As a warm-up we show that the linear program (LP$_\varepsilon$) 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 $Vol(\mathcal{I}_s) \geq B - 1/B^c$, implying constraint (Vol$_\varepsilon$). Moreover, as $OPT(\mathcal{I}_s) = B$, the α_ε-competitiveness of $\mathcal{A}$ implies constraint (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 $OPT(\mathcal{I}_\ell) \leq \lceil \frac{B}{1-\ell} \rceil$ we get constraint (CR$_\varepsilon$). Hence the optimal value of (LP$_\varepsilon$) is a valid lower bound on the competitive ratio α_ε.

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

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$_\varepsilon$) *satisfies $\alpha^\star_\varepsilon \in [\alpha - O(\varepsilon), \alpha + O(\varepsilon)]$.*

To extend the above argument to the fully-dynamic case, we observe that any solution to (LP$_\varepsilon$) defined by packing of $\mathcal{I}_s$ as above must satisfy some constraint (CR$_\varepsilon$) 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 $1/B^c = 1/n^\delta$, 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$_\varepsilon$)*, 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$_\varepsilon$ 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 B 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/B 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$_\varepsilon$). 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 $\alpha + O(\varepsilon)$ if $B/(1-\ell)$ 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 $(\alpha + O(\epsilon))$-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$_\varepsilon$.

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

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 B, 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 $\mathcal{I}^k_\ell$ 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 $\mathcal{I}^k_\ell$ with $k = \lfloor B/(1-\ell) \rfloor$.)

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 $\mathcal{I}^k_\ell$ 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.*

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 $2\varepsilon\cdot OPT(\mathcal{I})+O(f(\varepsilon^{-1}))$ 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. 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 $\mathcal{I}^k_\ell$ for some k and $\ell > 1/2$, packed in no more bins than $\mathcal{I}'$ (see Figure 2). By Lemma 2.4, the number of bins used by our packing of $\mathcal{I}'$ (and of $\mathcal{I}$) is at most

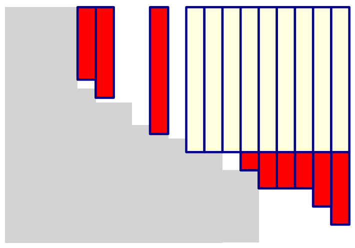


Figure 2: A packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey). Parts of large items not in the instance $\mathcal{I}^k_\ell$ are indicated in red.

$$\alpha_\varepsilon\cdot OPT(\mathcal{I}^k_\ell)\leq\alpha_\varepsilon\cdot OPT(\mathcal{I}')\leq(\alpha_\varepsilon+O(\epsilon))\cdot OPT(\mathcal{I})+O(f(\varepsilon^{-1})).$$

To obtain worst-case bounds, we extend this idea as follows: we near-optimally pack large items of size exceeding 1/4. Given this packing of small and large items, we pack items of size in the range $(\varepsilon,1/4]$ (which we refer to as *medium* items) using first-fit so that we only open a new bin if all bins are at least 3/4 full. If we open a new bin, by a volume bound this packing has a.c.r $4/3<\alpha+O(\epsilon)$. If we don't open a new bin, this packing is $\alpha+O(\epsilon)$ competitive against an easier instance (obtained by removing the medium items), and so we are $\alpha+O(\epsilon)$ competitive. These ideas underlie the following two theorems, a more complete proof of which appears in §B.2.

Theorem 2.6. *An α_ε-feasible packing of the small items of an instance $\mathcal{I}$ can be extended into a packing of all of $\mathcal{I}$ using at most $(\alpha_\epsilon+O(\varepsilon))\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$ bins in linear time for any fixed ϵ.*

Theorem 2.7. *For a dynamic instance $\mathcal{I}_t$, given a fully-dynamic $(1+\epsilon)$-a.c.r algorithm with additive term $f(\epsilon^{-1})$ for items of size greater than 1/4 in $\mathcal{I}_t$, and a fully-dynamic α_ε-feasible packing of its small items, one can maintain a packing with a.c.r $(\alpha_\epsilon+O(\varepsilon))$ with additive term $O(f(\epsilon^{-1}))$. This can be done using worst-case recourse*

1. *$O(\varepsilon^{-1})$ per item move in the near-optimal fully-dynamic packing of the items of size $>1/4$,*
2. *$O(\epsilon^{-1})$ per insertion or deletion of medium items, and*
3. *$O(1)$ per item move in the α_ϵ-feasible packing of the small items.*

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 2.8 (Bin curve-fitting)**.** *Given a list of bin sizes $0\leq b_0\leq b_1\leq\ldots,b_K\leq 1$ and relative frequencies $f_0,f_1,f_2,\ldots,f_K$, such that $f_x\geq 0$ 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 $s_1,\ldots,s_m\leq 1$ 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., $T = O(\varepsilon^{-1})$. Our approach is inspired by the algorithm of [17], 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.

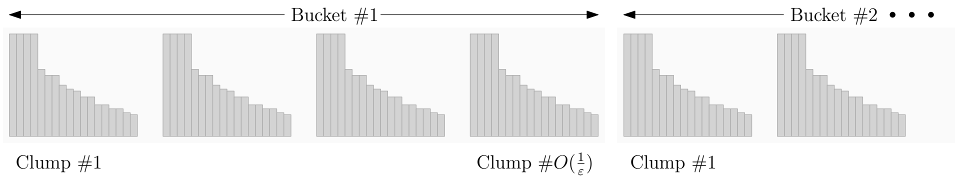


Figure 3: Buckets have $O(\varepsilon^{-1})$ clumps, clumps have T bins.

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 $O(T/\epsilon)$ 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 $O(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $O(\varepsilon)$ fraction of bins have no guarantees whatsoever.

We now relate this process to the value of LP_ε. We first show that setting $T = O(\varepsilon^{-1})$ 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 ($\text{LP}_\varepsilon$) depends on the total volume B of small items, and f_x may change if B changes, it is convenient to work with a normalized version, referred to as (LPnew_ε) (see Figure 4 in §B). Now n_x can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$. However, we also need each f_x to be an integer multiple of $1/T$ for some integer $T = O(\varepsilon^{-1})$. We achieve this by slightly modifying the LP solution, obtaining the following.

Lemma 2.9 (Multiples of ε). *For any optimal solution $\{n_x\}$ to (LPnew_ε) with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + O(\varepsilon)$.*

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

Lemma 2.10 (Small Items Follow the LP). *Let $\varepsilon \leq 1/6$. Using $O(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $O(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing.*

2.2.3 Our Algorithm

From Lemma 2.9 and Lemma 2.10, we can maintain an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing of the small items (that is, a packing of inducing a solution to (LP_ε) with objective value $(\alpha^\star_\varepsilon + O(\varepsilon))$), all while using $O(\varepsilon^{-2})$ worst-case recourse. From Theorem 2.6 and Theorem 2.7, using a $(1+\varepsilon)$-a.c.r packing of the large items one can extend such a packing of the small items into an $(\alpha^\star_\varepsilon + O(\varepsilon))$-approximate packing for $\mathcal{I}_t$, where $\alpha^\star_\varepsilon \leq \alpha + O(\varepsilon)$, 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 $(\alpha + O(\varepsilon))$-a.c.r throughout the epoch. As for this approach's recourse, we note that the number of large items at the end of an epoch of length $\epsilon \cdot N$ is at most $O(N/\epsilon)$, and so repacking them incurs $O(N/\varepsilon)/(\epsilon \cdot N) = O(\epsilon^{-2})$ 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. [5, Theorem 9] 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 2.7, our $(\alpha + O(\varepsilon))$-feasible solution for the small items of $\mathcal{I}_t$ can be extended dynamically to an $(\alpha + (O(\varepsilon)))$-a.c.r packing of all of $\mathcal{I}_t$, using $\tilde{O}(\varepsilon^{-4})$ worst-case recourse.

From the above discussions we obtain our main result – tight algorithms for unit costs.

Theorem 2.11 (Unit Costs: Upper Bound)**.** *There is a polytime fully-dynamic* BIN PACKING *algorithm which achieves $\alpha + O(\varepsilon)$ a.c.r, additive term $O(\varepsilon^{-2})$ and $O(\varepsilon^{-2})$ amortized recourse, or additive term poly(ε^{-1}) and $\tilde{O}(\varepsilon^{-4})$ worst case recourse, under unit movement costs.*

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:

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 §C.2.1). We now discuss how to obtain 1.69-a.c.r based on this lemma and the HARMONIC algorithm [20].

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 [20] 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 $(1.69+O(\varepsilon))\cdot OPT+O(\varepsilon^{-1})$ bins [20]. 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 $1 - O(\epsilon)$ full, and hence the bins are $1-O(\epsilon)$ 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 $O(\epsilon^{-2})$ recourse cost for this approach, implying the following lemma, and by [20], a $(1.69 + O(\varepsilon))$-a.c.r algorithm with $O(\epsilon^{-2})$ recourse.

Lemma 3.4. *For all $\varepsilon \leq \frac{1}{6}$ there exists an asymptotically $(1 + O(\varepsilon))$-competitive bin packing algorithm with $O(\varepsilon^{-2})$ 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 [24] 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 $\leq \varepsilon$) 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 $O(1)$.

(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 [24].)

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 (P1) and (P2). The challenge is in maintaining Property (P3): 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 (P3). 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 (P3); 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 Lemma C.2, we obtain 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 [5, 9, 17]), $poly(\epsilon^{-1})$ upper and lower bounds are known for $(1+\epsilon)$-a.c.r. algorithms [5], 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 $(1+O(\varepsilon))\cdot OPT(\mathcal{I}_t)+O(\varepsilon^{-2})$ bins at all times t while using only $O(\varepsilon^{-1})$ 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 (for completeness we prove this fact in §D). 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 [25], given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j\le i} k_j\right) + 1$, 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 (P1)-(P4) 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 §D.

5 Acknowledgments

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.

Appendix

A Tabular List of Prior and Current Results

Here we recap and contrast the previous upper and lower bounds for dynamic bin packing with bounded recourse, starting with the upper bounds, in Table 1.

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

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
General	2	$\log n$	1	✗	w.c. if n known	**Fact C.1**
	1.589	1	1	✗		**Theorem 3.5**
	1.333	1	1	✗	insertions only	Gambosi et al. [12]
Unit	$1.5+\varepsilon$	ϵ^{-1}	ϵ^{-1}	✓	insertions only	Balogh et al. [4]
	$\alpha+\varepsilon$	ϵ^{-2}	ε^{-2}	✗	$\alpha \approx 1.387$	**Theorem 2.11**
	$\alpha+\varepsilon$	$poly(\epsilon^{-1})$	$\varepsilon^{-4}\log(\varepsilon^{-1})$	✓	$\alpha \approx 1.387$	**Theorem 2.11**
	$\alpha+\varepsilon$	ϵ^{-1}	ε^{-2}	✓	$\alpha \approx 1.387$	Feldkord et al. [10]
Size	$1+\varepsilon$	1	$\varepsilon^{-O(\epsilon^{-2})}$	✓	insertions only	Epstein and Levin [9]
	$1+\varepsilon$	ϵ^{-2}	ϵ^{-4}	✓	insertions only	Jansen and Klein [17]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-3}\log(\varepsilon^{-1})$	✓	insertions only	Berndt et al. [5]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-4}\log(\varepsilon^{-1})$	✓		Berndt et al. [5]
	$1+\varepsilon$	ϵ^{-2}	ϵ^{-1}	✗		**Theorem 4.1**

We contrast these upper bounds with matching and nearly-matching lower bounds, in Table 2. Note that here an amortized bound is a stronger bound than a corresponding worst case bound.

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

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
General	1.540	$o(n)$	∞	✗	as hard as online	**Theorem 3.1**
Unit	1.333	$o(n)$	1	✓		Ivković and Lloyd [16]
	$\alpha-\varepsilon$	$o(n)$	1	✓	$\alpha \approx 1.387$	Balogh et al. [3]
	$\alpha-\varepsilon$	$o(\epsilon^2 \cdot n^\delta)$	$\Omega(\varepsilon^2 \cdot n^{1-\delta})$	✗	for all $\delta \in (0, 1/2]$	**Theorem 2.1**
Size	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✓		Berndt et al. [5]
	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✗		**Theorem 4.2**

We emphasize again the tightness and near-tightness of our upper and lower bounds for the different movement costs. For general movement costs we show that the problem is at least as hard as online bin packing (without repacking), while the problem admits a 1.589-asymptotically competitive algorithm, nearly matching the state of the art 1.578 online algorithm of [1]. For unit movement costs, we show the lower bound of $\alpha \approx 1.387$ of Balogh et al. [3] is sharp, by presenting an algorithm with a.c.r of $\alpha + \epsilon$ and additive term and recourse polynomial in $1/\epsilon$. We further simplify and strengthen the previous lower bound, by showing that any algorithm with a.c.r better than α requires polynomial additive term times recourse. Finally, for size movement costs, we show that an a.c.r of $1 + \epsilon$ implies a recourse cost of $\Omega(\epsilon^{-1})$, even allowing for amortization (strengthening the hardness result of Berndt et al. [5]). A simple lazy algorithm which matches this bound proves this tradeoff to be optimal too.

B Omitted Proofs of Section 2 (Unit Movement Costs)

Here we provide proofs for our unit recourse upper and lower bounds.

First, we show that the optimal value of (LP$_\varepsilon$), restated below for ease of reference, is roughly $\alpha \approx 1.387$, where we recall that α is such that $1 - 1/\alpha$ is a solution to the following equation

$$3 + \ln(1/2) = \ln(x) + 1/x. \tag{1}$$

Lemma 2.2. *The optimal value $\alpha_\varepsilon^\star$ of* (LP$_\varepsilon$) *satisfies $\alpha_\varepsilon^\star \in [\alpha - O(\varepsilon), \alpha + O(\varepsilon)]$.*

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

Proof. We first modify (LP$_\varepsilon$) slightly to make it easier to work with—this will affect its optimal value only by $O(\varepsilon)$.

(i) Change the $B - 1/B^c$ term in the RHS of inequality (Vol$_\varepsilon$) to B, and remove the floor and ceiling in the inequalities (CR$_\varepsilon$). As $B \geq 1/\varepsilon$, this affects the optimal value by $O(\varepsilon)$.

(ii) Divide the inequalities through by B, and introduce new variables n_x for N_x/B, and

(iii) Replace $\alpha_\varepsilon - 1$ by a new variable α'_ε, and change the objective value to $\alpha'_\varepsilon + 1$.

This gives the LP (LPnew$_\varepsilon$), whose optimal value is $\alpha_\varepsilon^\star \pm O(\varepsilon)$. We will provide a feasible solution to this linear program and a feasible dual solution for its dual linear program (Dual$_\varepsilon$) whose objective values are $\alpha + O(\varepsilon)$ and $\alpha - O(\varepsilon)$, respectively. This proves the desired result.

$$\begin{array}{rlll}
\text{min. } \alpha'_\varepsilon + 1 & & & (\mathrm{LPnew}_\varepsilon)\\
n_0 + \sum_{x\in\mathcal{S}_\varepsilon}(1-x)\cdot n_x \geq 1 & & &\\
n_0 + \sum_{x\in\mathcal{S}_\varepsilon} n_x - \alpha'_\varepsilon & \leq 1 & &\\
n_0 + \sum_{x\in\mathcal{S}_\varepsilon, x\leq t-\varepsilon} n_x & \leq \dfrac{\alpha'_\varepsilon}{1-t} & \forall t\in\mathcal{S}_\varepsilon &\\
n_x \geq 0 & & &
\end{array}$$

$$\begin{array}{rlll}
\text{max. } Z - q_0 + 1 & & & (\mathrm{Dual}_\varepsilon)\\
q_0 + \sum_{t\in\mathcal{S}_\varepsilon} \dfrac{q_t}{1-t} & \leq 1 & & (\mathrm{d1})\\
q_0 + \sum_{t\in\mathcal{S}_\varepsilon} q_t & \geq Z & & (\mathrm{d2})\\
q_0 + \sum_{t\geq x+\varepsilon, t\in\mathcal{S}_\varepsilon} q_t & \geq (1-x)\cdot Z & \forall x\in\mathcal{S}_\varepsilon & \\
 & & & (\mathrm{d3})
\end{array}$$

Figure 4: The modified LP and its dual program

Upper Bounding OPT(LPnew$_\varepsilon$). We first give a solution that is nearly feasible, and then modify it to give a feasible solution with value at most $\alpha + O(\varepsilon)$.

Let C denote $\alpha - 1$. Define $n_x := \int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\, dy$ for all $x \in \mathcal{S}_\varepsilon$ and

$$n_0 := 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C \cdot dy}{(1-y)} = 1 - C \ln\Big(\frac{1}{2}\Big) + C \ln\Big(1 - \frac{1}{\alpha}\Big).$$

The first constraint of (LPnew$_\varepsilon$) is satisfied up to an additive $O(\varepsilon)$:

$$\begin{aligned} n_0 + \sum_x (1-x) n_x &= 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C}{(1-y)}\, dy + \sum_{x \in \mathcal{S}_\varepsilon} (1-x) \int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\, dy \\ &\geq 1 - \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C}{(1-y)}\, dy + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C(1-y)}{(1-y)^2}\, dy - O(\varepsilon) \\ &= 1 - O(\varepsilon). \end{aligned}$$

Next, the second constraint of (LPnew$_\varepsilon$).

$$\begin{aligned} n_0 + \sum_{x \in \mathcal{S}_\varepsilon} n_x &= 1 - \int_{1/2}^{\frac{1}{\alpha}} \frac{C \cdot dy}{1-y} + \sum_{x \in \mathcal{S}_\varepsilon} \int_{x-\varepsilon}^{x} \frac{C \cdot dy}{(1-y)^2} \\ &= 1 - \int_{1/2}^{\frac{1}{\alpha}} \frac{C \cdot dy}{1-y} + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{C \cdot dy}{(1-y)^2} \\ &= 1 + C\Big(- \ln(1/2) + \ln\Big(1 - \frac{1}{\alpha}\Big) - 2 + \frac{1}{1 - \frac{1}{\alpha}}\Big) \\ &= 1 + C \\ &= \alpha, \end{aligned}$$

where the penultimate step follows by (1), and the last step uses $C = \alpha - 1$. Performing a similar calculation for the last set of constraints in (LPnew$_\varepsilon$), we get

$$\begin{aligned} n_0 + \sum_{x \in \mathcal{S}_\varepsilon, x < t - \varepsilon} n_x &= n_0 + \sum_{x \in \mathcal{S}_\varepsilon} n_x - \sum_{x \in \mathcal{S}_\varepsilon, x \geq t - \varepsilon} n_x \\ &= \alpha - \sum_{x \geq t-\varepsilon, x \in \mathcal{S}_\varepsilon} \int_{x-\varepsilon}^{x} \frac{C}{(1-y)^2}\, dy \\ &\leq \alpha - \int_{t}^{\frac{1}{\alpha}} \frac{C}{(1-y)^2}\, dy + O(\varepsilon) \\ &= \alpha - C\Big(\frac{\alpha}{\alpha - 1} - \frac{1}{1-t}\Big) + O(\varepsilon) \\ &= \frac{\alpha - 1}{1-t} + O(\varepsilon), \end{aligned}$$

where the second equality follows from the previous sequence of calculations. To satisfy the constraints of (LPnew$_\varepsilon$), we increase n_0 to $n_0 + O(\varepsilon)$ and set α'_ε to $\alpha - 1 + O(\varepsilon)$. Since t is always $\geq 1/2$, this will also satisfy the last set of constraints. Since the optimal value of (LPnew$_\varepsilon$) is $\leq \alpha - 1 + O(\varepsilon)$, which implies that $\alpha^\star_\varepsilon \leq \alpha - 1$, since we had subtracted 1 from the objective function when we constructed (LPnew$_\varepsilon$) from (LP$_\varepsilon$)).

Lower Bounding OPT(LPnew$_\varepsilon$). As with our upper bound on OPT(LPnew$_\varepsilon$), we start with a nearly-feasible dual solution to (Dual$_\varepsilon$) and later modify it to obtain a feasible solution. Set $Z = \alpha(\alpha - 1)$, $q_0 = (\alpha-1)^2$, $q_{\frac{1}{2}+\varepsilon} = \alpha(\alpha-1)/2$, and $q_t = \alpha(\alpha-1)\varepsilon$ for all $t \in \mathcal{S}_\varepsilon$ with $t \geq \frac{1}{2} + 2\varepsilon$. The objective value of (Dual$_\varepsilon$) with respect to this solution is exactly $\alpha(\alpha-1)-(\alpha-1)^2+1=\alpha$. We will now show that it (almost) satisfies the constraints. For sake of brevity, we do not explicitly write that variable t takes values in $\mathcal{S}_\varepsilon$ in the limits for the sums below. First, consider constraint (d1):

$$\begin{aligned}
q_0 + \sum_{t=\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} &= q_0 + \frac{q_{\frac{1}{2}+\varepsilon}}{\frac{1}{2}-\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} \\
&\le q_0 + 2q_{\frac{1}{2}+\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} \frac{q_t}{1-t} \\
&\le (\alpha-1)^2 + \alpha(\alpha-1) + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \frac{\alpha(\alpha-1)\,dx}{1-x} \\
&= (\alpha-1)^2 + \alpha(\alpha-1) + \alpha(\alpha-1)\Big(\ln\left(\frac{1}{2}\right) - \ln\left(1-\frac{1}{\alpha}\right)\Big) \\
&= (\alpha-1)^2 + \alpha(\alpha-1) + \alpha(\alpha-1)\Big(\frac{3-2\alpha}{\alpha-1}\Big) \qquad = 1
\end{aligned}$$

where we used Equation (1), which follows from the definition of α, in the penultimate equation. Next, consider constraint (d2).

$$\begin{aligned}
q_0 + \sum_{t=\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} q_t &= q_0 + q_{\frac{1}{2}+\varepsilon} + \sum_{t>\frac{1}{2}+\varepsilon}^{\frac{1}{\alpha}} q_t \\
&\ge (\alpha-1)^2 + \frac{1}{2}\alpha(\alpha-1) + \int_{\frac{1}{2}}^{\frac{1}{\alpha}} \alpha(\alpha-1)\,dx - O(\varepsilon) \\
&= (\alpha-1)^2 + \frac{1}{2}\alpha(\alpha-1) + \alpha(\alpha-1)\Big(\frac{1}{\alpha}-\frac{1}{2}\Big) - O(\varepsilon) \qquad = Z - O(\varepsilon)
\end{aligned}$$

Finally, consider constraint (d3) for any $x \in \mathcal{S}_\varepsilon$:

$$\begin{aligned}
q_0 + \sum_{t\ge x+\varepsilon, t\in\mathcal{S}_\varepsilon} q_t &\ge (\alpha-1)^2 + \int_{x+\varepsilon}^{\frac{1}{\alpha}} \alpha(\alpha-1)\,dx - O(\varepsilon) \\
&= (\alpha-1)^2 + \Big(\frac{1}{\alpha} - x - \varepsilon\Big)\alpha(\alpha-1) - O(\varepsilon) \\
&= (\alpha-1)\Big(\alpha - 1 + \Big(\frac{1}{\alpha} - x - \varepsilon\Big)\cdot\alpha\Big) - O(\varepsilon) \\
&= Z\cdot(1-x-\varepsilon) - O(\varepsilon).
\end{aligned}$$

Finally, increase q_0 to $q_0 + O(\varepsilon)$ to ensure that constraints (d2) and (d3) are satisfied. This is now a feasible solution to (Dual$_\varepsilon$) with objective value $Z - q_o + 1 - O(\varepsilon) = \alpha - O(\varepsilon)$. Hence the optimal value $\alpha^\star_\varepsilon$ for the LP (LPnew$_\varepsilon$) is at least $\alpha - O(\varepsilon)$. $\square$

B.1 Proof of the Lower Bound

We now prove that $\alpha^\star_\varepsilon$, the optimal value of (LP$_\varepsilon$), is such that any algorithm with a.c.r below this $\alpha^\star_\varepsilon$ must have either polynomial additive term or recourse (or both).

Lemma 2.3. *For all $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\alpha^\star_\varepsilon$ is the optimal value of* (LP$_\varepsilon$)*, 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.*

Proof. For any $x \in \mathcal{S}_\varepsilon \cup \{0\}$, again define N_x as the number of bins with free space in the range $[x, x+\varepsilon)$ when $\mathcal{A}$ faces input $\mathcal{I}_s$. Inequality (Vol$_\varepsilon$) is satisfied for the same reason as above. Recall that $B = \Theta(n^\delta)$. As $\mathcal{A}$ is $(\alpha_\varepsilon^\star - \Omega(\varepsilon))$-asymptotically competitive with additive term $o(\varepsilon \cdot n^\delta)$, i.e., $o(\varepsilon \cdot B)$, and $OPT(\mathcal{I}_s) = B$, we have $N_0 + \sum_{x \in \mathcal{S}_\varepsilon} N_x \leq (\alpha_\varepsilon^\star - \Omega(\varepsilon) + o(\varepsilon)) \cdot B \leq \alpha_\varepsilon^\star \cdot B$. That is, the N_x's satisfy constraint (small$_\varepsilon$) with $\alpha_\varepsilon = \alpha_\varepsilon^\star$.

We now claim that there exists a $\ell \in \mathcal{S}_\varepsilon$ such that

$$N_0 + \sum_{x \in \mathcal{S}_\varepsilon, x \leq \ell - \varepsilon} N_x + \left\lfloor \frac{B}{1-\ell} \right\rfloor \geq \alpha_\varepsilon^\star \cdot \left\lceil \frac{B}{1-\ell} \right\rceil \tag{2}$$

holds (notice the opposite inequality sign compared to constraint (CR$_\varepsilon$)). Suppose not. Then the quantities N_0, N_x for $x \in \mathcal{S}_\varepsilon$, and $\alpha_\varepsilon^\star$ strictly satisfy the constraints (CR$_\varepsilon$). If they also strictly satisfy the constraint (small$_\varepsilon$), then we can maintain feasibility and slightly reduce $\alpha_\varepsilon^\star$, which contradicts the definition of $\alpha_\varepsilon^\star$. Therefore assume that constraint (small$_\varepsilon$) is satisfied with equality. Now two cases arise: (i) All but one variable among $\{N_0\} \cup \{N_x \mid x \in \mathcal{S}_\varepsilon\}$ are zero. If this variable is N_0, then tightness of (small$_\varepsilon$) implies that $N_0 = \alpha_\varepsilon^\star B$. But then we satisfy (Vol$_\varepsilon$) with slack, and so, we can reduce N_0 slightly while maintaining feasibility. Now we satisfy all the constraints strictly, and so, we can reduce $\alpha_\varepsilon^\star$, a contradiction. Suppose this variable happens to be N_x, where $x \in \mathcal{S}_\varepsilon$. So, $N_x = \alpha_\varepsilon^\star B$. We will show later in Theorem 2.2 that $\alpha_\varepsilon^\star \leq 1.4$. Since $(1-x) \leq 1/2$, it follows that $(1-x)N_x \leq 0.7B$, and so we satisfy (Vol$_\varepsilon$) with slack. We again get a contradiction as argued for the case when N_0 was non-zero, (ii) There are at least two non-zero variables among $\{N_0\} \cup \{N_x \mid x \in \mathcal{S}_\varepsilon\}$ – let these be N_{x_1} and N_{x_2} with $x_1 < x_2$ (we are allowing x_1 to be 0). Now consider a new solution which keeps all variables N_x unchanged except for changing N_{x_1} to $N_{x_1} + \frac{\eta}{1-x}$, and N_{x_2} to $N_{x_2} - \frac{\eta}{1-x}$, where η is a small enough positive constant (so that we continue to satisfy the constraints (CR$_\varepsilon$) strictly). The LHS of (Vol$_\varepsilon$) does not change, and so we continue to satisfy this. However LHS of (small$_\varepsilon$) decreases strictly. Again, this allows us to reduce $\alpha_\varepsilon^\star$ slightly, which is a contradiction. Thus, there must exist a ℓ which satisfies (2). We fix such a ℓ for the rest of the proof.

Let $\mathcal{B}$ denote the bins which have less than ℓ free space. So, $|\mathcal{B}| = N_0 + \sum_{x \in \mathcal{N}_\varepsilon : x \leq \ell - \varepsilon} N_x$. Now, we insert $\lfloor \frac{B}{1-\ell-\varepsilon} \rfloor$ items of size $\ell + \varepsilon$. (It is possible that $\ell = 1/\alpha$, and so $\ell + \varepsilon \notin \mathcal{S}_\varepsilon$, but this is still a valid instance). We claim that the algorithm must move at least ε volume of small items from at least εB bins in $\mathcal{B}$. Suppose not. Then the large items of size $\ell + \varepsilon$ can be placed in at most εB bins in $\mathcal{B}$. Therefore, the total number of bins needed for $\mathcal{I}_\ell$ is at least $N_0 + \sum_{x \in \mathcal{N}_\varepsilon : x \leq \ell - \varepsilon} N_x - \varepsilon B + \left\lfloor \frac{B}{1-\ell} \right\rfloor$, which by inequality (2), is at least $(\alpha_\varepsilon^\star - O(\varepsilon)) \cdot OPT(\mathcal{I}_{\ell+\varepsilon})$, because $OPT(\mathcal{I}_{\ell+\varepsilon}) = \left\lceil \frac{B}{1-\ell-\varepsilon} \right\rceil = \left\lceil \frac{B}{1-\ell} \right\rceil + O(\varepsilon B)$. But we know that $\mathcal{A}$ is $(\alpha_\varepsilon^\star - \Omega(\varepsilon))$-asymptotically competitive with additive term $o(\varepsilon \cdot n^\delta)$ (which is $o(\varepsilon \cdot OPT(\mathcal{I}_{\ell+\varepsilon}))$. So it should use at most $(\alpha_\varepsilon^\star - \Omega(\varepsilon) + o(\varepsilon)) \cdot OPT(\mathcal{I}_{\ell+\varepsilon})$ bins, which is a contradiction. Since each small item has size $1/B^c$, the total number of items moved by the algorithm is at least $\varepsilon^2 B / B^c$. This is $\Omega(\varepsilon \cdot n^{1-\delta})$, because $\varepsilon \geq 1/B$, and $B^c = \Theta(n^{1-\delta})$. □

B.2 Matching Algorithmic Results

We now address the omitted proofs of our matching upper bound.

B.2.1 LP$_\varepsilon$ as a Factor-Revealing LP

We now present the omitted proofs allowing us to use (LP$_\varepsilon$) to upper bound our algorithm's a.c.r. We start by showing that an optimal solution to (LP$_\varepsilon$) induces a packing of the small items which can be trivially extended (i.e., without moving any items) to an $(\alpha_\varepsilon^\star + O(\varepsilon))$-competitive packing of any number of ℓ-sized items, for any $\ell > 1/2$.

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 $k \in \mathbb{N}$.*

Proof. Fix ℓ and k. Let N_x be the bins with exactly free space in the packing of $\mathcal{I}_s$ and let $N = \sum_{x|x\geq\ell, x\in\mathcal{S}_\varepsilon} N_x$ be the bins with at least ℓ free space; if $\ell \geq 1/\alpha$, then $N = 0$. Let $N' = \sum_{x|x\leq\ell-\varepsilon, x\in\mathcal{S}_\varepsilon} N_x$. Our algorithm first packs the size-ℓ items in the N bins of the packing before using new bins, and hence uses $N' + \max(N, k)$ bins. If $k \leq N$, we are done because of the constraint (small$_\varepsilon$), so assume $k \geq N$. A volume argument bounds the number of bins in the optimal solution for $\mathcal{I}_\ell^k$:

$$\mathrm{OPT}(\mathcal{I}_\ell^k) \geq \begin{cases} k & \text{if } k(1-\ell) \geq B \\ k + \big(B - k(1-\ell)\big) & \text{else.} \end{cases}$$

We now consider two cases:

- $k(1-\ell) \geq B$: Using constraint (CR$_\varepsilon$), the number of bins used by our algorithm is

$$N' + k \leq \tfrac{\alpha_\varepsilon B}{1-\ell} + O(\varepsilon B) + \left(k - \tfrac{B}{1-\ell}\right) \leq (\alpha_\varepsilon + O(\varepsilon))k.$$

- $k(1-\ell) < B$: Since k lies between N and $\frac{B}{1-\ell}$, we can write it as a convex combination $\frac{\lambda_1 B}{1-\ell} + \lambda_2 N$, where $\lambda_1 + \lambda_2 = 1, \lambda_1, \lambda_2 \geq 0$. We can rewrite constraints (small$_\varepsilon$) and (CR$_\varepsilon$) as

$$N' + \tfrac{B}{1-\ell} \leq \tfrac{\alpha_\varepsilon B}{1-\ell} + O(\varepsilon B) \quad \text{and} \quad N' + N \leq \alpha_\varepsilon B.$$

  Combining them with the same multipliers λ_1, λ_2, we see that $N' + k$ is at most

$$\alpha_\varepsilon\left(\tfrac{\lambda_1 B}{1-\ell} + \lambda_2 B\right) + O(\varepsilon B) = \alpha_\varepsilon\left(B + \tfrac{\lambda_1 \ell B}{1-\ell}\right) + O(\varepsilon B) \leq \alpha_\varepsilon\left(B + \ell k\right) + O(\varepsilon B).$$

  The desired result follows because $B + \ell k = k + (B - k(1-\ell))$ and B is a lower bound on $\mathrm{OPT}(\mathcal{I}_\ell^k)$. □

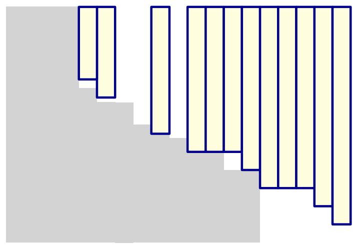
We now proceed to provide a linear-time algorithm which for any fixed ϵ, given an input $\mathcal{I}$ produces a packing into $(1 + O(\varepsilon)) \cdot OPT(\mathcal{I})$ bins such that in almost all bins large items occupy either no space or more than half the bin.

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.*

Proof. Suppose this packing uses $N \leq (1+\varepsilon) \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins of which $B \geq 2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 4$ are "bad" bins – bins with some $v \in (0, 1/2]$ volume taken up by large items. We show that this packing can be improved to require fewer than $OPT(\mathcal{I})$, which would lead to a contradiction. Indeed, repeatedly combining the contents of any two bad bins until no two such bins remain would decrease the number of bins by one and the number of bad bins by at most two for each combination (so this process can be repeated at least $\lfloor B/2 \rfloor \geq \varepsilon \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1}) + 1$), and so would result in a new packing of the $\mathcal{I}$ using at most $N - \lfloor B/2 \rfloor < OPT(\mathcal{I})$ 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 all of $\mathcal{I}$ using at most $(\alpha_\epsilon + O(\varepsilon)) \cdot OPT(\mathcal{I}) + O(\varepsilon^{-2})$ bins in linear time for any fixed ϵ.*

Proof. We run the linear-time algorithm of De La Vega and Lueker [8] to compute a packing of the large items of $\mathcal{I}$ into at most $(1+\varepsilon)\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$ many bins. By Observation 2.5, all but at most $2\varepsilon\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$ of these bins have at most $\frac{1}{2}$ volume occupied by small items. We use these bins in our packing of $\mathcal{I}'$. For the remaining bins we "glue" all large items occupying the same bin into a single *huge* item. Note that any packing of $\mathcal{I}'$ trivially induces a similar packing of the as-of-yet unpacked large items of $\mathcal{I}$ with the same number of bins (simply pack large items glued together in the place of their induced glued item). Moreover, by construction all large items of $\mathcal{I}'$ are huge (i.e., have size greater than 1/2), and clearly $OPT(\mathcal{I}')\leq(1+O(\varepsilon))\cdot OPT(\mathcal{I})+O(\varepsilon^{-2})$, as $\mathcal{I}'$ can be packed using this many bins. As the free space in all bins is an integer multiple of ϵ, we can round the huge items' sizes to integer multiples of ϵ and obtain a packing with the same number of bins for $\mathcal{I}'$. Such a rounding allows us to bucket sort the huge items of $\mathcal{I}'$ (and bins) in time $O(n/\epsilon)$. All the above steps take $O_\varepsilon(n)$ time. It remains to address the obtained packing's approximation ratio.


(a) A greedy packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey).

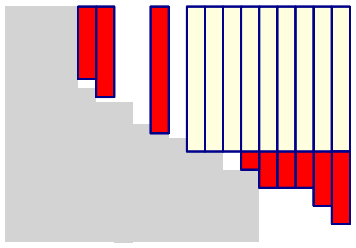

(b) Packing of instance $\mathcal{I}_\ell^k$ obtained from $\mathcal{I}'$ by removing parts of huge items $\mathcal{I}_\ell^k$ (in red).

Figure 5: A packing of instance $\mathcal{I}'$ and the instance $\mathcal{I}_\ell^k$ obtained from $\mathcal{I}'$.

In order to upper bound the number of bins used to pack $\mathcal{I}'$ (and therefore $\mathcal{I}$), we create a new instance of the form $\mathcal{I}_\ell^k$ for some k and $\ell>1/2$. Specifically, if we sort the bins containing small items in decreasing order of free space, we remove all large items packed in a bin with f free space such that some bin with at least f free space contains no large item of $\mathcal{I}$. Let the smallest remaining large item size be ℓ. We decrease the size of all k remaining larger items to ℓ, yielding the instance $\mathcal{I}_\ell^k$ packed on top of the curve using the same number of bins as our packing of $\mathcal{I}$ (see Figure 5b). By Lemma 2.4, ignoring the additive $O(\epsilon^{-2})$ term due to the packing of the large items, the packing of $\mathcal{I}'$ on top of the small items – and hence the packing we obtain for $\mathcal{I}$ – uses at most $\alpha_\varepsilon\cdot OPT(\mathcal{I}_\ell^k)\leq\alpha_\varepsilon\cdot OPT(\mathcal{I}')\leq\alpha_\varepsilon\cdot(1+O(\epsilon))\cdot OPT(\mathcal{I})$ bins. □

Theorem 2.6 immediately gives us amortized recourse bounds. Extending it slightly allows us to obtain worst-case recourse bounds, as follows: replace the static $(1+\epsilon)$-a.c.r algorithm by a dynamic algorithm and round the huge items to sizes which are multiples of ϵ. Doing this naively yields an $(\alpha+O(\epsilon))$ a.c.r, with worst-case recourse bounds that are at most $O(\epsilon^{-3})$ times the recourse bounds of a fully-dynamic $(1+\epsilon)$-a.c.r bin packing algorithm for items of size at least ϵ. Using the worst-case recourse bounds from Berndt et al. [5, Theorem 9], we get an $\tilde{O}(\epsilon^{-6})$ worst-case recourse bound.

The following theorem improves this worst-case recourse bound to being only an $O(\epsilon^{-1})$-times worse, instead of the naive $O(\epsilon^{-3})$ times worse. This, combined with Berndt et al. [5, Theorem 9], gives an improved $\tilde{O}(\epsilon^{-4})$ worst-case recourse bound.

Theorem 2.7. *For a dynamic instance $\mathcal{I}_t$, given a fully-dynamic $(1+\epsilon)$-a.c.r algorithm with additive term $f(\epsilon^{-1})$ for items of size greater than $1/4$ in $\mathcal{I}_t$, and a fully-dynamic α_ε-feasible packing of its small items, one can maintain a packing with a.c.r $(\alpha_\epsilon+O(\varepsilon))$ with additive term $O(f(\epsilon^{-1}))$. This can be done using worst-case recourse*

1. *$O(\varepsilon^{-1})$ per item move in the near-optimal fully-dynamic packing of the items of size $>1/4$,*

2. *$O(\epsilon^{-1})$ per insertion or deletion of medium items, and*

3. *$O(1)$ per item move in the α_ϵ-feasible packing of the small items.*

Proof. The idea here is similar to the proof of Theorem 2.6, and we mainly discuss the recourse bound.

By the theorem's hypothesis, we have a $(1+\epsilon)$-a.c.r packing of the subinstance made up of the large items of size greater than 1/4. We "glue" items in bins which are at least half full into single huge items, yielding an instance $\mathcal{I}'$ with $OPT(\mathcal{I}') \leq (1+O(\varepsilon)) \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1}))$, and which by Observation 2.5 has at most $2\varepsilon \cdot OPT(\mathcal{I}) + O(f(\varepsilon^{-1}))$ non-huge items. We pack these items in individual bins. We pack $\mathcal{I}'$ greedily on top of the curve, by packing the huge items in order of increasing huge item size, according to FirstFit, with the bins sorted in increasing order of free space (see Figure 5a). As in the linear-time algorithm of Theorem 2.6, rounding the huge items' sizes to integer products of ϵ does not increase the number of bins used to pack $\mathcal{I}'$. For Theorem 2.6 this allowed us to pack the huge items in linear time. This rounding also proves useful in obtaining low worst-case recourse, as follows.

First, consider only the small items and the large items of size greater than 1/4. We dynamically pack the former into some α_ϵ-feasible packing, and the large items using the fully-dynamic $(1+\epsilon)$-a.c.r algorithm of the theorem's statement. Whenever a bin is changed in the packing of the large items of size greater than 1/4 (an item added/removed by the dynamic packing algorithm), we either changed a bin which was less than half full (in which case we move its $O(1)$ items in our packing of $\mathcal{I}_t$), or we create a new item in the dynamic instance $\mathcal{I}'_t$. Now, we can dynamically keep the huge items of $\mathcal{I}'_t$ packed as though inserted in sorted FirstFit order as above as follows: whenever a huge item is added, we add it in the bin "after" all items of the same size, possibly removing an item of larger size from this bin, which we then re-insert (deletion is symmetric). As the number of different huge item sizes is $O(\varepsilon^{-1})$, we move at most $O(\varepsilon^{-1})$ huge items, and so at most $O(\varepsilon^{-1})$ large items of $\mathcal{I}_t$ (as these large items have size greater than 1/4, and so the huge items correspond to at most three such items). Finally, we now discuss how to pack medium items (items of size in the range $(\epsilon, 1/4]$).

We strive to keep the medium-sized items on top of our packing for the large and small items so as to guarantee that the bins in the prefix of all but the last bin (sorted by time of opening) which contain a medium-sized item are all at least 3/4 full if we ignore the fact that small item groups and huge items have their sizes rounded to products of ϵ. By a volume argument (accounting for above-mentioned rounding), if we open a new bin, our packing has a.c.r $4/3 + O(\varepsilon) < \alpha + (\varepsilon)$. If no new bin is opened, we can safely ignore the medium-sized items and compare the number of bins we open against an easier instance which does not contain these medium-sized items, which as we shall see later yields an a.c.r of $\alpha + O(\epsilon)$. We now discuss details of dynamically packing medium-sized items in this way.

Insertion of a medium-sized item is trivial using no recourse, by adding it to the first bin which is less than 3/4 full and accommodates this new item (or opening a new bin if necessary). Removal of a medium-sized item is not much harder. Upon the removal of some or all medium-sized items from some bin B, we take the last medium items (according to the bins' order) and reinsert them into B until B is at least 3/4 full or until it is the last bin with medium items. As medium items have size in the range $(\varepsilon, 1/4]$, this requires $O(\epsilon^{-1})$ worst-case recourse. Similarly, a change in the packing of small items can only increase or decrease the volume used in a bin by $O(\varepsilon)$. Such an insertion can be addressed by removing at most $O(1)$ items from the bin until it does not overflow, and then reinserting them (in the case of an insertion of small items into a bin), or by inserting some last $O(1)$ medium items into this bin in the case of deletion. It remains to address changes due to updates in the packing of large items (which are grouped into huge items).

Recall that all the huge items have their sizes rounded up to products of ϵ. This rounding allows us to obtain a packing as in Figure 5a by only repacking $O(\epsilon^{-1})$ such huge items following an update to the packing of the large items, if we ignore medium items. Now, consider the medium items that are displaced due to such a move. Following a removal of a huge item, a huge item of larger size may replace the removed item, and this huge item may be replaced in turn by a larger huge item, and so on for the ϵ^{-1} different huge item sizes. We address potential overflowing of these bins by removing the minimum number of medium items to guarantee these bins do not overflow and repacking them. As each medium item has size greater than ϵ, we remove at most one such item per huge item size class; that is, $O(\epsilon^{-1})$ such items. Similarly, the removal of a huge item potentially causes a larger huge item to take its place, and so on for $O(\epsilon^{-1})$ size classes, until no next-sized huge item exists or the next huge item does not fit. To avoid overfilling of these bins, we move at most one medium item per size class (as the medium items have size greater than ϵ). These $O(\epsilon^{-1})$ are repacked as above. The last bin affected may find that it has too few medium items and is less than 3/4 full; we address this using $O(\epsilon^{-1})$ worst-case recourse as in our solution for deletion of medium items in a bin.

To summarize, our worst-case recourse is $O(\varepsilon^{-1})$ per item move in the $(1+\varepsilon)$-a.c.r dynamic packing of the large items, $O(\epsilon^{-1})$ per addition or deletion of a medium item and $O(1)$ per item move in the packing of small items. It remains to bound the obtained a.c.r of our packing obtained by extending the α_ε-feasible packing of the small items.

Similarly to the proof of Theorem 2.6, the a.c.r of this algorithm is at most $\alpha + O(\epsilon)$ if no bins are opened due to the medium items. On the other hand, as argued before, this algorithm's a.c.r is at most $4/3 < \alpha + O(\epsilon)$ if such bins are opened. Finally, we observe that the additive term is at most $O(f(\epsilon^{-1}))$, incurred by the packing of the large items and the number of non-huge items stored in designated bins. □

B.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$_\varepsilon$), which we abstract thus:

Definition 2.8 (Bin curve-fitting). *Given a list of bin sizes $0 \leq b_0 \leq b_1 \leq \ldots, b_K \leq 1$ and relative frequencies $f_0, f_1, f_2, \ldots, f_K$, such that $f_x \geq 0$ 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 $s_1, \ldots, s_m \leq 1$ 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., $T = O(\varepsilon^{-1})$. Our algorithm 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 $\Theta(1/\varepsilon)$ *buckets* each. Formally, each clump has T bins, with $f_x \cdot T \in \mathbb{N}$ bins of size $\approx b_x$ for $x = 0, \ldots, K$. The bins in a clump are ordered according to their target b_x, so each clump looks like a 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 6. For each bucket, all bins except those in the last clump are full to within additive ε of their target size.
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 $O(T/\epsilon)$ 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

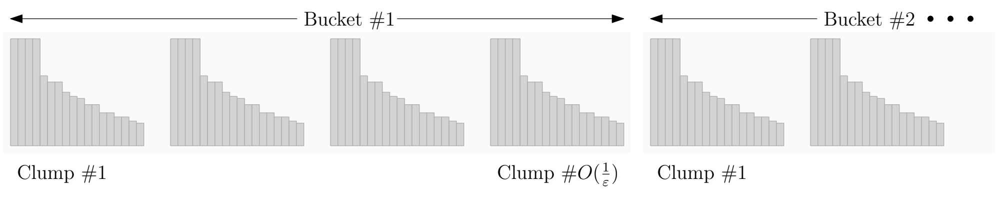


Figure 6: Buckets have $O(1/\varepsilon)$ clumps, clumps have T bins.

that on average the bins are full to within $O(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $O(\varepsilon)$ fraction of bins have no guarantees whatsoever.

As observed in §2, the above approach approach solves bin curve-fitting using $O(T/\varepsilon) = O(\varepsilon^{-2})$ worst-case recourse, provided all items have size at most ε (as in our case). provided all freq We now relate this process to the value of LP_ε. We first show that setting $T = O(1/\varepsilon)$ and restricting to frequencies to 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 ($\mathrm{LP}_\varepsilon$) depends on the total volume B of small items, and f_x may change if B changes, it is convenient to work with a normalized version, referred to as ($\mathrm{LPnew}_\varepsilon$) (see Figure 4) . Now n_x can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$. However, we also need each f_x to be integer multiples of $1/T$ for some integer $T = O(1/\varepsilon)$. We achieve this by slightly modifying the LP solution (which requires a somewhat careful proof).

Lemma 2.9 (Multiples of ε)**.** *For any optimal solution $\{n_x\}$ to ($\mathrm{LPnew}_\varepsilon$) with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + O(\varepsilon)$.*

Proof. For sake of brevity, let $\mathcal{S}'_\varepsilon := \mathcal{S}_\varepsilon \cup \{0\}$. Consider the indices $x \in \mathcal{S}'_\varepsilon$ in increasing order, and modify n_x in this order. Let $\Delta_x := \sum_{x' \in \mathcal{S}'_\varepsilon : x' < x} n_{x'}$, and let $\tilde{\Delta}_x$ be the analogous expression for $\tilde{n}_x$. We maintain the invariant that $|\Delta_x - \tilde{\Delta}_x| \le \varepsilon$, which is trivially true for the base case $x = 0$. Inductively, suppose it is true for x. If $\Delta_x > \tilde{\Delta}_x$, define $\tilde{n}_x$ be n_x rounded up to the nearest multiple of ε, otherwise it is rounded down; this maintains the invariant. If we add $O(\varepsilon)$ to the old α'_ε value, this easily satisfies the second and third set of constraints, since our rounding procedure ensures that the prefix sums are maintained up to additive ε, and $t \in [0, \frac{1}{2}]$. Checking this for the first constraint turns out to be more subtle. We claim that

$$\tilde{n}_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x)\tilde{n}_x \ge 1 - O(\varepsilon).$$

For an element $x \in \mathcal{S}'_\varepsilon$, let Δn_x denote $n_x - \tilde{n}_x$. It is enough to show that $|\sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x| \le O(\varepsilon)$. Indeed,

$$\begin{aligned}\tilde{n}_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x)\tilde{n}_x &= n_0 + \sum_{x \in \mathcal{S}_\varepsilon} (1-x)n_x - \Delta n_0 - \sum_{x \in \mathcal{S}_\varepsilon} (1-x)\Delta n_x \\ &\ge 1 - \sum_{x \in \mathcal{S}'_\varepsilon} \Delta n_x + \sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x \ge 1 - O(\varepsilon) + \sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x.\end{aligned}$$

We proceed to bound $\sum_{x \in \mathcal{S}'_\varepsilon} x \cdot \Delta n_x$. Define $\Delta n_{\le x}$ as $\sum_{x' \in \mathcal{S}'_\varepsilon : x' \le x} \Delta n_{x'}$. Define $\Delta n_{<x}$ analogously. Note that $\Delta n_{\le x}$ stays bounded between $[-\varepsilon, +\varepsilon]$. Let I denote the set of $x \in \mathcal{S}'_\varepsilon$ such that $\Delta n_{\le x}$ changes sign, i.e., $\Delta n_{<x}$ and $\Delta n_{\le x}$ have different signs. We assume w.l.o.g. that $\Delta n_{\le x} = 0$ for any $x \in I$—we can do so by splitting Δn_x into two parts (and so, having two copies of x in $\mathcal{S}'_\varepsilon$). Observe that for any two consecutive $x_1, x_2 \in I$, the function $\Delta n_{\le x}$ is unimodal as x varies from x_1 to x_2, i.e., it has only one local maxima or minima. This is because Δn_x is negative if $\Delta n_{<x}$ is positive, and *vice versa.*

Let the elements in I (sorted in ascending order) be $x_1, x_2, \ldots, x_q$. Let S_i denote the elements in $\mathcal{S}'_\varepsilon$ which lie between x_i and x_{i+1}, where we include x_{i+1} but exclude x_i. Note that $\sum_{x \in S_i} \Delta n_x = \Delta n_{\leq x_{i+1}} - \Delta n_{\leq x_i} = 0$. Let $x'_i = x_i + \varepsilon$ be the smallest element in S_i. Now observe that

$$\sum_{x \in S_i} x \cdot \Delta n_x = x'_i \cdot \sum_{x \in S_i} \Delta n_x + \sum_{x \in S_i} (x - x'_i) \cdot \Delta n_x = \sum_{x \in S_i} (x - x'_i) \cdot \Delta n_x.$$

Because of the unimodal property mentioned above, we get $\sum_{x \in S_i} |\Delta n_x| \leq 2\varepsilon$. Therefore, the absolute value of the above sum is at most $2\varepsilon \cdot (x_{i+1} - x'_i) \leq 2\varepsilon^2 |S_i|$, using that $x_{i+1} - x'_i = (|S_i| - 1)\varepsilon$. Now summing over all S_i we see that $\sum_x x \cdot \Delta n_x \leq O(\varepsilon)$ because $|\mathcal{S}_\varepsilon|$ is $O(1/\varepsilon)$. This proves the desired claim.

So to satisfy the first constraint we can increase n_0 by $O(\varepsilon)$. And then, increasing α'_ε by a further $O(\varepsilon)$ satisfies the remaining constraints, and proves the lemma. $\square$

Let $\tilde{n}_x$ be $i_x \cdot \varepsilon$ where i_x is an integer. Note that $\sum_x \tilde{n}_x \leq \alpha + O(\varepsilon) \leq 2$, so dividing through by ε, $\sum_x i_x \leq 2/\varepsilon$. Now for any index $x \in \{0\} \cup \mathcal{S}_\varepsilon$, we define $f_x := \frac{\tilde{n}_x}{\sum_{x'} \tilde{n}_{x'}} = \frac{i_x}{\sum_{x'} i_{x'}}$. If we set $T := \sum_x i_x \leq 2/\varepsilon$, then T is an integer at most $2/\varepsilon$, and f_x are integral multiples of $1/T$, which satisfies the requirements of our algorithm. Next, we show that the dynamic solution maintained by our algorithm corresponds to a near-optimal solution to LP_ε.

Lemma 2.10 (Small Items Follow the LP)**.** *Let $\varepsilon \leq 1/6$. Using $O(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $O(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + O(\varepsilon))$-feasible packing.*

Proof. The recourse bound is immediate, as each insertion or deletion causes a single item to move from at most $T \cdot 3/\varepsilon$ bins and $T = O(\varepsilon^{-1})$. For the rest of the argument, ignore the last bucket, contributing $O(\varepsilon^{-2})$ bins to our additive term. Let the total volume of items in the other bins be B. Since $\eta = \sum f_x b_x$ is the average bin-size, we expect to use $\approx B/\eta$ bins for these items. We now show that we use at most $(1 + O(\varepsilon)) \cdot \frac{f_x B}{\eta}$ and at least $(1 - O(\varepsilon)) \cdot \frac{f_x B}{\eta}$ bins of size b_x for each x.

Indeed, each (non-last) bucket satisfies the property that all bins in it, except perhaps for those in the last clump, are at least ε-close to the target value. Since each bucket has at least $1/\varepsilon$ clumps, it follows that if there are N clumps and the target average bin-size is η, then $(1 - \varepsilon)N$ clumps are at least $(\eta - \varepsilon)$ full on average. The total volume of a clump is $\eta \cdot T$, so $N \leq \frac{B}{(1-\varepsilon)(\eta - \varepsilon) \cdot T} = \frac{B}{\eta T}(1 + O(\varepsilon))$, where we use that $\eta \geq 1/4$. Therefore, the total number of bins of size b_x used is $f_x T \cdot N \leq (1 + O(\varepsilon)) \cdot \frac{B f_x}{\eta}$. The lower bound for the number of bins of size b_x follows from a similar argument and the observation that if we scale the volume of small items up by a factor of $(1 + O(\varepsilon))$, this volume would cause each bin to be filled to its target value. This implies that we use at least $(1 - O(\varepsilon)) \cdot \frac{B f_x}{\eta}$ bins with size b_x.

We now show that the $\bar{N}_x$ satisfy LPnew_ε with $\alpha^\star_\epsilon + O(\epsilon)$. Recall that we started with an optimal solution to LPnew_ε of value $\alpha^\star_\varepsilon + O(\varepsilon)$, used Lemma 2.9 to get $f_x = \tilde{n_x} / \sum_{x'} \tilde{n_{x'}}$ and ran the algorithm above. By the computations above, $\bar{N}_x$, the number of bins of size b_x used by our algorithm, is

$$(1 + O(\varepsilon)) \cdot \frac{f_x B}{\sum_{x'} f_{x'} b_{x'}} = (1 + O(\varepsilon)) \cdot \frac{\tilde{n}_x B}{\sum_{x'} \tilde{n}_{x'} b_{x'}} \leq (1 + O(\varepsilon)) \cdot \tilde{n}_x B,$$

where the last inequality follows from the fact that $\sum_{x'} \tilde{n}_{x'} b_{x'} \geq 1$ (by the first constraint of LPnew_ε). Likewise, by the same argument, we find that these $\bar{N}_x$ satisfy CR_ε with $\alpha_\epsilon = \alpha^\star_\varepsilon + O(\epsilon)$. Finally, since $\tilde{n}_x$ satisfies the constraints of LPnew_ε (up to additive $O(\varepsilon)$ changes in α_ε), we can verify that the quantities $\bar{N}_x$ satisfy the last two constraints of LP_ε (again up to additive $O(\varepsilon)$ changes in α_ε). To see that they also satisfy Vol_ε, we use the following calculation:

$$\sum_x \bar{N}_x \geq (1 - 3\varepsilon) \cdot \sum_x \frac{f_x B}{\sum_x b_x f_x} = (1 - O(\varepsilon)) \cdot \frac{B}{\sum_x b_x f_x} \geq (1 - O(\varepsilon)) \cdot B,$$

because $\sum_x f_x = 1$ and $b_x \leq 1$ for all x. Therefore, scaling all variables with $(1 - O(\varepsilon))$ will satisfy constraint Vol_ε as well. It follows that $\bar{N}_x$ satisfy LP_ε with $\alpha_\varepsilon = \alpha^\star_\varepsilon + O(\varepsilon)$. □

B.2.3 Our Algorithm

Here we provide missing proofs of some of the claims made in our algorithm's analysis in §2.2.3.

Amortized Algorithm: In §2.2.3 we claimed that if we start an epoch with a packing using $N \leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins, as we do (here $\mathcal{I}_t$ is the instance at the beginning of the epoch), then if for $\varepsilon \cdot N$ updates we address updates naïvely, our solution at any given time t' during the epoch uses at most $N_{t'} \leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon^{-2})$ bins. To see this, note that as each update can only change $OPT(\mathcal{I}_{t'})$ and the number of bins we use, $N_{t'}$, by one. Therefore, $N \cdot (1 - \varepsilon) \leq N_{t'} \leq N \cdot (1 + \varepsilon)$ and $OPT(\mathcal{I}'_t) \geq OPT(\mathcal{I}) + \varepsilon \cdot N$. By our upper bound on N, we obtain

$$\begin{aligned}
N_{t'} &\leq (1 + \varepsilon) \cdot N \\
&\leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_t) + \varepsilon \cdot N + O(\varepsilon^{-2}) \\
&\leq (\alpha + O(\varepsilon)) \cdot (OPT(\mathcal{I}_{t'}) + \varepsilon \cdot N) + \varepsilon \cdot N + O(\varepsilon^{-2}) \\
&= (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon \cdot N) + O(\varepsilon^{-2}) \\
&\leq (\alpha + O(\varepsilon)) \cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon \cdot N_{t'}) + O(\varepsilon^{-2}),
\end{aligned}$$

where we used in the last step the fact that $N \leq N_{t'}/(1 - \varepsilon)$. Subtracting the $O(\varepsilon \cdot N_{t'})$ term from both sides of the above and dividing through by $(1 - O(\varepsilon))$, we obtain the claimed bound.

C Omitted Proofs of Section 3 (General Movement Costs)

Here we provide proofs for our general amortized recourse upper and lower bounds, and discuss cases for which the algorithmic bounds can be made worst case (in §C.2.5).

C.1 Matching the Lower Bounds for Online Algorithms

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.*

Proof of Theorem 3.1. Take the adversary process $\mathcal{B}$, and use it to generate an instance for the fully-dynamic algorithm $\mathcal{A}$ as follows. (When there are k items in the system, let them be labeled $e_1, e_2, \ldots, e_k$.)

I: Given system in a state with k elements, use $\mathcal{B}$ to generate the next element e_{k+1}, having movement cost $(2\beta)^{-(k+1)}$.

II: If $\mathcal{A}$ places e_{k+1} into some bin and does not move any other item, go back to Step I to generate the next element.

III: However, if $\mathcal{A}$ moves some items, and e_j is the item with the smallest index that is repacked, delete items $e_{j+1}, \ldots, e_{k+1}$. This may in turn cause elements to be repacked, so delete all items with indices strictly higher than than the smallest index item that is repacked. Eventually we stop at a state with $k' \geq 1$ elements such that only element $e_{k'}$ has been repacked. Now go back to Step I. (Also, $e_{k'+1}, \ldots, e_{k+1}$ are deemed undefined.)

Since the location of each item e_i is based only on the knowledge of prior elements in the sequence $e_1, e_2, \ldots, e_{i-1}$ and their bins, the resulting algorithm is another online algorithm. So if the length of the sequence eventually goes to infinity, we are guaranteed to reach an instance for which the a.c.r of this algorithm will be at least c. Hence we want to show that for any n, the length of the sequence eventually reaches n (or the adversary process stops, having showed a lower bound of c). Consider a potential function Φ which is zero when the system has no elements. When a new element is added by $\mathcal{B}$, we increase Φ by β times the movement cost for this element. Moreover, when $\mathcal{A}$ moves elements, we subtract the movement costs of these elements from Φ. Since $\mathcal{A}$ ensures an amortized recourse bound of β, the potential must remain non-negative.

For a contradiction, suppose the length of the sequence remains bounded by n. Hence, there is some length $k < n$ such that Step III causes the sequence to become of length k arbitrarily often. Note that the total increase in Φ between two such events is at most $\beta \sum_{i=k+1}^{n} (2\beta)^{-i} \leq \frac{(2\beta)^{-(k+1)}}{2\beta - 1}$. Since $\beta \geq 2$, this increase is strictly less than $(2\beta)^{-k}$, the total decrease in Φ due to the movement of element k alone. Since the potential decreases between two such events, there can only be finitely many such events, so the length of the sequence, i.e., the number of items in the system increases over time, eventually giving us the claimed lower bound. □

C.2 (Nearly) Matching the Upper Bounds for Online Algorithms

We start by proving our lemma for packing 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.)*

Proof. We round down movement costs of each item to the next-lower power of two. We maintain all items sorted by movement cost, with the costliest items in the first bin. All bins (except perhaps the last bin) contain $k-1$ items; if all items have size $1/k$ then bins contain k items. Insertion and deletion of an item of cost $c_i = 2^\ell$ can be assumed to be performed at the last bin containing an item of this cost (possibly incurring an extra movement cost of c_i, by replacing item i with another item of cost c_i). If addition or deletion leaves a bin with one item too many or too few, we move a single item to/from the last bin containing an item of the next lower cost, and so on. Since items are sorted and the costs are powers of 2, the total movement cost is at most $c_i + c_i \cdot (1 + 1/2 + 1/4 + \ldots) = 3 \cdot c_i$; the loss due to rounding means the (worst case) recourse is ≤ 6. The lemma follows. □

C.2.1 A Simple 2-approximate Solution

Suppose we round all item sizes to powers of 2 and use Lemma 3.3 on items of size at least $1/n$, where n is the maximum number of items in the instance $\mathcal{I}_t$ over all times t, while packing all items of size less than $1/n$ into a single bin.[2] Then, we ensure that all but $1 + \log_2 n$ bins are full. This approach can be applied to general instances by rounding up item sizes to powers of two, yields the following fact.

Fact C.1. *There exists a fully-dynamic* BIN PACKING *algorithm with a.c.r 2 and additive term $1 + \log_2 n$, using constant worst case recourse.*

[2] We assume we know n. If n is unknown, we can obtain the same amortized recourse by a simple "guess and double/halve" approach.

However, this simple result is highly unsatisfactory for two reasons. Firstly, as we shall show, its a.c.r is suboptimal. The second reason concerns its additive term, which can be blown up to be arbitrarily large without effecting the optimal solution in any significant way, by adding N items of size $1/N$ for arbitrarily large N. In what follows, we will aim to design algorithms with better a.c.r and both additive term and recourse independent of n.

C.2.2 The Super Harmonic family of online algorithms

A *Super-harmonic* (abbreviated as SH) algorithm consists of a partition of the unit interval $[0,1]$ into $K+1$ intervals, $[0,\varepsilon],(t_0=\varepsilon,t_1](t_1,t_2],\dots,(t_{K-1},t_K=1]$. Small items (i.e., items of size at most ε) are packed using FirstFit into dedicated bins; because the items are small, all but one of these are at least $1-\varepsilon$ full. For larger items, each arriving item is of type i if its size is in the range $(t_{i-1},t_i]$. Items are colored either blue or red by the algorithm, with each bin containing items of at most two distinct item types i and j. If a bin contains only one item type, its items are all colored the same, and if a bin contains two item types i and j, then all items of type i are colored blue and items of type j are colored red (or vice versa). The SH algorithm has associated with it three number sequences $(\alpha_i)_{i=1}^K,(\beta_i)_{i=1}^K,(\gamma_i)_{i=1}^K$, and an underlying bipartite *compatibility graph* $\mathcal{G}=(V,E)$ whose role will be made clear shortly. 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 β_i type i (resp., less than γ_j type j items). The compatibility graph determines which pair of (colored) item types can share a bin. The compatibility graph $\mathcal{G}=(V,E)$ is defined on the vertex set $V=\{b_i\mid i\in[K]\}\cup\{r_i\mid j\in[K]\}$, with an edge $(b_i,r_j)\in E$ indicating blue items of type i and red items of type j are *compatible*; they are allowed to be placed in a common bin. Apart from the above properties, an SH algorithm must satisfy the following invariants.

(P1) The number of open bins is $O(1)$.

(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 nothing but blue type i items and one containing nothing but red type j items.

The above invariants allow one to bound the asymptotic competitive ratio of an SH algorithm, depending on the choices of $(\alpha_i)_{i=1}^K,(\beta_i)_{i=1}^K,(\gamma_i)_{i=1}^K$ and the compatibility graph $\mathcal{G}$. In particular, Seiden [24] showed the following.

Lemma C.2 (Seiden [24])**.** *There exists an SH algorithm with a.c.r* 1.589.

A particular property the SH algorithm implied by Lemma C.2 which we will make use of later is that its parameters or inverses are all at most a constant; in particular, $\varepsilon^{-1}=O(1)$ and $K=O(1)$, and similarly $\beta_i,\gamma_i=O(1)$ for all $i\in[K]$.

In the following sections we proceed to describe how to maintain the above invariants that suffice to bound the competitive ratio of an SH algorithm. We start by addressing the problem of maintaining the SH invariants for the large items, in Section C.2.3. We then show how to pack small items (i.e., items of size at most ε) into bins such that all but a constant of these bins are at least $1-\varepsilon$ full, in Section C.2.4.[3] Finally we conclude with our upper bound in Section C.2.6.

[3]Strictly speaking, we will only pack small items into bins which are $1-\epsilon$ full *on average.* However, redistributing these small items or portions of these items within these bins will make these bins $1-\epsilon$ full without changing the number of bins used by the solution. The bounds on SH algorithms therefore carries through.

C.2.3 SH Algorithms: Dealing with Large Items

First, we round all movement costs to powers of 2, increasing our recourse cost by at most a factor of 2. Now, our algorithm will have recourse cost which will be some function of $K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$; as for our usage, we have $K = O(1)$, and $\beta_i, \gamma_i = O(1)$ for all $i \in [K]$, we will simplify notation and assume that these values are indeed all bounded from above by a constant. Consequently, when moving around groups of up to β_i blue (resp. γ_i red) type i items whose highest movement cost is c, the overall movement cost will be $O(c)$. The stipulation that $K = O(1)$ will prove useful shortly. We now explain how we maintain the invariants of SH algorithms.

Satisfying Property (P1). We keep all blue items (resp. red items) of type i in bins containing up to β_i items (resp. γ_i items). We sort all bins containing type-i items by the cost of the costliest type-i item in the bin, where only the last bin containing type-i items contains less than β_i blue (alternatively, γ_i red) type-i items. Therefore, if we succeed in maintaining the above, the number of open bins is $O(1)$; i.e., Property (P1) is satisfied. We explain below how to maintain this property together with Properties (P2) and (P3).

Satisfying Property (P2). We will only satisfy Property (P2) *approximately*, such that the number of red type-i items is in the range $[\lfloor \alpha_i \cdot n_i \rfloor, \lfloor \alpha_i \cdot n_i \rfloor + \delta_i]$, where $\delta_i = \max\{\alpha_i(\beta_i - \gamma_i) + 1 + \gamma_i, 0\}$. Removing these at most δ_i red type-i items results in a smaller bin packing instance, and a solution requiring the same number of bins, satisfying the invariants of SH algorithms. Notice that this change to the solution can change the number of open bins by at most $\sum_i \delta_i \leq \sum_i \alpha_i \beta_i + K = O(1)$. Therefore, satisfying Property (P2) approximately suffices to obtain our sought-after a.c.r. In fact, we satisfy a stronger property: each prefix of bins containing type-i items has a number of red type-i items in the range $[\lfloor \alpha_i \cdot n'_i \rfloor, \lfloor \alpha_i \cdot n'_i \rfloor + \delta_i]$, where n'_i is the number of type-i items in the prefix.

Maintaining the above prefix invariant on deletion is simple enough: when removing some type-i item of cost 2^k, we move the last type-i item of the next movement cost to this item's place in the packing, continuing until we reach a type-i item with no cheaper type-i items. The movement cost here is at most $2^k + 2^{k-1} + 2^{k-2} + \cdots = O(2^k)$, so the (worst case) recourse is constant. As the prefix invariant was satisfied before deletion, it is also satisfied after deletion, as we effectively only remove an item from the last bin. When inserting a type-i item, we insert the item into the last appropriate bin according to the item's movement cost. This might cause this bin to overflow, in which case we take the cheapest item in this bin and move it into the last appropriate bin according to this item's cost, continuing in this fashion until we reach an open bin, or are forced to open a new bin. (The recourse here is a constant, too). The choice of color for type-i items in a newly-opened bin depends on the number of type-i items before this insertion, n_i, and the number of red type-i items before this insertion, m_i. If $m_i + \gamma_i \leq \lfloor \alpha_i(n_i + \gamma_i) \rfloor + \delta_i$, the new bin's red items are colored red. By this condition, the number of red items for the following γ_i insertions into this bin will satisfy our prefix property. If the condition is not satisfied, the bin is colored blue. Now, as

$$\begin{aligned} m_i + \gamma_i &> \lfloor \alpha_i(n_i + \gamma_i) \rfloor + \delta_i \\ &\geq \lfloor \alpha_i(n_i + \beta_i) \rfloor + \alpha_i(\gamma_i - \beta_i) - 1 + \delta_i \\ &\geq \lfloor \alpha_i(n_i + \beta_i) \rfloor - 1 + \gamma_i \end{aligned}$$

we find that after this new bin contains β_i type i items, the number of red items in the prefix is m_i while the number of type-i items is $n'_i = n_i + \beta_i$, and so this prefix too satisfies the prefix condition.

We conclude that the above methods approximately maintain Property (P2) (as well as Property (P1)) while only incurring constant worst case recourse.

Satisfying Property (P3). Finally, in order to satisfy Property (P3), we consider the groups of up to β_i and γ_j blue and red items of type i and j packed in the same bin as nodes in a bipartite graph. A blue type i (resp. red type j) group is a copy of node b_i (resp. r_j) in the compatibility graph $\mathcal{G}$, and copies of nodes b_i and r_j are connected if they are connected in $\mathcal{G}$. If a particular node b_i and r_j are placed in the same bin, then we treat that edge as matched. Each node has a cost which is simply the maximum movement cost of an item in the group which the node represents. All nodes have a preference order over their neighbors, preferring a costlier neighbor, while breaking ties consistently. We will maintain a stable matching in this subgraph, where two nodes are matched if the items they represent. This stability clearly implies Property (P3). We now proceed to describe how to maintain this bipartite graph along with a stable matching in it.

The underlying operations we will have is addition and removal of a node of red or blue type i items of cost 2^k; i.e., with costliest item having movement cost 2^k. As argued before, as each group contains $O(1)$ items, the movement cost of moving items of such a group is $O(2^k)$. Using this operation we can implement insertion and deletion of single items, by changing the movement cost of groups with items moved, implemented by removal of a group and re-insertion with a higher/lower cost if the cost changes (or simple insertion/removal for a new bin opened/bin closed). As argued above, the cost of items moved is during updates in order to satisfy Property (P2) is $O(2^k)$, where 2^k is the cost of the item added/removed; consequently, the costs of removals and insertions of groups is $O(2^k)$. We therefore need to show that the cost of insertion/removal of a group of cost 2^k is $O(2^k)$.

Insertions and deletions of blue and red nodes is symmetric, so we consider insertions and deletions of a blue node b only. Upon insertion of some node b of cost 2^k, we insert b into its place in the ordering, and scan its neighbors for the first neighbor r which strictly prefers b to its current match, b'. If no such r exists, we are done. If such an r exists, b' is unmatched from r (its items are removed from its bin) and b is matched to r (the items of b are placed in the same bin as r's bin). As b' has strictly less than b, its cost is at most 2^{k-1}. We now proceed similarly for b' as though we inserted b' into the graph. The overall movement cost is at most $2^k + 2^{k-1} + 2^{k-2} + \cdots = O(2^k)$.

Upon deletion of a blue node b of cost 2^k, if it had no previous match, we are done. If b did have a match r, this match scans its neighbors, starting at b, for its first neighbor b' of cost at most 2^k which prefers r to its current match. If the cost of b' is 2^k, we match r to the last blue node of the same type as b', denoted by b''. If b'' was previously matched, we proceed to match its match as if b'' were removed (i.e., as above). As every movement decreases the number of types of blue nodes of a given cost to consider, the overall movement cost is at most $K \cdot (2^k + 2^{k-1} + 2^{k-2} + \dots) = O(K \cdot 2^k) = O(2^k)$, where here we rely the number of types being $K = O(1)$.

We conclude that Property (P3) can be maintained using constant worst case recourse.

Lemma C.3. *Properties (P1), (P2) and (P3) can be maintained using constant worst-case recourse.*

C.2.4 SH Algorithms: Dealing with Small Items

Here we address the problem of packing small items into bins so that all but a constant number of bins are kept at least $1-\varepsilon$ full. Lemma 3.3 allows us to do just this for items of size in the range $[\varepsilon', \varepsilon]$, for $\varepsilon' = \Omega(\varepsilon)$. Specifically, considering all integer values c in the range $[\lceil \frac{1}{\varepsilon} \rceil, \lceil \frac{1}{\varepsilon'} \rceil]$, then, as $\varepsilon^{-1} = O(1)$ and consequently $\lceil \frac{1}{\varepsilon'} \rceil - \lceil \frac{1}{\varepsilon} \rceil = O(1)$, we obtain the following.

Corollary C.4. *All items in the range $[\varepsilon', \varepsilon]$ for any $\varepsilon' = \Omega(\varepsilon)$ can be packed into bins which are all (barring perhaps $O(\epsilon^{-1}) = O(1)$ bins) at least $1 - \varepsilon$ full.*

It now remains to address the problem of efficiently maintaining a packing of items of size at most ε' into bins which are at least $1-\varepsilon$ full (again, up to some $O(1)$ possible additive term). As $\varepsilon' = \Omega(\varepsilon)$, we will attempt to pack these items into bins which are at least $1 - O(\varepsilon')$ full on average. For notational simplicity from here on, we will abuse notation and denote ε' by ε, contending ourselves with a packing which is $1 + O(\varepsilon)$-competitive, and is therefore keeps bins $1 - O(\varepsilon)$ full on average (as OPT is close to the volume bound for instance made of only small items).

Let us first give the high-level idea of the algorithm before presenting the formal details. Define the *density* of an item as c_j/s_j, the ratio of its movement cost to its size. We arrange the bins in some fixed order, and the items in each bin will also be arranged in the order of decreasing density. This means the total order on the items (consider items in the order dictated by the ordering of bins, and then by the ordering within each bin) is in decreasing order of density as well. Besides this, we want all bins, except perhaps the last bin, to be approximately full (say, at least $1 - O(\varepsilon)$ full). The latter property will trivially guarantee $(1 + O(\varepsilon))$-competitive ratio. When we insert an item j, we place it in the correct bin according to its density. If this bin overflows (i.e., items in it have total size more than 1), then we remove some items from this bin (the ones with least density) and transfer them to the next bin – these items will have only smaller density than j, and so, their movement cost will be comparable to that of j. If the next bin can accommodate these items, then we can stop the process, otherwise this could lead to a long cascade. To prevent such long cascades, we arrange the bins in *buckets* – each bucket consists of about $O(\varepsilon^{-1})$ consecutive bins, and all these buckets are approximately full except for the last bin in the bucket. Again, it is easy to see that this property will ensure $(1 + O(\varepsilon))$-competitive ratio. Note that this extends the idea of Berndt et al. Once we have these buckets, the above-mentioned cascade stops when we reach the last bin of a bucket. Consequently we have cascades of length at most $O(\varepsilon^{-1})$. If the last bin also overflows, we will add another bin at the end of this bucket, and if the bucket now gets too many bins, we will split it into two smaller buckets. One proceeds similarly for the case of deletes – if an item is deleted, we *borrow* some items from the next bin in the bucket, and again this could cascade only until the last bin in the bucket. (If the bucket ever has too few bins, we merge it with the next bucket).

However, this cascade is not the only issue. Because items are atomic and have varying sizes, it is possible that insertion of a tiny item (say of size $O(\varepsilon^2)$) could lead us to move items of size $O(\varepsilon)$. In this case, even though the density of the latter item is smaller than the inserted item, its total movement cost could be much higher. To prevent this, we ensure that whenever a bin overflows, we move out enough items from it so that it has $\Omega(\varepsilon)$ empty space. Now, the above situation will not happen unless we see tiny items amounting to a total size of $\Omega(\varepsilon)$. In such a case, we can charge the movement cost of the larger item to the movement cost of all such tiny items.

The situation with item deletes is similar. When a tiny item is deleted, it is possible that the corresponding bin underflows, and the item borrowed from the next bin is large (i.e., has size about ε). Again, we cannot bound the movement cost of this large item in terms of that of the item being deleted. To take care of such issues, we do not immediately remove such tiny items from the bin. We call such items *ghost* items – they have been deleted, but we have not removed them from the bins containing them. When a bin accumulates ghost items of total size about $\Omega(\varepsilon)$, we can afford to remove all these from the bin, and the total movement cost of such items can pay for borrowing items (whose total size would be $O(\varepsilon)$) from the next bin.

The analysis of the movement cost is done via by a potential function argument, to show the following result (whose proof appears in Section C.2.5):

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

We first describe the algorithm formally. Let B_i denote the items stored in a bin i. As mentioned above, our algorithm maintains a solution in which items are stored in decreasing order of density.

I.e., for all $i < i'$, for every pair of jobs $j \in B_i$ and $j' \in B_{i'}$ we will have $c_j/v_j \geq c_{j'}/v_{j'}$. Recall that B_i could contain ghost jobs. Let A_i denote the jobs in B_i which have not been deleted yet (i.e., are not ghost jobs), and G_i denote the ghost jobs. We shall use $s(B_i)$ to denote the total size of items in B_i (define $s(A_i)$ similarly). We maintain the following invariants, satisfied by all bins B_i that are not the last bin in their bucket:

$$\begin{aligned} &\mathrm{P}_0 : 1 - 3\varepsilon \leq s(B_i) \leq 1. \\ &\mathrm{P}_1 : s(A_i) \geq 1 - 4\varepsilon. \end{aligned} \tag{3}$$

Finally, a bin B_i which is the last bin in its bucket has no ghost jobs. That is, $s(G_i) = 0$. Each bucket has at most $3/\varepsilon$ bins. Furthermore, each bucket, except perhaps for the last bucket, has at least ε^{-1} bins.

Our algorithm is given below. We use two functions GrowBucket(U) and SplitBucket(U) in these procedures, where U is a bucket. The first function is called when the bucket U *underflows*, i.e., when U has less than ε^{-1} bins. If U is the last bucket, then we need not do anything. Otherwise, let U' be the bucket following U. We merge U and U' into one bucket (note that the last bin of U need not satisfy the invariant conditions above, and so, we will need to do additional processing to ensure that the conditions are satisfied for this bin). The function SplitBucket(U) is called when U contains more than $3/\varepsilon$ bins. In this case, we split it into two buckets, each of size more than ε^{-1}.

Algorithm INSERT(j)

```
1: Add job $j$ into appropriate bin $i$
2: if $s(B_i) > 1$ then
3:    EraseGhost($i, s_j$)
4: end if
5: if $s(B_i) > 1$ then
6:    OVERFLOW($i$, $s(B_i) - 1 + 2\varepsilon$)
7: end if
```

Algorithm DELETE(j)

```
1: Let $i$ be the bin containing $j$
2: if $i$ is the last bin in its bucket then
3:    Erase $j$ from $i$
4: else
5:    Mark $j$ as a ghost job.
6:    if $s(A_i) < 1 - 4\varepsilon$ then
7:       Erase all ghost jobs from bin $i$
8:       BORROW($i$, $1 - 3\varepsilon - s(A_i)$)
9:    end if
10: end if
```

Algorithm OVERFLOW(i, v)

```
1:  Let $X$ be the minimum density jobs in
2:      $B_i$ s.t. $s(X) \geq v$
3:  if $i$ is the last bin its bucket or
4:        $v \geq 1 - 3\varepsilon$ then
5:     Add a new bin $i'$ after $i$ in this bucket
6:     Move $X$ from $i$ to $i'$.
7:     Let $U$ be the bucket containing $i$.
8:     if $U$ has more than $3/\varepsilon$ bins then
9:        SplitBucket($U$)
10:    end if
11: else
12:    Move $X$ from bin $i$ to bin $i+1$
13:    if $s(B_{i+1}) > 1 - \varepsilon$ then
14:       EraseGhost($i+1, s(B_{i+1}) - 1 + 2\varepsilon$)
15:    end if
16:    if $s(B_{i+1}) > 1 - \varepsilon$ then
17:       OVERFLOW($i+1, s(B_{i+1}) - 1 + 2\varepsilon$)
18:    end if
19: end if
```

Algorithm BORROW(i, v)

```
1:  Let $X$ be the minimum density jobs in $B_{i+1}$
    s.t. $s(X \cap A_{i+1}) \geq \min(v, s(A_{i+1}))$
2:  Remove $X$ from $B_{i+1}$ (erase ghosts in $X$)
3:  Move $A \triangleq X \cap A_{i+1}$ to $B_i$
4:  if bin $i+1$ is empty then
5:     Remove this bin from its bucket $U$
6:     if $U$ has less than $\varepsilon^{-1}$ bins and
7:           it is not the last bucket then
8:        GrowBucket($U$)
9:        if $U$ has more than $3/\varepsilon$ bins then
10:          SplitBucket($U$)
11:       end if
12:    end if
13:    if $s(A_i) < 1 - 3\varepsilon$ then
14:       BORROW($i$, $1 - s(A_i) - 3\varepsilon$)
15:    end if
16: else
17:    if $s(A_{i+1}) < 1 - 3\varepsilon$ then
18:       Erase all ghost jobs from bin $i+1$
19:       BORROW($i+1$, $1 - 3\varepsilon - s(A_{i+1})$)
20:    end if
21: end if
```

We first describe the function INSERT(j) for an item j. We insert job j into the appropriate bin i (recall that the items are ordered by their densities). If this bin overflows, then we call the procedure EraseGhost(i, s_j). The procedure EraseGHost(i, s), where i is a bin and s is a positive quantity, starts erasing ghost jobs from B_i till one of the following events happen: (i) B_i has no ghost jobs, or (ii) total size of ghost jobs removed exceeds s. Since all jobs are of size at most ε, this implies that the total size of ghost jobs removed is at most $\min(s(G_i), s + \varepsilon)$. Now its possible that even after removing these jobs, the bin overflows (this will happen only if $s(G_i)$ was at most s_j). In this case, we offload some of the items (of lowest density) to the next bin in the bucket. Recall that when we do this, we would like to create $O(\varepsilon)$ empty space in bin i. So we transfer jobs of least density in B_i of total size at least $s(B_i) - 1 + 2\varepsilon$ to the next bin (since all jobs are small, the empty space in bin i will be in the range $[2\varepsilon, 3\varepsilon]$) . This is done by calling the procedure OVERFLOW(i, $s(B_i) - 1 + 2\varepsilon$). The procedure OVERFLOW(i, v), where i is a bin and v is a positive quantity, first builds a set X of items as follows – consider items in B_i in increasing order of density, and keep adding them to X till the total size of X, denoted by $s(X)$, exceeds v (so, the total size of X is at most $v + \varepsilon$). We now transfer X to the next bin in the bucket (note that by construction, X does not have any ghost jobs). The same process repeats at this bin (although we will say that overflow occurs at this bin if $s(B_i)$ exceeds $1 - \varepsilon$). This cascade can end in several ways – it is not difficult to show that between two consecutive calls to OVERFLOW, the parameter v grows by at most ε. If v becomes larger than $1 - 3\varepsilon$, we just create a new bin and assign all of X to this new bin. If we reach the last bin, we again create a new bin and add X to it. In both these cases, the size of the bucket increases by 1, and so we may need to split this bucket. Finally, it is possible that even after transfer of X, $s(A_i)$ does not exceed $1 - \varepsilon$ – we can stop the cascade at this point.

Next we consider the case of deletion of an item j. Let i be the bin containing j. If i is the last bin in its bucket, then we simply remove j (recall that the last bin in a bucket cannot contain ghost jobs). Otherwise, we mark j as a ghost job. This does not change $s(B_i)$, but could decrease $s(A_i)$. If it violates property P_1, we borrow enough items from the next bin such that i has free space of about 2ε only. The function BORROW(i, v), where i is a bin and v is a positive quantity, borrows densest (non-ghost) items from the next bin $i+1$ of total size at least v. So it orders the (non-ghost) items in $i+1$ in decreasing density, and picks them till the total size accumulated is at least v. This process may cascade (and will stop before we reach the last bin). However there is one subtlety – between two consecutive calls to BORROW, the value of the parameter v may grow (by up to ε), and so, it is possible that bin $i+1$ becomes empty, and we are not able to transfer enough items from $i+1$ to i. In this case, we first remove $i+1$, and continue the process (if the size of the bucket becomes too small, we handle this case by merging it with the next bin and splitting the resulting bucket if needed). Since we did not move enough items to i, we may need to call BORROW(i, v') again with suitable value of v'. There is a worry that the function BORROW(i, v) may call BORROW(i, v') with the same bin i, and so, whether this will terminate. But note that, whenever this case happens, we delete one bin, and so, this process will eventually terminate.

C.2.5 Analysis

We begin by showing properties of the OVERFLOW and the BORROW functions.

Lemma C.5. *Whenever* OVERFLOW*(i, v) is called, bin i has no ghost jobs. Furthermore, $s(B_i) = v + 1 - 2\varepsilon$. Finally, when* OVERFLOW*(i, v) ends, $1 - 3\varepsilon \leq s(B_i) \leq 1 - 2\varepsilon$, and $s(G_i) = 0$.*

Similarly, whenever BORROW*(i, v) is called, bin i has no ghost jobs. Furthermore, $s(B_i) = v + 1 - 3\varepsilon$. Finally, when* BORROW*(i, v) ends, either i is the last bin in its bucket or $1 - 3\varepsilon \leq s(B_i) \leq 1 - 2\varepsilon$, and $s(G_i) = 0$.*

Proof. When we insert an item j in a bin i, we erase ghost items from i till either (i) we erase all ghost items, or (ii) we erase ghost items of total size at least s_j. If the second case happens, then the fact that $s(B_i) \leq 1$ before insertion of item j implies that we will not call OVERFLOW in Line 3 of INSERT(j). Therefore, if we do call OVERFLOW, it must be the case that we ended up deleting all ghost jobs in i. Further we call OVERFLOW with $v = s(B_i) - 1 + 2\varepsilon$. Similarly, before we call OVERFLOW($i+1, v$) in Line 17, we try to ensure ghost jobs of the same volume v from bin $(i+1)$. If we indeed manage to remove ghost jobs of total size at least v, we will not make a recursive call to OVERFLOW. This proves the first part of the lemma.

When OVERFLOW(i, v), terminates, we make sure that we have transferred the densest v volume out of i to the next bin. Since job sizes are at most ε, we will transfer at most items of total size in the range $[v, v + \varepsilon]$ out of i. Since $s(B_i) = v + 1 - 2\varepsilon$ before calling this function, it follows that $s(B_i)$ after end of this function lies in the range $[1 - 3\varepsilon, 1 - 2\varepsilon]$. The claim for the BORROW borrow function follows similarly. □

The following corollary follows immediately from the above lemma.

Corollary C.6. *Properties P_0 and P_1 from (3) are satisfied throughout by all bins which are not the last bins in their bucket. All bins B_i which are last in their bucket satisfy $s(G_i) = 0$.*

We are now ready to prove the claimed bounds obtained by our algorithm for small items.

Proof of Lemma 3.4. First we bound the competitive ratio. Barring the last bucket, which has $O(\varepsilon^{-1})$ bins, all other buckets have at least ε^{-1} bins. All bins (except perhaps for the last bin) in each of these buckets have at most 4ε space that is empty or is filled by ghost jobs. Therefore, the competitive ratio is $(1 + O(\varepsilon))$ with an additive $O(\varepsilon^{-1})$ bins.

It remains to bound the recourse cost of the algorithm. For a solution $\mathcal{S}$, define the potential:

$$\Phi(\mathcal{S}) = \frac{4}{\varepsilon^2} \sum_{\text{bins } i} \Phi(i), \qquad \text{where}$$

$$\Phi(i) = c(G_i) + c(\text{fractional sparsest items above } 1 - \varepsilon \text{ in } B_i).$$

The second term in the definition of $\Phi(i)$ is evaluated as follows: arrange the items in B_i in decreasing order of density, and consider the items occupying the last ε space in bin i (the total size of these items could be less than ε if the bin is not completely full). Observe that at most one item may be counted fractionally here. The second term is simply the sum of the movement costs of all such item, where a fractional item contributes the appropriate fraction of its movement cost.

Now we bound the amortized movement cost with respect to potential Φ. First consider the case when we insert item j in bin i. This could raise Φ_i by c_j. If bin i does not overflow, we do not pay any movement cost. Further, deletion of ghost jobs from bin i can only decrease the potential. Therefore, the amortized movement cost is bounded by c_j. On the other hand, suppose bin i overflows and this results in calling the function OVERFLOW(i, v) with a suitable v. Before this function call, let I denote the set of items which are among the sparsest items above $(1 - \varepsilon)$ volume in bin i (i.e., these contribute towards Φ_i). Let d be the density of the least density item in I. Since $s(B_i) \geq 1$, it follows that $\Phi_i \geq d\varepsilon$. When this procedure ends, Lemma C.5 shows that $s(B_i) \leq 1 - \varepsilon$, and so, Φ_i would be 0. Thus, Φ_i decreases by at least $d\varepsilon - c_j$. Lemma C.5 also shows that if we had recursively called OVERFLOW(i', v') for any other bin i', then $s(B_{i'})$ would be at most $1 - \varepsilon$, and so, $\Phi_{i'}$ would be 0 when this process ends. It follows that the overall potential function Φ decreases by at least $4(d\varepsilon - c_j)/\varepsilon^2$. Let us now estimate the total movement cost. We transfer items of total size at most 1 from one bin to another. This process will clearly end when we reach the end of the bucket, and so the total size of items moved during this process is at most $3/\varepsilon$, i.e., the number of bins in this bucket. The density of these items is at most d, and so the total movement cost is at most $3d/\varepsilon$. Thus, the amortized movement cost is at most c_j/ε^2.

Now we consider deletion of an item j which is stored in bin i. Again the interesting case is when this leads to calling the function BORROW. Before j was deleted, $s(B_i)$ was at least $1 - 3\varepsilon$ (property P_0). After we mark j as a ghost job, $s(A_i)$ drops below $1 - 4\varepsilon$. So the total size of ghost jobs is at least ε. Since we are removing all these jobs from bin i, Φ_i decreases by at least $d\varepsilon$, where d is the density of the least density ghost job in i. As in the case of insert, Lemma C.5 shows that whenever we make a function call BORROW(i', v'), $\Phi_{i'}$ becomes 0 when this process ends. So, the potential Φ decreases by at least $4d/\varepsilon$. Now, we count the total movement cost. We transfer items of size at most 1 between two bins, and so, we just need to count how many bins are affected during this process (note that if we make several calls to BORROW with the same bin i', the the *total* size of items transferred to i' is at most 1). Let U be the bucket containing i. If we do not call SPLITBUCKET(U), then we affect at most $3/\varepsilon$ bins. If we call SPLITBUCKET(U), then it must be the case that U had only ε^{-1} bins. When we merge U with the next bucket, and perhaps split this merged bucket, the new bucket U has at least $2/\varepsilon$ bins, and so, we will not call SPLITBUCKET(U) again. Thus, we will touch at most $3/\varepsilon$ buckets in any case. It follows that the total movement cost is at most $3d/\varepsilon$ (all bins following i store items of density at most d). Therefore, the amortized movement cost is negative. This proves the desired result. □

Dealing with Small Items: Summary. Combining Corollary C.4 and Lemma 3.4, we find that we can pack small items into bins which, ignoring some $O(1)$ many bins are $1 - \varepsilon$ full on average.

Formally, we have the following.

Lemma C.7. *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\varepsilon^2})$ amortized recourse for instances where all items have size at most ε which packs items into bins which, ignoring some $O(1)$ bins, are at least $1-\varepsilon$ full on average.*

Worst case bounds. Note that the above algorithm yields worst case bounds for several natural scenarios, given in the following corollaries.

Corollary C.8. *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\delta \cdot \varepsilon^2})$ worst case recourse for instances where all items have size in the range $[\delta, \varepsilon]$ which packs items into bins which, ignoring some $O(1)$ bins, are at least $1-\varepsilon$ full on average.*

Proof (Sketch). The algorithm is precisely the algorithm of Lemma C.7, only now in our analysis, as each item has size δ, any single removal can incur movement of at most ϵ^2 size, by our algorithm's definition. The worst case migration factor follows. □

Finally, if we group the items of size less than $(1/n, \epsilon]$ into ranges of size $(2^i, 2i+1]$ (guessing and doubling n as necessary) requires only $O(\log n)$ additive bins (one per size range), while allowing worst case recourse bounds by the previous corollary.

Corollary C.9. *For all $\varepsilon \leq \frac{1}{6}$ there exists a fully-dynamic bin packing algorithm with $O(\frac{1}{\varepsilon^2})$ worst case recourse for instances where all items have size at most ε which packs items into bins which, ignoring some $O(\log n)$ bins, are at least $1-\varepsilon$ full on average.*

C.2.6 SH Algorithms: Conclusion

§C.2.4 and §C.2.3 show how to maintain all invariants of SH algorithms, using $O(1)$ amortized recourse, provided $\varepsilon^{-1} = O(1)$ and $K = O(1)$, and $\beta_i, \gamma_i = O(1)$ for all $i \in [K]$. As the SH algorithm implied by Lemma C.2 satisfies these conditions, we obtain this section's main positive result.

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.*

D Omitted Proofs of Section 4 (Size Movement Costs)

Here we provide proofs for our matching amortized size cost upper and lower bounds.

D.1 Amortized Migration Factor Upper Bound

We start with a description and analysis of the trivial optimal algorithm for size costs.

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

Proof. We divide the input into *epochs.* The first epoch starts at time 0. For an epoch starting at time t, let V_t be the total volume of items present at time t. The epoch starting at time t ends when the total volume of items inserted or deleted during this epoch exceeds εV_t. We now explain the bin packing algorithm. Whenever an epoch ends (say at time t), we use an offline A(F)PTAS (e.g., [8] or [19]) to efficiently compute a solution using at most $(1+\varepsilon) \cdot OPT(\mathcal{I}_t) + O(\varepsilon^{-2})$ bins. (Recall that $\mathcal{I}_t$

denotes the input at time t.) We pack items arriving during an epoch in new bins, using the first-fit algorithm to pack them. If an item gets deleted during an epoch, we *pretend* that it is still in the system and continue to pack it. When an epoch ends, we remove all the items which were deleted during this epoch, and recompute a solution using an off-line A(F)PTAS algorithm as indicated above.

To bound the recourse cost, observe that if the starting volume of items in an epoch starting at time t is V_t, the volume at the end of this epoch is at most $V_t + A_t$, where A_t is the volume of items that arrived and departed during this epoch. As $A_t > \varepsilon V_t$, the cost of reassigning these items of volume at most $V_t + A_t$ can be charged to A_t. Specifically, the amortized migration cost of the epoch starting at time t is at most $(V_t + A_t)/A_t < V_t/\epsilon V_t + 1 = O(\varepsilon^{-1})$. Consequently, the overall amortized migration cost is $O(\varepsilon^{-1})$.

To bound the competitive ratio, we use the following easy fact: the optimal number of bins to pack a bin packing instance of total volume V lies between V and $2V$. (E.g., the first-fit algorithm achieves this upper bound.) Consider an epoch starting at time t. Let V_t denote the volume of the input $\mathcal{I}_t$ at time t. The algorithm uses at most $(1+\varepsilon)\cdot OPT(\mathcal{I}_t)+O(\varepsilon^{-2})$ bins at time t. Consider an arbitrary time t' during this epoch. As a packing of the instance $\mathcal{I}_{t'}$ can be extended to a packing of $\mathcal{I}_t$ by packing $\mathcal{I}_t \setminus \mathcal{I}_{t'}$ with the first fit algorithm, using a further $2\varepsilon V_t$ bins, we have $OPT(\mathcal{I}_t) \leq OPT(\mathcal{I}_{t'}) + 2\varepsilon V_t$. On the other hand, our algorithm uses at most an additional $2\varepsilon V_t$ bins at time t' compared to the beginning of the epoch. Therefore, the number of bins used at time t' is at most

$$\begin{aligned}(1+\varepsilon)\cdot OPT(\mathcal{I}_t) + 2\varepsilon V_t + O(\varepsilon^{-2}) &\leq (1+\varepsilon)\cdot(OPT(\mathcal{I}_{t'}) + 2\varepsilon V_t) + 2\varepsilon V_t + O(\varepsilon^{-2}) \\ &\leq (1+\varepsilon)\cdot OPT(\mathcal{I}_{t'}) + 5\varepsilon V_t + O(\varepsilon^{-2}).\end{aligned}$$

Now observe that $OPT(\mathcal{I}_{t'}) \geq V_t - \varepsilon V_t = (1-\varepsilon)\cdot V_t$ because the total volume of jobs at time t' is at least $V_t - \varepsilon V_t$, and so $OPT(\mathcal{I}_{t'}) \geq V_t/2$. Therefore, $5\varepsilon V_t \leq 10\varepsilon\, OPT(\mathcal{I}_{t'})$. It follows that the number of bins used by the algorithm at time t' is at most $(1+O(\varepsilon))\cdot OPT(\mathcal{I}_{t'}) + O(\varepsilon^{-2})$. $\square$

D.2 The Matching Lower Bound

Here we prove our matching lower bound for amortized recourse in the size costs setting.

We recall that our proof relies on Sylvester's sequence. For ease of reference we restate the salient properties of this sequence which we rely on in our proofs. The Sylvester sequence is given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$, or equivalently $k_{i+1} = k_i \cdot (k_i - 1) + 1$ for $i \geq 0$. The first few terms of this sequence are $2, 3, 7, 43, 1807, \ldots$ In what follows we let c be a large positive integer to be specified later, and $\varepsilon := 1/\prod_{\ell=1}^{c} k_\ell$. For notational simplicity, since our products and sums will always be taken over the range $[c]$ or $[c]\setminus\{i\}$ for some $i \in [c]$, we will write $\sum_\ell k_\ell$ and $\prod_\ell 1/k_\ell$, $\sum_{\ell\neq i} 1/k_\ell$, $\prod_{\ell\neq i} 1/k_\ell$, etc., taking the range of ℓ to be self-evident from context. In our proof of Theorem 4.2 and the lemmas building up to it, we shall make use of the following properties of the Sylvester sequence, its reciprocals and this $\epsilon = 1/\prod_\ell k_\ell$.

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

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

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

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

Properties (P4) and (P3) are immediate, while property (P2) follows directly from the recursive definition of the sequence's terms, $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$. Finally, the same definition readily implies $\frac{1}{k_i} = \frac{1}{k_i - 1} - \frac{1}{k_{i+1}-1}$, from which property (P1) follows by induction.

The instances we consider for our lower bound will be comprised of items with sizes given by entries of the following vector $\vec{s} \in [0,1]^{c+1}$, defined as follows:

$$s_i := \begin{cases} \frac{1}{k_i} \cdot (1 - \frac{\varepsilon}{2}) & i \in [c] \\ \varepsilon \cdot (\frac{3}{2} - \frac{\varepsilon}{2}) & i = c+1. \end{cases}$$

Let $\vec{1}$ be the all-ones vector, and $\vec{e_i}$ the i-th standard basis vector. In what follows we will denote by $\vec{\chi} \in \mathbb{N}^{c+1}$ a characteristic vector of a feasibly-packed bin; i.e., χ_i is the number of items of size s_i in the bin, and as the bin is not over-flowing, we have $\vec{\chi} \cdot \vec{s} = \sum_i \chi_i \cdot s_i \leq 1$. The following fact is easy to check from the definition of $\vec{s}$.

Fact D.1. $\vec{1} \cdot \vec{s} = 1$.

Proof. By property (P1), we have $\sum_{i=1}^{c} s_i = (1 - \frac{\varepsilon}{2}) \cdot (1 - \varepsilon) = 1 - s_{c+1}$. □

We now show that the item sizes given by the entries of $\vec{s}$ are such that in many situations (in particular, situations where no s_{c+1}-sized items are present), any feasible packing in a bin will leave some gap.

Lemma D.2. *Let χ denote a feasible packing of items into a bin; i.e., $\vec{\chi} \in \mathbb{N}^{c+1}$, $\vec{\chi} \cdot \vec{s} \leq 1$. Then,*

(a) If $\chi_{c+1} = 1$ and $\vec{\chi} \neq \vec{1}$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon/2$.

(b) If $\chi_{c+1} = 0$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon/2$.

(c) If $\chi_{c+1} = 0$ and $\vec{\chi} \neq k_i \vec{e_i}$, then $\vec{\chi} \cdot \vec{s} \leq 1 - \varepsilon$.

Proof. For ease of notation, we define a vector $\vec{t}$, where $t_i = 1/k_i$ for $i = 1, \dots, c$ and $t_{c+1} = 0$. So, $s_i = (1 - \varepsilon/2) \cdot t_i$ for $i = 1, \dots, c$.

Observation D.3. *The dot product $\vec{\chi} \cdot \vec{t}$ is at most 1.*

Proof. By property (P3), each t_i is an integral multiple of ε. It follows that $\vec{\chi} \cdot \vec{t}$ is also an integral multiple of ε. Clearly, 1 is an integral multiple of ε. Therefore, if $\vec{\chi} \cdot \vec{t} > 1$, then $\vec{\chi} \cdot \vec{t}$ is at least $1 + \varepsilon$. But then, $\vec{\chi} \cdot \vec{s} \geq (1 - \varepsilon/2) \cdot \vec{\chi} \cdot \vec{t} \geq (1 - \varepsilon/2) \cdot (1 + \varepsilon) > 1$, a contradiction. □

Part (a): First, observe that $\vec{\chi} \cdot \vec{t} \neq 1$. Indeed, it is at most 1, by Observation D.3). Moreover, if $\vec{\chi} \cdot \vec{t} = 1$, the fact that $\chi_{c+1} = 1$ would imply $\vec{\chi} \cdot \vec{s} = \vec{\chi} \cdot (1 - \varepsilon/2) \cdot \vec{t} + s_{c+1} = (1 - \varepsilon/2) + s_{c+1} > 1$, a contradiction. Also, there must be some index $i \in [c]$ such that $\chi_i = 0$. Indeed, otherwise the fact that $\vec{\chi} \neq \vec{1}$ and Fact D.1 imply that $\vec{\chi} \cdot \vec{s} > \vec{1} \cdot \vec{s} = 1$. So let i be an index such that $\chi_i = 0$. By property (P4), for all $j \neq i$, the value $t_j = 1/k_j$ is an integral multiple of $k_i \varepsilon$. Therefore, $\vec{\chi} \cdot \vec{t}$ is a multiple of $k_i \varepsilon$. Since $\vec{\chi} \cdot \vec{t} < 1$ and 1 is an integral multiple of $k_i \varepsilon$, it follows that $\vec{\chi} \cdot \vec{t} \leq 1 - k_i \varepsilon \leq 1 - 2\varepsilon$. Therefore, $\vec{\chi} \cdot \vec{s} \leq (1 - \varepsilon/2) \cdot (1 - 2\varepsilon) + s_{c+1} = 1 - \varepsilon + \varepsilon^2/2 \leq 1 - \varepsilon/2$.

Part (b) follows directly from Observation D.3 above. Indeed, since $\chi_{c+1} = 0$, we have that $\vec{\chi} \cdot \vec{s} = (1 - \varepsilon/2) \vec{\chi} \cdot \vec{t} \leq (1 - \varepsilon/2)$.

Part (c): First, observe that $\chi_i < k_i$ for $i = 1, \dots, c$. Indeed, if $\chi = k_i \vec{e_i} + \chi'$ for some index i and some non-zero non-negative vector χ', then $\vec{\chi} \cdot \vec{t} = 1 + \vec{\chi'} \cdot \vec{t} > 1$, which contradicts Observation D.3. Now let i be an index such that $\chi_i \geq 1$. Since $\chi_i < k_i$, property (P2) implies that $\chi_i \cdot t_i = \chi_i \cdot \varepsilon \cdot \prod_{\ell \neq i} k_\ell$ is not an integral multiple of $k_i \varepsilon$. But, by property (P4) for all $j \neq i$ we have that $t_j = 1/k_j$ is an integral multiple of $k_i \varepsilon$. Thus, $\vec{\chi} \cdot \vec{t} (\leq 1)$ is not an integral multiple of $k_i \varepsilon = k_i / \prod_\ell k_\ell = 1/\prod_{\ell \neq i} k_\ell$. Consequently, $\vec{\chi} \cdot \vec{t} < 1$. But, by property (P3), we have that $\vec{\chi} \cdot \vec{t}$ is an integral multiple of ε, from which it follows that $\vec{\chi} \cdot \vec{t} \leq 1 - \varepsilon$. Therefore, as $\chi_{c+1} = 0$, we have $\vec{\chi} \cdot \vec{s} = \vec{\chi} \cdot (1 - \epsilon/2) \cdot \vec{t} \leq (1 - \epsilon/2) \cdot (1 - \varepsilon) \leq 1 - \varepsilon$. □

Equipped with the above lemma, we can show two instances with similar overall weight, $\mathcal{I}$ to $\mathcal{I}'$ for which near-optimal solutions different significantly. Specifically, we define the input instances $\mathcal{I}$ and $\mathcal{I}'$, as follows. For some large N a product of $\prod_\ell k_\ell$, 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}. It is easy to check that $c = \Theta(\log\log 1/\varepsilon)$, and so the total number of items is $n = \Theta(N \cdot \log\log(1/\varepsilon))$. Therefore the additive $o(n)$ term, denoted by $f(n)$, satisfies

$$f(n) < \epsilon \cdot n/(42 \cdot \Theta(\log\log(1/\varepsilon))) = \epsilon N/42.$$

We now proceed to proving that approximately-optimal packings of the above two instances differ significantly.

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.*

Proof. We begin by proving our claimed bound on $\mathcal{A}$'s packing of $\mathcal{I}$. Fact D.1 shows that the optimal number of bins is N. Therefore, $\mathcal{A}$ is allowed to use at most $(1+\varepsilon/7)N + \varepsilon N/42 = (1+\varepsilon/6)N$ bins. Now suppose there more than $N/3$ bins for which the characteristic vector is not $\vec{1}$. Lemma D.2(a) shows that each such bin must leave out at least $\varepsilon/2$ space. Therefore, the total unused space in these bins is greater than $\varepsilon N/6$, which implies that the algorithm must use at least $N + \varepsilon N/6$ bins, a contradiction.

We now proceed to prove our claimed bound on $\mathcal{A}$'s packing of $\mathcal{I}'$. Let us first find the optimal value $OPT(\mathcal{I}')$. By property (P1), the total volume of all the items is equal to $N(1-\varepsilon)(1-\varepsilon/2)$. By Lemma D.2(b), any bin can be packed to an extent of at most $(1-\varepsilon/2)$. Therefore the optimal number of bins is at least $N(1-\varepsilon)$. Furthermore, we can achieve this bound by packing N items of size s_i in N/k_i bins for each index i. Therefore, the algorithm is allowed to use at most $(1+\varepsilon/7)(1-\varepsilon)N + \varepsilon N/42 \le (1-\frac{35\varepsilon}{42})N$ bins when packing $\mathcal{I}'$.

Suppose there are at least $N/2$ bins each of which is assigned items of at least two different sizes by the algorithm. By Lemma D.2(c), the algorithm will leave at least ε unused space in such bins; moreover, by Lemma D.2(b), the algorithm will leave at least $\varepsilon/2$ unused space in every bin. Thus, the total unused space in the bins is at least $\varepsilon N/2 + \varepsilon N/4 = 3\varepsilon N/4$. Since the total volume of the items is equal to $N(1-\varepsilon)(1-\varepsilon/2)$, we see that the total number of bins used by the algorithm is at least $N(1-\varepsilon)(1-\varepsilon/2) + 3\varepsilon N/4 \ge N(1-3\varepsilon/4) > (1-\frac{35\varepsilon}{42})N$, a contradiction. □

We are now ready to prove this section's main result.

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 $\Omega(\varepsilon^{-1})$.*

Proof. We will show that any algorithm $\mathcal{A}$ using at most $(1+\varepsilon/7)\cdot OPT(\mathcal{I}_t) + o(n)$ bins at time t uses at least $1/160\varepsilon$ amortized recourse, for arbitrarily large optima and instance sizes, implying our theorem. We consider the two instance $\mathcal{I}$ and $\mathcal{I}'$ defined above.

Suppose we first provide instance $\mathcal{I}$ to $\mathcal{A}$, and then remove all items of size s_{c+1} to get the instance $\mathcal{I}'$. Let B_1 and B_2 be the sets of bins guaranteed by Lemma 4.3 when we had the instances $\mathcal{I}$ and $\mathcal{I}'$, respectively. Notice that as algorithm $\mathcal{A}$ uses at most $(1-\frac{35\varepsilon}{42})N \le N$ bins while packing $\mathcal{I}'$, and $|B_1| \ge 2N/3$, $|B_2| \ge N/2$, we have that either (i) $|B_1 \cap B_2| \ge N/12$, or (ii) at least $N/12$ of the bins of B_1 are closed in $\mathcal{A}$'s packing of $\mathcal{I}'$.

Consider any bin which lies in both B_1 and B_2. In instance $\mathcal{I}$, algorithm $\mathcal{A}$ had assigned one item of each size to this bin, whereas in instance $\mathcal{I}'$ the algorithm assigns this bin items of only one size.

Therefore, the total size of items which need to go out of (or into) this bin when we transition from $\mathcal{I}$ to $\mathcal{I}'$ is at least $1/2$. Therefore, the total volume of items moved during this transition is at least $N/48$. Similarly, if $N/12$ of the bins of B_1 are closed in $\mathcal{A}$'s packing of $\mathcal{I}'$, at least $1 - s_{c+1} \geq 1/2$ volume must leave each of these bins, and so the total volume of items moved during this transition is at least $N/24 > N/48$.

Repeatedly switching between $\mathcal{I}$ and $\mathcal{I}'$ by adding and removing the N items of size s_{c+1} a total of T times (for sufficiently large T), we find that the amortized recourse is at least

$$\frac{T \cdot N/48}{N + T \cdot 3\varepsilon \cdot N} \geq \frac{1}{160\varepsilon}.$$

□

References

[1] Balogh, J., Békési, J., Dósa, G., Epstein, L., and Levin, A. 2017. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728*.

[2] Balogh, J., Békési, J., and Galambos, G. 2012. New lower bounds for certain classes of bin packing algorithms. *Theoretical Computer Science 440*, 1–13.

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

[4] Balogh, J., Békési, J., Galambos, G., and Reinelt, G. 2014. On-line bin packing with restricted repacking. *Journal of Combinatorial Optimization 27,* 1, 115–131.

[5] Berndt, S., Jansen, K., and Klein, K. 2015. Fully dynamic bin packing revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*. 135–151.

[6] Coffman Jr, E. G., Csirik, J., Galambos, G., Martello, S., and Vigo, D. 2013. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*. Springer, 455–531.

[7] Corless, R. M., Gonnet, G. H., Hare, D. E., Jeffrey, D. J., and Knuth, D. E. 1996. On the Lambert-W function. *Advances in Computational mathematics 5,* 1, 329–359.

[8] De La Vega, W. F. and Lueker, G. S. 1981. Bin packing can be solved within 1+ ε in linear time. *Combinatorica 1,* 4, 349–355.

[9] Epstein, L. and Levin, A. 2009. A robust APTAS for the classical bin packing problem. *Math. Program. 119,* 1, 33–49.

[10] Feldkord, B., Feldotto, M., and Sören, R. 2017. A tight approximation for fully dynamic bin packing without bundling. *arXiv preprint arXiv:1711.01231*.

[11] Gambosi, G., Postiglione, A., and Talamo, M. 1990. New algorithms for on-line bin packing. In *Algorithms and Complexity, Proceedings of the First Italian Conference*. 44–59.

[12] Gambosi, G., Postiglione, A., and Talamo, M. 2000. Algorithms for the relaxed online bin-packing model. *SIAM journal on computing 30,* 5, 1532–1551.

[13] Heydrich, S. and van Stee, R. 2016. Beating the harmonic lower bound for online bin packing. In *43rd International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl.

[14] Hoberg, R. and Rothvoss, T. 2017. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2616–2625.

[15] Hochbaum, D. S. 1996. *Approximation algorithms for NP-hard problems.* PWS Publishing Co.

[16] Ivković, Z. and Lloyd, E. L. 1996. A fundamental restriction on fully dynamic maintenance of bin packing. *Information Processing Letters 59,* 4, 229–232.

[17] Jansen, K. and Klein, K. 2013. A robust AFPTAS for online bin packing with polynomial migration,. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*. 589–600.

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

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

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

[21] Ramanan, P., Brown, D. J., Lee, C.-C., and Lee, D.-T. 1989. On-line bin packing in linear time. *Journal of Algorithms 10,* 3, 305–326.

[22] Richey, M. B. 1991. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics 34,* 1-3, 203–227.

[23] Sanders, P., Sivadasan, N., and Skutella, M. 2009. Online scheduling with bounded migration. *Mathematics of Operations Research 34,* 2, 481–498.

[24] Seiden, S. S. 2002. On the online bin packing problem. *Journal of the ACM (JACM) 49,* 5, 640–671.

[25] Sylvester, J. J. 1880. On a point in the theory of vulgar fractions. *American Journal of Mathematics 3,* 4, 332–335.

[26] Ullman, J. 1971. *The performance of a memory allocation algorithm.*

[27] van Vliet, A. 1995. *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.

[28] Woeginger, G. 1993. Improved space for bounded-space, on-line bin-packing. *SIAM Journal on Discrete Mathematics 6,* 4, 575–581.

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