Beating the Harmonic Lower Bound for Online Bin Packing*

Sandy Heydrich[1] and Rob van Stee[2]

1 **Max Planck Institute for Informatics, Saarbrücken, Germany; and Graduate School of Computer Science, Saarbrücken, Germany**
heydrich@mpi-inf.mpg.de

2 **Department of Computer Science, University of Leicester, Leicester, UK**
rvs4@le.ac.uk

Abstract

In the online bin packing problem, items of sizes in $(0, 1]$ arrive online to be packed into bins of size 1. The goal is to minimize the number of used bins. Harmonic++ achieves a competitive ratio of 1.58889 and belongs to the Super Harmonic framework [Seiden, J. ACM, 2002]; a lower bound of Ramanan et al. shows that within this framework, no competitive ratio below 1.58333 can be achieved [Ramanan et al., J. Algorithms, 1989]. In this paper, we present an online bin packing algorithm with asymptotic performance ratio of 1.5815, which constitutes the first improvement in fifteen years and reduces the gap to the lower bound by roughly 15%.

We make two crucial changes to the Super Harmonic framework. First, some of the decisions of the algorithm will depend on *exact sizes* of items, instead of only their types. In particular, for item pairs where the size of one item is in $(1/3, 1/2]$ and the other is larger than $1/2$ (a *large* item), when deciding whether to pack such a pair together in one bin, our algorithm does not consider their types, but only checks whether their total size is at most 1.

Second, for items with sizes in $(1/3, 1/2]$ (*medium* items), we try to pack the larger items of every type in pairs, while combining the smallest items with large items whenever possible. To do this, we *postpone* the coloring of medium items (i.e., the decision which items to pack in pairs and which to pack alone) where possible, and later select the smallest ones to be reserved for combining with large items. Additionally, in case such large items arrive early, we pack medium items with them whenever possible. This is a highly unusual idea in the context of Harmonic-like algorithms, which initially seems to preclude analysis (the ratio of items combined with large items is no longer a fixed constant).

For the analysis, we carefully mark medium items depending on how they end up packed, enabling us to add crucial constraints to the linear program used by Seiden. We consider the dual, eliminate all but one variable and then solve it with the ellipsoid method using a separation oracle. Our implementation uses additional algorithmic ideas to determine previously hand set parameters automatically and gives certificates for easy verification of the results.

We give a lower bound of 1.5766 for algorithms like ours. This shows that fundamentally different ideas will be required to make further improvements.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms

Keywords and phrases Bin packing, online algorithms, harmonic algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.41

* A full version of this paper can be found at http://arxiv.org/abs/1511.00876.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the online bin packing problem, a sequence of *items* with sizes in the interval $(0, 1]$ arrive one by one and need to be packed into *bins*, so that each bin contains items of total size at most 1. Each item must be irrevocably assigned to a bin before the next item becomes available. The algorithm has no knowledge about future items. There is an unlimited supply of bins available, and the goal is to minimize the total number of used bins (bins that receive at least one item). Bin packing is a classical and well-studied problem in combinatorial optimization. Extensive research has gone into developing approximation algorithms for this problem, e.g. [5, 7, 6, 12, 17, 9]. Such algorithms have provably good performance for any possible input and work in polynomial time. In fact, the bin packing problem was one of the first for which approximation algorithms were designed [10].

For bin packing, we are typically interested in the long-term behavior of algorithms: how good is the algorithm for large inputs? If we simply compare to the optimal solution, the worst ratio is often determined by some very small inputs. To avoid such pathological instances, the *asymptotic performance ratio* was introduced: For a given input sequence σ, let $\mathcal{A}(\sigma)$ be the number of bins used by algorithm $\mathcal{A}$ on σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

$$\mathcal{R}_{\mathcal{A}}^{\infty} = \limsup_{n\to\infty} \sup_{\sigma} \left\{ \frac{\mathcal{A}(\sigma)}{\mathrm{OPT}(\sigma)} \,\middle|\, \mathrm{OPT}(\sigma) = n \right\}. \tag{1}$$

From now on, we only consider the asymptotic competitive ratio unless otherwise stated.

Lee and Lee [13] presented an algorithm called Harmonic, which partitions the interval $(0, 1]$ into $m > 1$ intervals $(1/2, 1], (1/3, 1/2], \ldots, (0, 1/m]$. The type of an item is defined as the index of the interval which contains its size. Each type of items is packed into separate bins (i items per bin for type i). For any $\varepsilon > 0$, there is a number m such that the Harmonic algorithm that uses m types has a performance ratio of at most $(1 + \varepsilon)\Pi_\infty$ [13], where $\Pi_\infty \approx 1.69103$.

If we consider the bins packed by Harmonic, then it is apparent that in bins with type 1 items, nearly half the space can remain unused. It is better to use this space for items of other types. After a sequence of papers which used this idea to develop ever better algorithms [13, 15, 16], Seiden [18] presented a general framework called Super Harmonic which captures all of these algorithms. Super Harmonic algorithms classify items based on an interval partition of $(0, 1]$ and give each item a color as it arrives, red or blue. For each type of items j, the fraction of red items is some constant denoted by α_j. Blue items are packed as in Harmonic, i.e., for each item type j, every bin with blue items contains a maximal number of blue items. (This may leave some space for smaller red items of different types.) Red items are packed in bins which are only partially filled. The idea is that hopefully, later blue items of other types will arrive that can be placed into the bins with red items. Seiden [18] showed that the Super Harmonic algorithm Harmonic++, which uses 70 intervals for its classification and has about 40 manually set parameters, achieves a performance ratio of at most 1.58889.

Ramanan et al. [15] gave a lower bound of $19/12 \approx 1.58333$ for this type of algorithm. It is based on inputs like the one shown in Figure 1, which contains a *medium* item (size in $(1/3, 1/2]$) and a *large* item (size in $(1/2, 1]$). Both of these items arrive N times for some large number N, and although they fit pairwise into bins, the algorithm never combines them like this. No matter how fine the item classification of an algorithm, pairs of items such as these, that the algorithm does not pack together into one bin, can always be found. (To complete the lower bound construction, we also need to consider inputs containing the

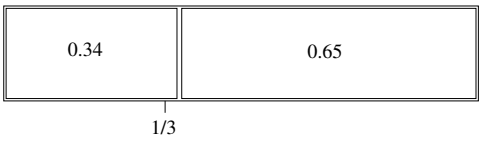
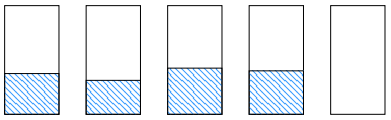
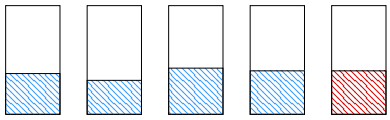


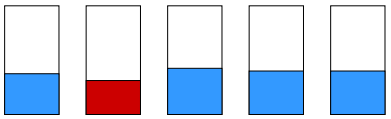
Figure 1 Part of the lower bound construction from Ramanan et al. [15]. The figure shows how one bin is packed in the *optimal* solution. Both of these items arrive many times.



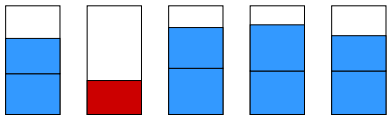
(a) Pack items one per bin with provisional coloring.



(b) A provisional red item arrives.



(c) We fix the colors. The smallest item becomes red.



(d) Additional blue items of the same type are added.

Figure 2 Illustration of the coloring in Extreme Harmonic. In this example, $\alpha = 1/9$. Note that the ratio of 1/9 does not hold (for the bins shown) at the time that the colors are fixed: 1/5 of the items are red at this point. The ratio 1/9 is achieved when all bins with blue items contain two blue items.

sizes $1/3+\varepsilon$, $1/2+\varepsilon$, which can be combined into a single bin, and the input consisting only of items of size $1/3+\varepsilon$.)

We avoid this lower bound construction by defining the algorithm so that it simply combines medium and large items *whenever* they fit together in a single bin. Essentially, we use Any Fit to combine such items into bins (under certain conditions specified below). This is a generalization of the well-known algorithms First Fit and Best Fit [19, 7], which have been used in similar contexts before [2, 1]. Proving formally that this helps to improve the asymptotic performance ratio requires a surprising amount of additional technical modifications to the algorithm and the proof, in particular setting up a complete marking scheme (see below).

As in the Super Harmonic framework, medium items that are packed in pairs are colored blue, and the ones that are packed alone into bins (possibly together with items of other types) are colored red. At this point it is important to note that medium items of any given type are not all exactly the same size, since the type only specifies an interval. This means that the items of any given type could arrive in such an order that all of the red items are slightly larger than the blue ones. Then, when large items arrive later, it could be that they are too large to fit in bins with red medium items, so the online algorithm is forced to pack them into new bins.

In order to benefit from using Any Fit, it is crucial to ensure that for each medium type, as much as possible, *it is the smallest items that are colored red.* We will do this by initially packing each medium item alone into a bin and giving it a provisional color. After several items of the same type have arrived, we will color the smallest one red and start packing additional medium items of the same type together with the other items, that are now colored blue. (See Figure 2.) In this way, we can ensure that at least *half* of the blue

items (namely, the ones that had already arrived at the time when we select the smallest to be red) are at least as large as the smallest red items. The point of this is that if those red items are still alone in bins at the end of the input, OPT cannot pack too many bins as shown in Figure 1, because this can only happen with large items that do not fit with the red items that remain alone in bins (Lemma 6).

We do not postpone the coloring decisions in the following two cases.

1. If a bin with suitable small red items is available, we will pack p into that bin and color it blue, regardless of the precise size of p.[1] In this case, in our analysis we will exploit the fact that these small items exist in the input, meaning that not *all* optimal bins are packed as shown in Figure 1: the small items must be packed somewhere (Lemma 7).
2. If bins with a *large* item are available, and p fits into such a bin, we will pack p in one such bin. This is the best case overall, since finding combinations like this was exactly our goal! However, there is a technical problem with this, which we discuss below.

Overall, we have three different cases: medium items are packed alone initially (in which case we have a guarantee about the sizes of some of the blue items), medium items are combined with smaller red items (so these red items exist and must be packed: Lemma 7), or medium items are combined with larger blue items (which is exactly our goal). The main technical challenge is to quantify these different advantages into one overall analysis. In order to do this (i.e., to prove Lemmas 6 and 7), we introduce - in addition to and separate from the coloring – a marking of the medium items, which we now describe.

$\mathcal{R}$ For any medium type j, a fraction α_j of the items marked $\mathcal{R}$ are red, and all of these **red** items are packed into mixed bins (i.e., together with a large item).

$\mathcal{B}$ For any medium type j, a fraction α_j of the items marked $\mathcal{B}$ are red, and the **blue** items are packed into mixed bins (i.e., together with red items of other (smaller) types)

$\mathcal{N}$ For any medium type j, a fraction α_j of the items marked $\mathcal{N}$ are red, and **none** of the red and blue items marked $\mathcal{N}$ are packed into mixed bins.

Our marking is illustrated in Figure 3. Maintaining the fraction α_j of red items for all marks separately is crucial for the analysis. However, we note here immediately that the fraction α_j of red items is **not** actually maintained continuously throughout the execution for all marks. This can be seen clearly for the items marked $\mathcal{R}$, where the ratio only becomes equal to α_j (ignoring rounding) after all the bins with single blue items in them receive additional blue items (see Figure 2).

Seemingly more problematically, it could happen that many large items arrive first, leading to more than an α_j fraction of the items of type j and mark $\mathcal{B}$ being packed with the large items and colored red. (Potentially, this could even happen to all of them.) While this is in principle exactly what we want to achieve, there is no guarantee that later in the input, sufficiently many additional items of type j will arrive to restore the correct ratio α_j. This is a problem for our analysis, which assumes the ratio α_j is maintained exactly. However, if we insist on maintaining this ratio throughout, i.e., if we color some of these items blue and pack them in pairs even though they could fit with existing large items, we end up with the same worst case instances as for Super Harmonic. We deal with this case by *modifying the input (for the analysis) after it has been packed.* By this and some additional postprocessing, we ensure that for each mark $\mathcal{R}, \mathcal{B}, \mathcal{N}$, an α_j fraction of the medium items of type j are indeed colored red in the end (ignoring rounding) as required. The postprocessing also ensures that

[1] Unless we already have sufficiently many blue items of the type of p, in which case we pack p into a separate bin and color it red to maintain the correct fraction of red items.

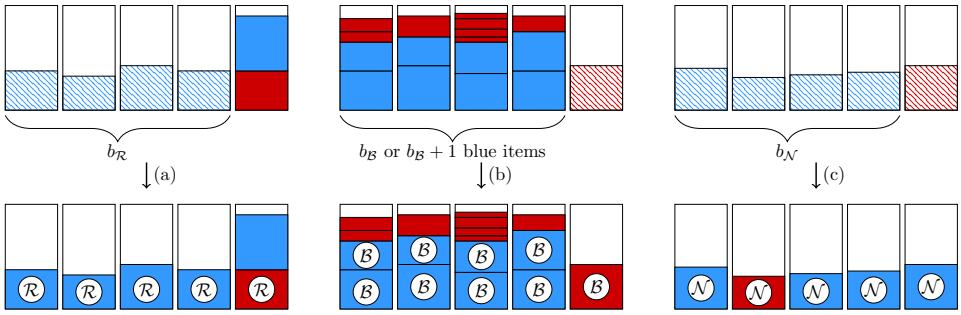


Figure 3 Illustrating the marking of the items. Again we take $\alpha_i = 1/9$. **(a)** Items get mark $\mathcal{R}$: provisionally blue items and a red item in a mixed bin. Bins with blue $\mathcal{R}$-items will receive a second blue item of the same type before a new bin is opened for this type. **(b)** Items get mark $\mathcal{B}$: a provisionally red item and blue items (in pairs) in mixed bins. **(c)** Items get mark $\mathcal{N}$: provisionally blue and provisionally red items. Note that in this step, the colors of items might be fixed to a different color than their provisional color. Bins with blue items will receive a second blue item of the same type before a new bin is opened for this type. See Fig. 2.

the marks are all correct. For instance, if a red item is packed with a blue item marked $\mathcal{N}$, the mark of that blue item gets adjusted in the end.

Like Seiden [18] and many other authors [19, 13, 15], we use weighting functions to analyze our algorithm. A weighting function defines a weight for each item type. By analyzing these, Seiden ended up with a set of mathematical programs that upper bounded the asymptotic performance ratio of Super Harmonic algorithms. These represented a kind of knapsack problems where each item has two different weights. Seiden used heuristics to get exact upper bounds for the solutions of these mathematical programs.

We use a different approach for the Extreme Harmonic framework. First of all we split each mathematical program into two standard linear programs, where both linear programs have a constraint that states its objective value should be smaller than that of the other one (representing for each one that the minimum is achieved for the set of weights it considers). To each linear program, we add two constraints that are based on the marking of the medium items. These constraints essentially state that in the optimal solution for a given input, there cannot be too many bins that are packed as shown in Figure 1 (unless the online algorithm also packs the items like this). This is the key to our improvement of the asymptotic performance ratio. However, after adding these constraints, the heuristic approach by Seiden can no longer be applied. Since each linear program has a very large number of variables but only four constraints, we take the dual and apply the ellipsoid method to solve it. To do this, we construct a separation oracle. This separation oracle solves a standard knapsack problem, making the results much easier to verify.

In order to apply the ellipsoid method, we write the dual in terms of just one variable, by eliminating two variables and assuming a third one to be given. This means that we can now do a straightforward binary search for the final remaining variable. We implemented a computer program which solves the knapsack problems and also does the other necessary work, including the automated setting of many parameters like item sizes and α values. As a result, our algorithm Son Of Harmonic requires far less manual settings than Harmonic++.

Our program uses an exact representation of fractions with arbitrary precision in order to avoid rounding errors. For our final calculations we have set the bound such that every dual LP is feasible; this means that our results do not rely on the correctness of any infeasibility claims (which are generally harder to prove). We provide a certificate and a

verifier program, and we also output the final set of knapsack problems directly to allow independent verification.

Our second main contribution is a new lower bound for all algorithms of this kind. The fundamental property of all these algorithms is that they color a fixed fraction of all items red (for each type). We show that no such algorithm can be better than 1.5766-competitive. Due to space constraints, this result is deferred to the full version.

1.1 Previous Results

The online bin packing problem was first investigated by Ullman [19]. He showed that the First Fit algorithm has performance ratio $\frac{17}{10}$. This result was then published in [7]. Johnson [11] showed that the Next Fit algorithm has performance ratio 2. Yao showed that Revised First Fit has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [21]. Brown and Liang independently improved this lower bound to 1.53635 [4, 14]. The lower bound stood for a long time at 1.54014, due to van Vliet [20], until it was improved to $\frac{248}{161} = 1.54037$ by Balogh et al. [3].

The *offline* version, where all the items are given in advance, is well-known to be NP-hard [8]. This version has also received a great deal of attention, for a survey see [5].

2 The Super Harmonic framework [18]

The fundamental idea is to first classify items by size, and then pack an item according to its type. We use numbers $t_1 \geq t_2 \geq \cdots \geq t_N$ to partition the interval $(0, 1]$ into subintervals (N is a parameter). We define $I_j = (t_{j+1}, t_j]$ for $i = 1, \ldots, N$ and $I_{N+1} = (0, t_{N+1}]$. An item of size s has type j if $s \in I_j$. A type j item has size at most t_j.

For each type j, a fraction $\alpha_j \in [0, 1]$ of items are colored red when they arrive, the rest are colored blue. Blue items are packed using Next Fit: we use each bin until exactly $\text{bluefit}_j := \lfloor 1/t_j \rfloor$ items are packed into it. Red items are also packed using Next Fit, but using only some fixed amount of the available space in a bin. This space is not necessarily exactly some value $1 - \text{bluefit}_j t_j$; for any given type j, there may be several other types that the algorithm will potentially pack into a bin together with items of type j. For each type of items that have size at most $1/3$, the algorithm chooses in advance an upper bound for the space that red items may occupy from a fixed set $\mathcal{D} = \{\Delta_i\}_{i=1}^{K}$ of spaces, where $\Delta_1 \leq \cdots \leq \Delta_K$. For *medium* items (i.e., items whose size is in $(1/3, 1/2]$), red items are packed one per bin. The number of red items of type i that are packed in one bin is denoted by redfit_i. In the space not used by blue (resp. red) items, the algorithm may pack red (resp. blue) items. Each bin will contain items of at most two different types.

A Super Harmonic algorithm uses a function $b : \{1, \ldots, N\} \to \{0, \ldots, K\}$ to map each item type to an index of a space in $\mathcal{D}$, indicating how much space for red items it leaves unused in bins with blue items of this type. Here $b(j) = 0$ means that no space is left for red items. The algorithm also uses a function $r : \{1, \ldots, N\} \to \{1, \ldots, K\}$ to map how much space (given by an index of $\mathcal{D}$) red items of each type require.

We say that the *class* of an item of type j is $b(j)$, if it is blue, and $r(j)$ if it is red.[2] Thus, the class of a blue item reflects how much space is left (at least) in a bin with blue items of this type, and the class of a red item indicates how much space red items of this type require (at most) in a bin. There are four kinds of bins.

[2] Seiden used the notation $\phi(j)$ and $\varphi(j)$ for these functions.

- Pure blue: $\{i|b(i) = 0, 1 \leq i \leq N\}$. No red items are ever packed into such bins.
- Unmixed blue: $\{(i, ?)|b(i) \neq 0, 1 \leq i \leq N\}$. There is at least one blue item in the bin, and red items might still be packed into it (in the free space of size $\Delta_{b(i)}$).
- Unmixed red: $\{(?, j)|\alpha_i \neq 0, 1 \leq i \leq N\}$. There is at least one red item in the bin and no blue items, but blue items might still be packed in it (in the free space of size $1 - \Delta_{r(i)}$).
- Mixed bins: $\{(i, j)|b_i \neq 0, \alpha_j \neq 0, t_j \leq \Delta_{b(i)}, 1 \leq i \leq N, 1 \leq j \leq N\}$. There are items of both colors.

An unmixed blue bin is *compatible* with a red item of type i if the bin is in a group $(j, ?)$ and $b(j) \geq r(i)$. An unmixed red bin is compatible with a blue item of type i if the bin is in a group $(?, j)$ and $b(i) \geq r(j)$. In both cases, the condition means that the blue items and the red items together would use at most 1 space in the bin (the blue items leave enough space for the red items).

3 Marking the items and the Extreme Harmonic framework

The heart of our improvement over the Super Harmonic framework is marking the medium items. It enables us to keep track of how they are packed, allowing us to prove the crucial Lemmas 6 and 7 later, which bound how often "bad" patterns of the form shown in Figure 1 (which have weight > 1.5815) can be used in the optimal solution. Mark Items divides the medium items into three sets $\mathcal{N}, \mathcal{B}$ and $\mathcal{R}$ (see Figure 3). Every time an item arrives, after it is packed using the new framework below, Mark Items performs one of the three steps in Figure 3 if possible. This is done to keep the number of provisionally colored items small (a constant). We define Mark Items formally in the full version.

▶ **Theorem 1.** *At all times, there are at most* $5/\alpha_i$ *provisionally colored items of type* i*.*

Once assigned, an item remains in a set until the end of the input. This holds even if e.g. a blue item is packed with a red $\mathcal{N}$-item, meaning that a more appropriate mark for the red item is $\mathcal{B}$. We change marks where needed only after all items have been packed.

Let n^i count the total number of items of type i, and n^i_r count the number of red items of type i. For a given type i and set X, denote the number of red items in set X by $n^i_r(X)$, the number of blue items by $n^i_b(X)$, and the total number of items by $n^i(X)$. After all items have arrived and after some postprocessing, we will have

$$n^i_r(X) = \lfloor \alpha_i n^i(X) \rfloor \text{ for } X \in \{\mathcal{N}, \mathcal{B}, \mathcal{R}\} \text{ and each medium type } i. \tag{2}$$

▶ **Definition 2.** An unmixed bin is *red-compatible* with a newly arriving item if (1) the bin contains (provisionally) blue items of type i, the new item is of type j and will be colored red, and $b(i) \geq r(j)$, *or* (2) the bin contains a large item of size s, the new item is medium and has size at most $1 - s$. The definition for unmixed bins being blue-compatible to new items is completely analogous.

We say that a (mixed or unmixed) bin is *red-open* if it contains some non-provisionally red items but can still receive additional red items. We define *blue-open* analogously.

Like Super Harmonic algorithms, an Extreme Harmonic algorithm first tries to pack a red (blue) item into a red-open (blue-open) bin with items of the same type and color; then it tries to find a unmixed compatible bin; if all else fails, it opens a new bin. Of course, the definition of compatible has been extended compared to Super Harmonic (where this concept was not defined explicitly). Note that the choice of bin depends on the actions of Mark Items, since that algorithm fixes the colors of some items and bins.

1 $n^i \leftarrow n^i + 1$
2 **if** *p is medium, $\alpha_i > 0$, and there exists a red-compatible bin B with a large item* **then**
3 Place p in B and label it as bonus item. `// special case: existing bin`
4 $n^i \leftarrow n^i - 1$ `// we do not count this item for type` i
5 **else**
6 **if** $n_r^i < \lfloor \alpha_i n^i \rfloor$ **then** `// pack a (provisionally) red item`
7 **if** *there is a bin B with a bonus item of type i or there is a bin B with a bonus item of type j and $r(i) \leq b(j)$* **then**
8 Label the medium item in B as type i and color it red. It is no longer a bonus item.
9 $n^i \leftarrow n^i + \text{redfit}_i$ `// count medium item as type` i `item(s)`
10 $n_r^i \leftarrow n_r^i + \text{redfit}_i$
11 Pack(p, blue) `// since we now have` $n_r^i \geq \lfloor \alpha_i n^i \rfloor$ `again`
12 **else** Pack(p, red)
13 **else** `// pack a (provisionally) blue item`
14 **if** $b(i) = 0$ then pack p using Next Fit into pure bins of type i and color p blue.
15 **else** Pack(p, blue)
16 Update the marks and colors using Mark Items.

Algorithm 1: How the Extreme Harmonic framework packs a single item p of type $i < n$. At the beginning, we set $n_r^i \leftarrow 0$ and $n^i \leftarrow 0$ for $1 \leq i \leq n$.

1 Try the following types of bins to place p with (provisional) color c in this order:
2 - a mixed or unmixed c-open bin with items of type i and **definite** color c
3 - a c-compatible unmixed bin (the bin becomes mixed, its items' colors are fixed)
4 - a new unmixed bin
5 If p was packed into a new bin, p is medium and $\alpha_i > 0$, give p provisionally the color c, else give it the definite color c. If p got the definite color red, $n_r^i \leftarrow n_r^i + 1$.

Algorithm 2: The algorithm Pack(p, c) for packing an item p of type i with color $c \in \{\text{blue, red}\}$.

The new framework is formally described in Algorithm 1 and 2. We require $\alpha_i < 1/3$ for all types i. We discuss the changes from Super Harmonic one by one. All the changes stem from our much more careful packing of medium items.

As can be seen in Algorithm 2 (lines 2, 4 and 5), medium items that are packed into new bins are initially packed **one** per bin and given a provisional color. The goal of having provisionally colored items is to try and make sure that the *smallest* items of each type become red in the end. Thus, we wait until some number of these items have arrived, and then color the smallest one red (Figure 2).

When an item arrives, in many cases, we cannot postpone assigning it a color, since a c-open or c-compatible bin is already available (see lines 2–3 of Pack(p, c)). Additionally, we need to check right at the start whether a suitable large item has already arrived. We deal with this case in lines 2–4 of Algorithm 1. In this special case, we *ignore* the value α_i. We pack the medium item with the large item as if it was a red item, but we *do not count* it towards the total number of existing items of its type; instead we label it a **bonus item**. Bonus items do not have a color or mark, at least initially.

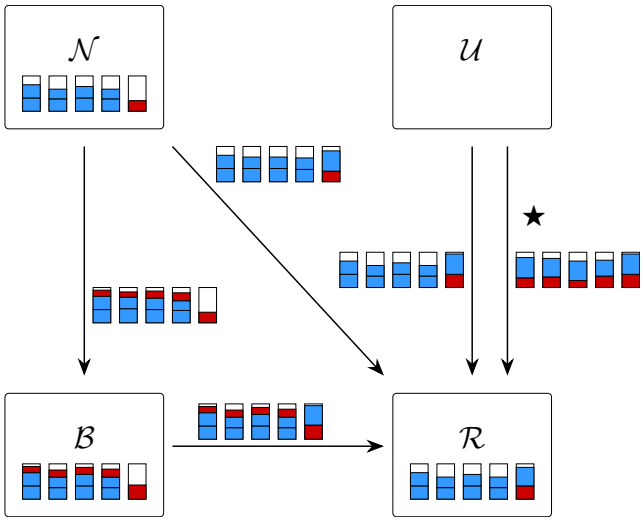


Figure 4 Reassigning marks after the input is complete. Items are sorted into their correct sets whenever possible, updating the marks that they received while the algorithm was running. Some item sizes are reduced. The bins next to the arrows indicate what sets of bins are being reassigned. The step marked with ★ takes place at the end of the postprocessing, after all other steps.

This means that we have (possibly temporarily) too many items of this type that are packed as red items (we do not count them towards the quantities n^i and n^i_r, but we do record that they exist). There are several ways that this can be fixed later on. Either, additional blue items of type i arrive and we can restore the correct ratio of red items. Or, some item of type j and size at most $1/3$ arrives that should be colored red and is compatible with blue items of type i. In this case, for our accounting, we replace the bonus item with redfit_j red items of type j, adjust the counts accordingly in lines 9–10, and color the new type j item blue.[3] Finally, it could also happen that some bonus items remain until the end; in this case we use careful post-processing so that each item does have a type and color at the end, and the ratio α_i is maintained. Note that we only modify item sizes for the analysis, and we only make items smaller, so the value of the optimal solution can only decrease and the implied competitive ratio can only increase as a result. Also note that allowing bonus items (i.e., occasionally packing too many items as red items) is essential to achieve a better competitive ratio; without this, we would get the same lower bound instances as before.

It can be seen that blue items of size at most $1/3$ are packed as in Super Harmonic. For red items of size at most $1/3$, we need to deal with existing bonus items in lines 9–10, and in line 3 of Pack(p, c), the provisional color of an existing item may be made permanent. Otherwise, the packing proceeds as in Super Harmonic. By the order in which existing bins are tried for packing new items, c-open bins always take precedence over other bins.

4 Postprocessing

After the algorithm has packed all items, we perform some postprocessing. For an overview of our changes of marks and sizes, see Figure 4. A formal version is given in the full paper.

[3] Note that the meanings of i and j are switched in the description of the algorithm for reasons of presentation.

▶ **Theorem 3.** *After postprocessing, (2) holds. Each blue item in $\mathcal{N}, \mathcal{R}$ and $\mathcal{B}$ is packed in a bin that contains two blue items. No bins with items in $\mathcal{N}$ or* red *items in $\mathcal{B}$ are mixed.*

In line 3 of Extreme Harmonic, bonus items are created. These are medium items which are packed together with a large blue item. Some of them may still be bonus when the algorithm has finished. Also, some of them may be labeled with a different type than the type they belong to according to their size. We call such items reduced items. In an additional postprocessing step, we split up reduced items into (possibly several) red items of the type with which we labeled the item. If any bonus items remain, we modify the packing that the algorithm outputs (for the analysis) by replacing some number of bins with a large blue item and a red medium item by the same number of bins with two blue medium items. Note that we only make items smaller, so all items still fit in their bins in both the optimal packing and the online packing. We finally achieve the following result.

▶ **Theorem 4.** *For each type i, we have $n_r^i \in [\lfloor \alpha_i n^i \rfloor - 3, \lfloor \alpha_i n^i \rfloor]$.*

5 Analysis using weights

Let $\mathcal{A}$ be an Extreme Harmonic algorithm. For analyzing the asymptotic performance ratio of $\mathcal{A}$, we will use the well-known technique of weighting functions: We assign weights to each item such that the number of bins that our algorithm uses in order to pack a specific input is equal to the sum of the weights of all items in this input. Then, we determine the maximum weight that can be packed in a single bin. Clearly, the offline algorithm cannot pack more weight than this in any of its bins, thus this maximum weight for a single bin gives us an upper bound on the competitive ratio.

Recall that the class of a red item of type i is $r(i)$ and the class of a blue item of type i is $b(i)$. Let $\mathfrak{r}$ be the smallest red item in a bin that has no blue items. Let the type of $\mathfrak{r}$ be ℓ, and $k = r(\ell)$. The weights of a non-large item p will depend on its class relative to k, and on its mark in case its class is k. The value of k (and the marks) become clear by running the algorithm. Note that the algorithm including the postprocessing does not depend on the weight functions in any way. There are $2K$ weighting functions in total, where $K = |\mathcal{D}|$ is the number of different spaces used for red items. For each k, w_k counts all the blue items, and v_k counts all the red items. The two weight functions of an item p of type i and mark m are given by the following table. The marks are only relevant for items of class k.

$w_k(p) = w_k(i,m)$		$v_k(p) = v_k(i,m)$	
$\frac{1-\alpha_i}{\text{bluefit}_i} + \frac{\alpha_i}{\text{redfit}_i}$	if $r(i) > k$	$\frac{1-\alpha_i}{\text{bluefit}_i} + \frac{\alpha_i}{\text{redfit}_i}$	if $b(i) < k$
$\frac{1-\alpha_i}{\text{bluefit}_i} + \frac{\alpha_i}{\text{redfit}_i}$	if $r(i) = k, m \in \{\mathcal{N}, \mathcal{B}, \emptyset\}$	$\frac{\alpha_i}{\text{redfit}_i}$	if $b(i) \geq k$
$\frac{1-\alpha_i}{\text{bluefit}_i}$	if $r(i) = k, m = \mathcal{R}$		
$\frac{1-\alpha_i}{\text{bluefit}_i}$	if $r(i) < k$		

▶ **Theorem 5.** *We have $\mathcal{A}(\sigma) \leq \max_{1 \leq k \leq K+1} \min\{\sum_{i=1}^n w_k(p_i), \sum_{i=1}^n v_k(p_i)\} + O(1)$ for any Extreme Harmonic algorithm $\mathcal{A}$ and any input σ.*

A *pattern* is a tuple $q = \{q_1, \ldots, q_m\}$ such that $\sum_{i=1}^m q_i t_{i+1} < 1$. Intuitively, a pattern describes the contents of a bin in the optimal offline solution. For a given weight function w, the weight of pattern q is $w(q) = w\left(1 - \sum_{i=1}^m q_i t_{i+1}\right) + \sum q_i w(t_i)$.

Denote the (finite) set of patterns by $\mathcal{Q}$. We can define an offline algorithm for a given input by a distribution χ over the patterns, where $\chi(q)$ indicates which fraction of the bins

are packed using pattern q. To show that a given Extreme Harmonic algorithm has performance ratio at most 1.5815 for input sequences with $\mathfrak{r}$ having class k, we must show

$$\frac{\min\{\sum_{i=1}^n w_k(p_i), \sum_{i=1}^n v_k(p_i)\}}{OPT(\sigma)} = \min\left\{\frac{\sum_{i=1}^n w_k(p_i)}{OPT(\sigma)}, \frac{\sum_{i=1}^n v_k(p_i)}{OPT(\sigma)}\right\}$$
$$\leq \min\left\{\sum_{q\in\mathcal{Q}} \chi(q) w_k(q), \sum_{q\in\mathcal{Q}} \chi(q) v_k(q)\right\} \leq 1.5815 \quad (3)$$

for all such inputs σ. As can be seen from this bound, the question now becomes: what is the distribution χ (the mix of patterns) that maximizes the minimum in (3)?

For this χ, the following constraints hold. Consider an input where $\mathfrak{r} > 1/3$. Let $m(q)$ be the number of $\mathcal{N}$-items of type ℓ in pattern q. Let q_1 be the pattern with an $\mathcal{N}$-item of type ℓ and an item of type i where $b(i) = k-1$. (Such an item is larger than $1-\mathfrak{r}$.) The parameters of the algorithm, in particular the type boundaries, must be such that this pattern is unique (i.e., no non-sand item can be added); it is easy to ensure this holds by setting an appropriate upper bound for the sand.

▶ **Lemma 6.** *If $\mathfrak{r} > 1/3$ and the type of $\mathfrak{r}$ is ℓ, then $m(q) \in \{0,1,2\}$ for all q, and $\chi(q_1) \leq \frac{1-\alpha_\ell}{1+\alpha_\ell} \sum_{q\neq q_1} \chi(q) m(q)$.*

For any j and q, let $n_j(q)$ be the number of items of type j in pattern q. Let q_2 be the pattern with a $\mathcal{B}$-item of the type of $\mathfrak{r}$ and an item larger than $1-\mathfrak{r}$. Like q_1, q_2 should be unique (this is easy to guarantee and check). Note that the patterns q_1 and q_2 are versions of the pattern shown in Figure 1.

▶ **Lemma 7.** *If $\mathfrak{r} > 1/3$, and ℓ is the type of $\mathfrak{r}$, $\frac{1-\alpha_\ell}{2}\chi(q_2) \leq \sum_{r(j)\leq b(\ell)} \sum_q \frac{\alpha_j}{\text{redfit}_j} \chi(q) n_j(q)$.*

Maximizing the minimum in (3) is the same as maximizing the first term under the condition that it is not larger than the second term—except that this condition might not be satisfiable, in which case we need to maximize the second term. We are led to consider two linear programs, which we will call LP_w^k and LP_v^k. Let $\mathcal{Q} = \{q_1, \ldots, q_{|Q|}\}$ and let $\chi_i = \chi(q_i), w_{ik} = w_k(q_i), n_{ij} = n_j(q_i), m_i = m(q_i)$. LP_w^k is the following linear program.

$$\begin{aligned}
\max \quad & \sum_{i=1}^{|\mathcal{Q}|} \chi_i w_{ik} && \text{// First term in (3)} && (4)\\
s.t. \quad & \chi_1 - \tfrac{1-\alpha_\ell}{2} \sum_{i=2}^{|\mathcal{Q}|} \chi_i m_i \leq 0 && \text{// Lemma 6} && (5)\\
& \tfrac{1-\alpha_\ell}{2}\chi_2 - \sum_{j:r(j)\leq b(\ell)} \sum_{i=3}^{|\mathcal{Q}|} \tfrac{\alpha_j}{\text{redfit}_j} \chi_i n_{ij} \leq 0 && \text{// Lemma 7} && (6)\\
& \sum_{i=3}^{|\mathcal{Q}|} \chi_i (w_{ik} - v_{ik}) \leq 0 && \text{// Bound on first term} && (7)\\
& \sum_{i=1}^{|\mathcal{Q}|} \chi_i \leq 1 && \text{// } \chi \text{ is a distribution} && (8)\\
& \chi_i \geq 0, 1 \leq i \leq |\mathcal{Q}| && \text{// } \chi \text{ is a distribution} && (9)
\end{aligned}$$

LP_w^k has a very large number of variables but only four constraints (apart from the non-negativity constraints). In (7) we use the following proposition.

▶ **Proposition 8.** $w_{1k} = v_{1k}, w_{2k} = v_{2k}$.

The dual DP_w^k is the following.

$$\begin{aligned}
\min \quad & y_4 && && (10)\\
s.t. \quad & y_1 + y_4 \geq w_{1k} && && (11)\\
& \frac{1-\alpha_\ell}{2} y_2 + y_4 \geq w_{2k} && && (12)\\
& -\tfrac{1-\alpha_\ell}{2} m_i y_1 - y_2 \sum_{j:r(j)\leq b(\ell)} \tfrac{\alpha_j}{\text{redfit}_j} n_{ij} + (w_{ik} - v_{ik}) y_3 + y_4 \geq w_{ik} && i = 3, \ldots, |\mathcal{Q}| && (13)\\
& y_i \geq 0 && i = 1, \ldots, 4 && (14)
\end{aligned}$$

If the objective value of DP_w^k as well as that of DP_v^k is at most some value y_4^* (or if one is infeasible), then y_4^* upper bounds the asymptotic performance ratio of our algorithm for this value of k by duality and by (3). It is easy to see that if for some feasible y^*, constraint (11) or (12) is not tight, then we can decrease y_1^* or y_2^* and still have a feasible solution. We therefore restrict our search to solutions for which (11) and (12) are tight. Given y_4^*, we then know the values of y_1^* and y_2^*. If the constraint (13) does not hold for pattern q_i and a given dual solution y^*, we have the following:

$$(1-y_3^*)w_{ik} + y_3^* v_{ik} + \frac{1-\alpha_\ell}{2} m_i y_1^* + y_2^* \sum_{j:r(j)\le b(\ell)} \alpha_j n_j > y_4^* \tag{15}$$

Note that we get exactly the same condition by considering DP_v^k due to symmetry.

Recall that w_{ik} and v_{ik} are just the sums of the respective weights of all the non-sand items in pattern q_i. Based on (15), we define a new weighting function $\omega(p)$ as follows.

$$\omega(p) = \begin{cases} (1-y_3^*)w_k(p) + y_3^* v_k(p) + \frac{1-\alpha_\ell}{2} y_1^* & \text{type of } p \text{ is } \ell \ (= \text{ type of } e) \\ (1-y_3^*)w_k(p) + y_3^* v_k(p) + y_2^* \alpha_j & \text{type of } p \text{ is } j, r(j) \le b(\ell) \\ (1-y_3^*)w_k(p) + y_3^* v_k(p) & \text{else} \end{cases}$$

The inequality (15) then turns into $\omega(q_i) > y_4^*$. For given y_4^*, we can therefore determine feasibility of (11)–(13) by using the ellipsoid method, fortunately for only one dimension: that is, we do a binary search for $y_3^* \in [0,1]$. For every value y_3^* that we consider, we solve a simple knapsack problem to determine $W = \max_{q\in\mathcal{Q}} \omega(q)$ using a dynamic program.

Summarizing the above discussion, proving that an algorithm is c-competitive can be done by running the described binary search for $k = 1, \ldots, K$ using $y_4^* = c$. Note that for $\mathfrak{r} \le 1/3$, we do not have conditions (5) and (6), and we can define $\omega(p) = (1-y_3^*)w_k(p) + y_3^* v_k(p)$ for all items.

For our algorithm Son Of Harmonic we have set initial values as follows. The last three columns contain item sizes and corresponding α_i values that were set manually, separated by semicolons. Numbers of the form $1/i$ until the value t_N are added automatically by our program if they are not listed below, but only up to 1/50; for very small items, we (automatically) merge some consecutive classes without loss of performance to speed up the binary search.

$c = \frac{15815}{10000}$	Item bounds and α values:	1/4;106/1000	3/20;0
$t_N = \frac{1}{2^{100}}$	33345/100000;0	8/39;8/100	29/200;0
$\gamma = \frac{2}{7}$	33340/100000;0	1/5;93/1000	1/7;16/100
(starting from $\frac{1}{14}$)	5/18;2/100	3/17;3/100	1/13;1/8
Last type before small type generation: $\frac{1}{50}$	7/27;105/1000	1/6;8/100	1/14;1/13

The remaining values α_i are set automatically using heuristics designed to speed up the search and minimize the resulting upper bound. In the range $(1/3, 1/2]$, we automatically generate item sizes (with corresponding α values and Δ_i values) that are less than t_N apart to ensure uniqueness of q_1 and q_2. The value γ specifies how much room is used by red items of size at most 1/14; larger items ($\le 1/3$) use at most 1/3 room. Our computer program and more information is available at `http://people.mpi-inf.mpg.de/~heydrich/extremeHarmonic/index.html`.

Acknowledgements. We thank the anonymous referees for their useful comments.

References

1 Luitpold Babel, Bo Chen, Hans Kellerer, and Vladimir Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Applied Mathematics*, 143(1-3):238–251, 2004. doi:10.1016/j.dam.2003.05.006.

2 János Balogh, József Békési, György Dósa, Jirí Sgall, and Rob van Stee. The optimal absolute ratio for online bin packing. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1425–1438. SIAM, 2015. doi:10.1137/1.9781611973730.94.

3 János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. In Klaus Jansen and Roberto Solis-Oba, editors, *Approximation and Online Algorithms – 8th International Workshop, WAOA 2010, Liverpool, UK, September 9-10, 2010. Revised Papers*, volume 6534 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2010. doi:10.1007/978-3-642-18318-8_3.

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

5 Edward G. Coffman, Michael R. Garey, and David S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

6 Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

7 Michael R. Garey, Ronald L. Graham, and Jeffrey D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, pages 143–150. ACM, 1972.

8 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the theory of of NP-Completeness*. Freeman and Company, San Francisco, 1979.

9 Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 830–839. SIAM, 2014. doi:10.1137/1.9781611973402.61.

10 David S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

11 David S. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

12 Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 312–320, 1982.

13 Chung-Chieh Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32:562–572, 1985.

14 Frank M. Liang. A lower bound for online bin packing. *Information Processing Letters*, 10:76–79, 1980.

15 Prakash V. Ramanan, Donna J. Brown, Chung-Chieh Lee, and D. T. Lee. Online bin packing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

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

17 Thomas Rothvoß. Approximating bin packing within o(log OPT * log log OPT) bins. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 20–29. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.11.

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

19 Jeffrey D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.

20 André van Vliet. An improved lower bound for online bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992.

21 Andrew C. C. Yao. New algorithms for bin packing. *Journal of the ACM*, 27:207–227, 1980.