

# Online Bin Packing with Resource Augmentation\*

Leah Epstein<sup>†</sup>

Rob van Stee<sup>‡</sup>

September 17, 2007

## Abstract

In competitive analysis, we usually do not put any restrictions on the computational complexity of online algorithms, although efficient algorithms are preferred. Thus if such an algorithm were given the entire input in advance, it could give an optimal solution (in exponential time). Instead of giving the algorithm more knowledge about the input, in this paper we consider the effects of giving an online bin packing algorithm larger bins than the offline algorithm it is compared to. We give new algorithms for this problem that combine items in bins in an unusual way and give bounds on their performance which improve upon the best possible bounded space algorithm. We also give general lower bounds for this problem which are nearly matching for bin sizes  $b \geq 2$ .

## 1 Introduction

In this paper we investigate the bin packing problem, one of the oldest and most thoroughly studied problems in computer science [2, 3]. In particular, we investigate this problem using the resource augmentation model, where the online algorithm has bins of size  $b \geq 1$  and is compared to an offline algorithm that has bins of size 1. We show improved upper bounds and general lower bounds for this problem.

**Problem Definition** In the *classical bin packing* problem, we receive a sequence  $\sigma$  of items  $p_1, p_2, \dots, p_N$ . Each item has a fixed *size* in  $(0, 1]$ . In a slight abuse of notation, we use  $p_i$  to indicate both the  $i$ th item and its size. We have an infinite supply of *bins* each with *capacity* 1. Each item must be assigned to a bin. Further, the sum of the sizes of the items assigned to any bin may not exceed its capacity. A bin is *empty* if no item is assigned to it, otherwise it is *used*. The goal is to minimize the number of bins used.

In the *resource augmentation* model [8, 11], one compares the performance of a particular algorithm  $\mathcal{A}$  to that of the optimal offline algorithm (denoted by OPT) in an unfair way. The optimal offline algorithm uses bins of capacity one, where  $\mathcal{A}$  is allowed to use bins of capacity  $b > 1$ . The goal is still to minimize the *number* of bins used.

In the *online* versions of these problems, each item must be assigned in turn, without knowledge of the next items. Since it is impossible in general to produce the best possible solution when computation occurs online, we consider approximation algorithms. Basically, we want to find an algorithm that incurs cost

---

\*A preliminary version of this paper appears in Proceedings of the 2nd Workshop on Approximation and Online Algorithms (WAOA 2004).

<sup>†</sup>School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. lea@idc.ac.il. Research supported by Israel Science Foundation (grant no. 250/01).

<sup>‡</sup>Department of Computer Science, University of Karlsruhe, D-76128 Karlsruhe, Germany. vanstee@ira.uka.de. Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-061-000.

which is within a constant factor of the minimum possible cost, no matter what the input is. This constant factor is known as the asymptotic performance ratio.

The resource augmentation model was introduced due to the following drawback of standard competitive analysis. Competitive analysis compares the performance of an online algorithm, which must pack each item upon arrival, to that of an omniscient and all-powerful offline algorithm that gets the input as a set. Resource augmentation gives more power to the online algorithm, making the analysis more general. In particular, it allows us to exclude instances that crucially depend on exact values in the input (in the current paper, the bin size) and that could give an unrealistically negative view of the performance of an online algorithm.

This approach has also been used to show that certain simple scheduling algorithms have constant competitive ratio for the problems  $1 \mid r_i, pmtn \mid \sum F_i$  and  $1 \mid r_i, pmtn \mid \sum w_i(1 - U_i)$  if they are given a small amount of resource augmentation [8]. A constant competitive ratio for these problems is not possible in the strict online model. Moreover, the strict online model often requires algorithms which seem unnecessarily complicated.

A bin-packing algorithm uses *bounded space* if it has only a constant number of bins available to accept items at any point during processing. These bins are called *open* bins. Bins which have already accepted some items, but which the algorithm no longer considers for packing are *closed* bins. While bounded space algorithms are sometimes desirable, it is often the case that unbounded space algorithms can achieve lower performance ratios.

We define the asymptotic performance ratio more precisely. For a given input sequence  $\sigma$ , and a fixed bin size  $b$ , let  $\text{cost}_{\mathcal{A},b}(\sigma)$  be the number of bins (of size  $b$ ) used by algorithm  $\mathcal{A}$  on  $\sigma$ . Let  $\text{cost}(\sigma)$  be the minimum possible cost to pack items in  $\sigma$  using bins of size 1. The *asymptotic performance ratio* for an algorithm  $\mathcal{A}$  is defined to be

$$R_{\mathcal{A},b}^{\infty} = \limsup_{n \rightarrow \infty} \max_{\sigma} \left\{ \frac{\text{cost}_{\mathcal{A},b}(\sigma)}{\text{cost}(\sigma)} \mid \text{cost}(\sigma) = n \right\}.$$

The *optimal asymptotic performance ratio* is defined to be  $R_{\text{OPT},b}^{\infty} = \inf_{\mathcal{A}} R_{\mathcal{A},b}^{\infty}$ . Our goal is to find for all values of  $b$  ( $b \geq 1$ ) an algorithm with asymptotic performance ratio close to  $R_{\text{OPT},b}^{\infty}$ .

**Previous Results** The classic online bin packing problem was first investigated by Ullman [14]. He showed that the FIRST FIT algorithm has performance ratio  $\frac{17}{10}$ . This result was then published in [6]. Johnson [7] showed that the NEXT FIT algorithm has performance ratio 2. Yao showed that REVISED FIRST FIT has performance ratio  $\frac{5}{3}$ . Currently the best known lower bound is 1.54014, due to van Vliet [15].

Define  $u_1 = 2, u_{i+1} = u_i(u_i - 1) + 1$ , and  $h_{\infty} = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103$ . Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to  $h_{\infty}$  [9]. They further showed that no bounded space online algorithm achieves a performance ratio less than  $h_{\infty}$  [9]. In addition, they developed the REFINED HARMONIC algorithm, which they showed to have a performance ratio of  $\frac{273}{228} < 1.63597$ . The next improvements were MODIFIED HARMONIC and MODIFIED HARMONIC 2. Ramanan, Brown, Lee and Lee showed that these algorithms have performance ratios of  $\frac{538}{333} < 1.61562$  and  $\frac{239091}{148304} < 1.61217$ , respectively [12]. Currently, the best known upper bound is 1.58889 due to Seiden [13].

Bin packing with resource augmentation was first studied by Csirik and Woeginger [4]. They give an optimal bounded space algorithm. Naturally, its asymptotic performance ratio is strictly decreasing as a function of the bin size of the online algorithm. Some preliminary general lower bounds for bin packing with resource augmentation were given in [5]. In Section 7, we will compare them to our new lower bounds.

**Our Results** In this paper, we first present new lower bounds for the online bin packing problem in the resource augmentation model by using improved sequences. For  $b \geq 2$ , our lower bounds show that the best bounded space algorithm is very close to optimal (among unbounded space algorithms).

We use the intuition gained from these lower bounds to develop new algorithms for this model, specifically for the case  $b \in [1, 2)$ . We introduce a general method which extends many previously studied algorithms for bin packing. This method takes the online bin size  $b$  as a parameter. We study four instances of the general method, each of our algorithms performs well for a different interval of values of  $b$ . By partitioning the interval  $[1, 2)$  in four sub-intervals and using the most appropriate algorithm on each sub-interval, we give upper bounds that improve upon the bounds from [4] on the entire interval. That is, these algorithms are better than the best possible bounded space algorithm.

Our analysis technique extends the general packing algorithm analysis technique developed by Seiden [13]. Specifically, unlike previous algorithms which pack the relatively small items by a very simple heuristic (Next-Fit, or any fit), we combine small items with large items in the same bins in order to achieve good performance (see the algorithms SMH and TMH).

## 2 Lower bounds - general structure

We first consider the question of lower bounds. Prior to this work, only a very simple general (non-bounded space) lower bound for resource-augmented online bin packing was known [5].

Our method follows along the lines laid down by Liang, Brown and Van Vliet [1, 10, 15]. We give some unknown online bin packing algorithm  $\mathcal{A}$  one of  $k$  possible different inputs. These inputs are defined as follows: Let  $\varrho = s_1, s_2, \dots, s_k$  be a sequence of *item sizes* such that  $0 < s_1 < s_2 < \dots < s_k \leq 1$ . Let  $\epsilon$  be a small positive constant. We define  $\sigma_0$  to be the empty input. Input  $\sigma_i$  consists of  $\sigma_{i-1}$  followed by  $n$  items of size  $s_i + \epsilon$ . Algorithm  $\mathcal{A}$  is given  $\sigma_i$  for some  $i \in \{1, \dots, k\}$ .

A *pattern* with respect to  $\varrho$  is a tuple  $p = \langle \text{size}(p), p_1, \dots, p_k \rangle$  where  $\text{size}(p)$  is a positive real number and  $p_i, 1 \leq i \leq k$  are non-negative integers such that  $\sum_{i=1}^k p_i s_i < \text{size}(p)$ . Intuitively, a pattern describes the contents of some bin of capacity  $\text{size}(p)$ . Define  $\mathcal{P}(\varrho, \beta)$  to be the set of all patterns  $p$  with respect to  $\varrho$  with  $\text{size}(p) = \beta$ . We write  $\mathcal{P}_{\text{OPT}}(\varrho) = \mathcal{P}(\varrho, 1)$  and  $\mathcal{P}_{\mathcal{A}}(\varrho) = \mathcal{P}(\varrho, b)$ . Note that these sets are necessarily finite. Given an input sequence of items, an algorithm is defined by the numbers and types of items it places in each of the bins it uses. Specifically, any online algorithm is defined by a function  $\Phi : \mathcal{P}_{\mathcal{A}}(\varrho) \mapsto \mathbb{R}_{\geq 0}$ . The algorithm uses  $\Phi(p)$  bins containing items as described by the pattern  $p$ . We define  $\phi(p) = \Phi(p)/n$ .

Consider the function  $\Phi$  that determines the packing used by online algorithm  $\mathcal{A}$  uses for  $\sigma_k$ . Since  $\mathcal{A}$  is online, the packings it uses for  $\sigma_1, \dots, \sigma_{k-1}$  are completely determined by  $\Phi$ . We assign to each pattern a *class*, which is defined as  $\text{class}(p) = \min\{i \mid p_i \neq 0\}$ . Intuitively, the class tells us the first sequence  $\sigma_i$  which results in some item being placed into a bin packed according to this pattern. I.e. if the algorithm packs some bins according to a pattern which has class  $i$ , then these bins will contain one or more items after  $\sigma_i$ . Define  $\mathcal{P}_{\mathcal{A}}(\varrho, i) = \{p \in \mathcal{P}_{\mathcal{A}}(\varrho) \mid \text{class}(p) \leq i\}$ . Then if  $\mathcal{A}$  is determined by  $\Phi$ , its cost for  $\sigma_i$  is simply  $n \sum_{p \in \mathcal{P}_{\mathcal{A}}(\varrho, i)} \phi(p)$ . Since the algorithm must pack every item, we have the following constraints

$$n \sum_{p \in \mathcal{P}(\varrho)} \phi(p) p_i \geq n, \quad \text{for } 1 \leq i \leq k.$$

For a fixed  $n$ , define  $\chi_i(n)$  to be the optimal offline cost for packing the items in  $\sigma_i$ . The following lemma gives us a method of computing the optimal offline cost for each sequence:

**Lemma 1 ([5])** For  $1 \leq i \leq k$ ,  $\chi^* = \lim_{n \rightarrow \infty} \chi_i(n)/n$  exists and is the value of the linear program: Minimize  $\sum_{p \in \mathcal{P}_{\text{OPT}}(\varrho, i)} \phi(p)$  subject to  $1 \leq \sum_{p \in \mathcal{P}_{\text{OPT}}(\varrho)} \phi(p) p_j$ , for  $1 \leq j \leq i$ , over variables  $\chi_i$  and  $\phi(p), p \in \mathcal{P}_{\text{OPT}}(\varrho)$ .

Given the construction of a sequence, we need to evaluate

$$c = \min_{\mathcal{A}} \max_{i=1, \dots, k} \limsup_{n \rightarrow \infty} \frac{\text{cost}_{\mathcal{A}}(\sigma_i)}{\chi_i(n)}.$$

As  $n \rightarrow \infty$ , we can replace  $\chi_i(n)/n$  by  $\chi_i^*$ . Once we have the values  $\chi_1^*, \dots, \chi_k^*$ , we can readily compute a lower bound for our online algorithm:

**Lemma 2** The optimal value of the linear program: Minimize  $c$  subject to

$$\begin{aligned} c &\geq \frac{1}{\chi_i^*} \sum_{p \in \mathcal{P}_{\mathcal{A}}(\varrho, i)} \phi(p), & \text{for } 1 \leq i \leq k; \\ 1 &\leq \sum_{p \in \mathcal{P}_{\mathcal{A}}(\varrho)} \phi(p) p_i, & \text{for } 1 \leq i \leq k; \end{aligned} \tag{1}$$

over variables  $c$  and  $\phi(p), p \in \mathcal{P}_{\mathcal{A}}(\varrho)$ , is a lower bound on the asymptotic performance ratio of any online bin packing algorithm.

**Proof.** For any fixed  $n$ , any algorithm  $\mathcal{A}$  has some  $\Phi$  which must satisfy the second constraint. Further,  $\Phi$  should assign an integral number of bins to each pattern. However, this integrality constraint is relaxed, and  $\sum_{p \in \mathcal{P}_{\mathcal{A}}(\varrho, i)} \phi(p)$  is  $1/n$  times the cost to  $\mathcal{A}$  for  $\sigma_i$  as  $n \rightarrow \infty$ . The value of  $c$  is just the maximum of the performance ratios achieved on  $\sigma_1, \dots, \sigma_k$ .  $\square$

### 3 Lower bound sequences: which inputs are hard to handle?

First of all, it can be seen that items of the form  $b/i + \epsilon$ , where  $\epsilon > 0$  is very small and  $i$  is an integer, are difficult to pack by any algorithm in the sense that if you pack only items of one such size in a bin, you leave almost  $b/i$  empty space in that bin. Moreover, within this class of items, it can be seen that the largest items are the worst in this sense: packing an item of size  $b/2 + \epsilon$  (and nothing else) into a bin leaves almost half that bin empty, but packing 99 items of size  $b/100 + \epsilon$  into a bin leaves only a little less than  $b/100$  empty space.

It is for these reasons that all our lower bound sequences start with the smallest items (so that the online algorithm has to take into account that larger and more difficult items may appear later, and thus has to reserve some room for them while packing the ‘easier’ items), and use only items of the form  $b/i + \epsilon$ . We will not explicitly mention the added  $\epsilon$  when discussing various item sizes below.

To determine lower bounds, we first extended the idea of a “greedy” sequence that was also used to give the best known lower bound for the standard online bin packing problem [1, 10, 15]. Interestingly, unlike these papers and the previous paper on resource augmentation in bin packing [4], it turns out to be insufficient to simply consider the “standard” greedy sequence that we will define first. More sequence and arguments are required to get a good lower bound.

A greedy sequence is built as follows. We start by taking the largest possible number of the form  $b/i$  that is less than 1, where  $i$  is a positive integer. If  $b \in [1, 2)$ , this is always the number  $b/2$ . Then we repeatedly add the largest possible  $b/i_j$  ( $i_j$  is an integer) that fits in the remaining space after packing all previous items

in a bin of size 1. That is, for  $b \in [1, 2)$ , the second item  $b/i_2$  has size strictly less than  $1 - b/2$ . We repeat this until the item found is sufficiently small.

We also use several modified greedy sequences. The reason to do that is that unlike the construction for bounded space algorithms [4], we need to consider the optimal offline packing of subsequences and not only of the complete sequence. For some intervals of values of  $b$ , some greedy items  $b/i_j$  are inconvenient to pack into bins of size 1, and a better lower bound can be proved by choosing such items differently. This is not an issue in the model without resource augmentation, where both algorithms deal with the same bin size.

The first modified sequence picks the item  $b/(i_2 + 1)$  instead of  $b/i_2$ , and continues greedily from that point. Another natural choice is  $b/(i_2 + 2)$  instead of  $b/i_2$ , but that does not improve the bounds. A second modified sequence keeps the first two items picked greedily,  $b/2$  and  $b/i_2$ , but it picks  $b/(i_3 + 1)$  as the next item, and continues greedily. A third version picks both the second and third item using this non-greedy method.

The lower bound is then constructed in the standard fashion: we have constructed a sequence of item sizes  $b/2, b/i_2, b/i_3, \dots, b/i_j$ . We invert the order of this sequence and start with the smallest item size. The inputs  $\sigma_i$  are then constructed as described above, where  $\epsilon$  is chosen in such a way that a set which consists of one instance of each item can be placed together in a single bin of size 1.

For  $b \geq 2$  we use only the basic greedy sequence.

## 4 The HARMONIC algorithm and variations

In this section we discuss the important HARMONIC algorithm [9] and possible variations on it. In the next section we will discuss the specific variations on HARMONIC that we have used in this paper.

The fundamental idea of these algorithms is to first classify items by size, and then pack an item according to its class (as opposed to letting the exact size influence packing decisions). For the classification of items, we need to partition the interval  $(0, 1]$  into subintervals. The standard HARMONIC algorithm uses  $n - 1$  subintervals of the form  $(1/(i + 1), 1/i]$  for  $i = 1, \dots, n - 1$  and one final subinterval  $(0, 1/n]$ . Each bin will contain only items from one subinterval (type). Items in subinterval  $i$  are packed  $i$  to a bin for  $i = 1, \dots, n - 1$  and the items in interval  $n$  are packed in bins using NEXT FIT.

A disadvantage of HARMONIC is that items of type 1, that is, the items larger than  $1/2$ , are packed one per bin, possibly wasting a lot of space in each single bin. To avoid this large waste of space, later algorithms used two extra interval endpoints, of the form  $\Delta > 1/2$  and  $1 - \Delta$ . Then, some small items can be combined in one bin together with an item of size  $\in (1/2, \Delta]$ . Items larger than  $\Delta$  are still packed one per bin as in HARMONIC. These algorithms furthermore use parameters  $\alpha^i$  ( $i = 3, \dots, n$ ) which represent the fraction of bins allocated to type  $i$  where the algorithm will reserve space for items  $\in (1/2, \Delta]$ . The remaining bins with items of type  $i$  still contain  $i$  items per bin.

**Example** MODIFIED HARMONIC (MH) is defined by  $n = 38$  (the number of intervals) and  $\Delta = 419/684$ .

$$\begin{aligned} \alpha^2 &= \frac{1}{9}; \\ \alpha^3 &= \frac{1}{12}; \\ \alpha^4 &= \alpha^5 = 0; \\ \alpha^i &= \frac{37 - i}{37(i + 1)}, \quad \text{for } 6 \leq i \leq 36; \\ \alpha^{37} &= \alpha^{38} = 0. \end{aligned}$$

The results of [12] imply that the asymptotic performance ratio of MH is at most  $\frac{538}{333} < 1.61562$ . (In the original definition,  $\Delta$  was used to denote  $1 - \Delta$ .)

In the current paper, we will use as interval endpoints the points of the form  $b/i$  (as long as they are below 1) instead of  $1/i$ , since items in  $(b/(i+1), b/i]$  can be placed exactly  $i$  to a bin in an (online) bin of size  $b$ . Moreover, sometimes we will also use points of the form  $\Delta, b - \Delta, 1 - b/2$  as interval endpoints, in order to combine items from different types where they would otherwise waste much space.

Note that for  $b \in [1, 2)$  we always have  $b/2 \leq 1$ . We now consider an algorithm  $\mathcal{A}$  that uses  $n$  basic intervals (some might be subdivided further):

$$\begin{aligned} I_{\mathcal{A}}^1 &= (b/2, 1] \\ I_{\mathcal{A}}^j &= (b/(j+1), b/j] \quad j = 2, \dots, n-1 \\ I_{\mathcal{A}}^n &= (0, b/n] \end{aligned}$$

In case  $\Delta$  is used as an endpoint, the interval  $I_{\mathcal{A}}^1 = (b/2, 1]$  is partitioned into two subintervals, which will be denoted by  $I_{\mathcal{A}}^{\Delta(2)} = (b/2, \Delta]$  and  $I_{\mathcal{A}}^{\Delta(1)} = (\Delta, 1]$ . ( $\Delta$  will always be chosen larger than  $b/2$ .) We will use two versions of algorithms, that are determined by whether they use  $b - \Delta$  or  $1 - b/2$  as an additional endpoint. We denote the largest possible size of an item of the smallest type by  $\varepsilon$ . This is  $b/n$  unless  $I_{\mathcal{A}}^n$  is divided further into two subintervals.

**Version 1** We use the endpoint  $b - \Delta$  (but not the endpoint  $1 - b/2$ ). Let  $j_{\Delta}$  be the integer such that  $b/(j_{\Delta} + 1) < b - \Delta \leq b/j_{\Delta}$ . Then  $I_{\mathcal{A}}^{j_{\Delta}}$  is partitioned into two subintervals, which will be denoted by  $I_{\mathcal{A}}^{\Delta(4)} = (b/(j_{\Delta} + 1), b - \Delta]$  and  $I_{\mathcal{A}}^{\Delta(3)} = (b - \Delta, b/j_{\Delta}]$ .

**Version 2** We use the endpoint  $1 - b/2$  (but not the endpoint  $b - \Delta$ ). Let  $j_{\Delta}$  be an integer such that  $b/(j_{\Delta} + 1) < 1 - b/2 \leq b/j_{\Delta}$ . In this version we always take  $n \geq j_{\Delta}$ .

If  $n \geq j_{\Delta} + 1$ , then  $I_{\mathcal{A}}^{j_{\Delta}}$  is partitioned into two subintervals, which will be denoted by  $I_{\mathcal{A}}^{\Delta(4)} = (b/(j_{\Delta} + 1), 1 - b/2]$  and  $I_{\mathcal{A}}^{\Delta(3)} = (1 - b/2, b/j_{\Delta}]$ .

Otherwise  $n_{\mathcal{A}} = j_{\Delta}$  and  $I_{\mathcal{A}}^n$  is partitioned into the two subintervals  $I_{\mathcal{A}}^{\Delta(4)} = (0, 1 - b/2]$  and  $I_{\mathcal{A}}^{\Delta(3)} = (1 - b/2, b/n]$ .

In both versions, the intervals are disjoint and cover  $(0, 1]$ .  $\mathcal{A}$  assigns each item a *type* depending on its size. An item of size  $s$  has type  $\tau_{\mathcal{A}}(s)$  where  $\tau_{\mathcal{A}}(s) = j \Leftrightarrow s \in I_{\mathcal{A}}^j$ . Note that either  $2 \leq j \leq n$  ( $j \neq j_{\Delta}$ ) or  $j = \Delta(i)$  for some  $1 \leq i \leq 4$ .

Note that if we place an item from the interval  $I_{\mathcal{A}}^{\Delta(2)}$  in a bin, the amount of space left over is at least  $b - \Delta$ . If possible, we would like to use this space to pack more items. To accomplish this, we assign each item a color, *red* or *blue*.  $\mathcal{A}$  attempts to pack red items with type  $I_{\mathcal{A}}^{\Delta(2)}$  items. For both versions, all items of types  $2, \dots, j_{\Delta} - 1$  and  $\Delta(k), k = 1, 2, 3$  (where applicable) are blue. Other items can be either red or blue.

To assign colors to items, the algorithm uses two sets of counters,  $e_{j_{\Delta}}, \dots, e_n$  and  $s_{j_{\Delta}}, \dots, s_n$ , all of which are initially zero. The counter  $s_{j_{\Delta}}$  counts the number of bins for items of type  $\Delta(4)$ , and the counter  $s_i$  keeps track of the total number of bins in which we packed items of type  $i$  for  $i = j_{\Delta} + 1, \dots, n$ . The counters  $e_i$  are defined analogously, but only count the number of bins containing *red* items of type  $\Delta(4)$  or  $i$ . These bins are also called red themselves.

For  $j_{\Delta} \leq i \leq n$ ,  $\mathcal{A}$  maintains the invariant  $e_i = \lfloor \alpha^i s_i \rfloor$ , i.e. the fraction of bins with type  $i$  items that contain red items is approximately  $\alpha^i$ . Recall that  $\alpha^i$  is defined only for  $j_{\Delta} \leq i \leq n$ . For each such interval,

at least one item can fit in a bin together with an item of size at most  $\Delta$  in a bin of size  $b$ . Moreover, for version 2 we combine only relatively small items with items of interval  $\Delta(2)$ , so in most cases several items fit together with the  $\Delta(2)$  item.

We now describe how blue and red items are packed. The packing of blue items is simple. For  $i < n$ , the number of items with sizes in  $(b/(i+1), b/i]$  which fit in a bin of size  $b$  is  $i$ . Blue items with such sizes are placed  $i$  in a bin, as in the HARMONIC algorithm. Note that the type of such an item is either  $i$  or  $\Delta(k)$  for some  $1 \leq k \leq 4$ . Small items (type  $n$ ) which are colored blue are packed into separate bins using NEXT FIT, again as in the HARMONIC algorithm.

For the red items, we consider the two versions of algorithms defined before separately.

**Version 1** One red item of type  $\Delta(4)$  can be combined with an item of type  $\Delta(2)$ . We define  $\gamma_{j_\Delta} = 1$ . For  $j_\Delta < j < n$ , the number of red items we will assign together with a type  $\Delta(2)$  item is  $\gamma_j = \lfloor j(b - \Delta)/b \rfloor$ . For type  $n$ , we treat the remaining space of  $b - \Delta$  in bins containing an item of type  $\Delta(2)$  as a bin, and use NEXT FIT to place red items in such bins. Clearly we can fill at least  $b - \Delta - b/n$  of this space by small red items.

**Version 2** If  $n = j_\Delta$ , it means that we combine only the smallest interval with items of type  $\Delta(2)$ . Then we can assign at least  $b - \varepsilon = 3b/2 - 1$  to blue items bins, and  $b - \Delta - \varepsilon = 3b/2 - \Delta - 1$  to red items bins. If  $n > j_\Delta$ , all the amounts are defined as for version 1, except for  $\gamma_{j_\Delta} = \lfloor (b - \Delta)/(1 - b/2) \rfloor$ .

We explain more precisely the method by which red items are packed with type  $\Delta(2)$  items. When a bin is opened, it is assigned to a *group*. If  $\varepsilon = b/n$ , the bin groups are named:

$$\Delta(1), \Delta(3), \Delta(4), 2, 3, \dots, j_\Delta - 1, j_\Delta + 1, \dots, n; \quad (2)$$

$$(\Delta(2), i), \quad \text{for } \alpha^i \neq 0, j_\Delta \leq i \leq n; \quad (3)$$

$$(\Delta(2), *); \quad (4)$$

$$(*, i), \quad \text{for } \alpha^i \neq 0, j_\Delta \leq i \leq n; \quad (5)$$

If  $\varepsilon = 1 - b/2$ , i.e. the smallest interval was partitioned, the bin groups are named:

$$\Delta(1), \Delta(3), \Delta(4), 2, 3, \dots, n - 1; \quad (6)$$

$$(\Delta(2), \Delta(4)); \quad (7)$$

$$(\Delta(2), *); \quad (8)$$

$$(*, \Delta(4)); \quad (9)$$

Bins from groups in (2) and (6) contain only blue items of the type they is named after. The closed bins all contain the maximum number of items they can have (explained earlier).

If the smallest interval was not partitioned, then for  $j_\Delta \leq i < n$ , a closed bin in group  $(\Delta(2), i)$  contains one type  $\Delta(2)$  item and  $\gamma_i$  type  $i$  items, and a closed bin in group  $(\Delta(2), n)$  contains one type  $\Delta(2)$  item and red items of total size at least  $b - \Delta - b/n$ . If the smallest interval was partitioned, a closed bin in group  $(\Delta(2), \Delta(4))$  contains red items of total size at least  $3b/2 - \Delta - 1$ . There is at most one open bin in any of these groups.

The group  $(\Delta(2), *)$  contains bins which hold a single blue item of type  $\Delta(2)$ . These bins are all open, as we hope to add red items to them later.

The group  $(*, j)$  contains bins which hold only red items of type  $j$ . Again, these bins are all open, but only one has fewer than  $\gamma_j$  items if  $j < n$ . For  $j = n$  only one bin can contain total size of less than  $b - \Delta - \varepsilon$  of red items of the last interval. We will try to add a type  $\Delta(2)$  item to these bins if possible.

We call bins in the last two group classes ((4) and (5), or (8) and (9)) *indeterminate*. Essentially, the algorithm tries to minimize the number of indeterminate bins, while maintaining all the aforementioned invariants. I.e. we try to place red and type  $\Delta(2)$  items together whenever possible; when this is not possible we place them in indeterminate bins in hope that they can later be combined.

On arrival of an item, it gets the same color as the previous item of the same type, if it can also fit into the same bin. Otherwise, we update the bins counter, and according to the counters, decide which color it gets.

## 5 Algorithms in this paper

After describing the general framework, we now describe the specific algorithms that we have designed. They are all instances of the general algorithm above. The first two algorithms, which deal with the case  $b \leq 4/3$ , handle the standard greedy lower bound sequence (defined in Section 3) well. The next two algorithms were designed to handle other input sequences better. We explain in Section 5.3 why the standard Harmonic approach is not so useful in the case  $b \geq 4/3$ .

### 5.1 Generalized Modified Harmonic (GMH)

This algorithm has the same structure as the regular MODIFIED HARMONIC, i.e.  $n = 38$ , and the same values of  $\alpha^i$ . The only difference is that the variable  $\Delta$  is adjusted to ensure that  $\Delta \in (b/2, 1)$  for  $b \in [1, 2)$ .

Specifically, we let  $\Delta$  grow linearly with the bin size until it reaches the value 1 for a bin size of 2, i.e.  $\Delta = 419/684 + 265(b - 1)/684$ . We applied this algorithm on the interval  $[1, 6/5)$ . This algorithm is of version 1 as we only modify  $\Delta$ .

### 5.2 Convenient Modified Harmonic (CMH)

On the interval  $[6/5, 4/3)$ , we focus on the items that could be packed together in one offline bin together with items of type 1, that is, items that are just larger than  $b/2$ . This was done specifically to handle the *greedy* input sequence, which starts with an item just larger than  $b/2$  and repeatedly adds an item of the form  $b/i_j + \varepsilon$  such that all items together fit into a bin of size  $b$ .

Our algorithm is of version 1 and does the following. Let

$$k = \left\lfloor \frac{1}{1 - b/2} \right\rfloor.$$

This means that the largest items that can be packed together with an item of size  $b/2$  in a single bin of size 1 are in the interval  $(1/(k + 1), 1/k]$  (possibly not every size in this interval can be so packed). Let  $\Delta = (k - 1)b/k$ . Note that in the interval of  $b$  we consider, we always have  $k = 3$  and hence  $\Delta = 2b/3$ . Note that  $b - \Delta = b/k$  and therefore  $I^{\Delta(3)} = \emptyset$ .

Our choice of  $\Delta$  ensures that items of type  $\Delta(2)$ , with sizes in  $(b/2, (k - 1)b/k]$ , can be packed very well together with items of type  $k$ , with sizes in  $(b/(k + 1), b/k]$ , in our case this is  $(b/4, b/3]$ . In the discussed interval we have  $b/2 + b/4 < 1$ , so in the optimal packing such items could also be together in one bin. The



choice of  $n = 38$  is as in GMH and the values  $\alpha^i$  are chosen by experimenting. The values we used are

$$\begin{aligned}\alpha^3 &= \frac{1}{8}; \\ \alpha^4 &= \frac{1}{10}; \\ \alpha^5 &= 0; \\ \alpha^i &= \frac{37-i}{37(i+4)}, \quad \text{for } 6 \leq i \leq 36; \\ \alpha^{37} &= \alpha^{38} = 0.\end{aligned}$$

### 5.3 Small Modified Harmonic (SMH)

On the interval  $[4/3, 12/7 \approx 1.7143)$ , it becomes more important how to pack smaller items (relative to  $b$ ). We define  $\Delta = 1$ , and  $n = 12$ . Thus  $I^{\Delta(1)} = \emptyset$ . Note that we use the second version of the algorithm, which means that in marked contrast to all other previously defined variations on Harmonic that the authors are aware of, we do not take  $\alpha_{12} = 0$ , that is, we pack some of the smallest items together with the large items.

We illustrate the reason. Consider a bin of size  $3/2$ . Taking  $\Delta = 1$  leaves a space of  $1/2$  in a bin. This space could be used to accommodate an item of size  $b/3 = 1/2$ . However, items of size in  $(b/4, b/3]$ , when packed three to a bin, occupy at least  $3b/4 = 9/8 > 1$ . Considering an offline packing we can see that such items do not fit together with an items of type  $\Delta(2)$ . Therefore there is no reason to improve their packing which is already relatively good.

However, items that do fit together with type  $\Delta(2)$  items do need to be packed more carefully (partly red and partly blue), including the ones from the last interval, since they can be combined in an offline packing. We determine the largest item type that OPT could pack together with an item in  $(b/2, 1]$  (i.e. the smallest  $i$  such that  $b/i \leq 1 - b/2$ ). Larger items are packed according to Harmonic, while a fraction of these smaller items are reserved to be packed together with an item of type  $\Delta(2)$ , i.e. in  $(b/2, 1]$ .

We explain how to fix the values  $\alpha^i$  for this algorithm in Section 6.2.

### 5.4 Tiny Modified Harmonic (TMH)

On the interval  $[12/7, 2)$ , it turns out that it is crucial to pack the smallest items better than with Harmonic. All other items are packed in their own bins according to Harmonic. We use the second version of the algorithm. We use  $\Delta = 1$  (so  $I^{\Delta(1)} = \emptyset$ ) and let  $n = j_\Delta$ .

In other words, we determine the number of intervals that we use in such a way that  $1 - b/2 \in (b/(n+1), b/n]$ . The smallest interval boundary of the form  $b/i$  is just larger than  $1 - b/2$  (or equal to it). This ensures that in the optimal packing, only items of the smallest type could be packed together with large items with size in  $(b/2, 1]$ . We use  $\alpha_{j_\Delta} = (2b-2)/(4-b)$ .

It would be possible to improve very slightly using the algorithm SMH with more intervals, but the number of intervals required grows without bound as  $b$  approaches 2, and it becomes infeasible to calculate all the patterns.

## 6 Analysis

An algorithm for a given bin size  $b$  can be used without change for any bin size  $c \geq b$ , and will have the same performance ratio since for any given sequence, the offline optimal packing and the cost of the algorithm

remain unchanged. This means that the function  $R_{\text{OPT},b}^\infty$  is monotonically decreasing in  $b$ . This property allows us to give bounds on an interval by *sampling* a large but finite number of points. An upper bound for the bin size  $b$  holds for  $b + \gamma$  for any  $\gamma > 0$ . A lower bound for the bin size  $b$  holds for a bin size  $b - \gamma$  for any  $\gamma > 0$ .

## 6.1 Weighting functions

The type of algorithm described in Section 4 can be analyzed using the method of weighting systems developed in [13]. The full generality of weighting systems is not required here, so we adopt a slightly different notation than that used in [13], and restrict ourselves to a subclass of weighting systems.

A weighting system for an algorithm  $\mathcal{A}$  is a pair  $(W_{\mathcal{A}}, V_{\mathcal{A}})$ .  $W_{\mathcal{A}}$  and  $V_{\mathcal{A}}$  are *weighting functions* which assign each item  $p$  a real number based on its size. The weighting functions for an algorithm  $\mathcal{A}$  are defined as follows.

If  $\varepsilon = 1 - b/2$ , the only value of  $\alpha^i$  which is not zero is  $\alpha^{j_\Delta}$ . The weighting functions are defined as follows.

Type of item $p$	$W_{\mathcal{A}}(p)$	$V_{\mathcal{A}}(p)$
$\Delta(1)$	1	1
$\Delta(2)$	1	0
$k \in \{2, 3, \dots, j_\Delta - 1\}$	$1/k$	$1/k$
$\Delta(3)$	$1/j_\Delta$	$1/j_\Delta$
$\Delta(4)$	$\frac{p(1 - \alpha^{j_\Delta})}{3b/2 - 1 - \Delta\alpha^{j_\Delta}}$	$\frac{p}{3b/2 - 1 - \Delta\alpha^{j_\Delta}}$

For the cases that  $\varepsilon = b/n$  we define the functions differently.

Type of item $p$	$W_{\mathcal{A}}(p)$	$V_{\mathcal{A}}(p)$
$\Delta(1)$	1	1
$\Delta(2)$	1	0
$k \in \{2, 3, \dots, j_\Delta - 1\}$	$1/k$	$1/k$
$\Delta(3)$	$1/j_\Delta$	$1/j_\Delta$
$\Delta(4)$	$\frac{1 - \alpha^j}{\gamma_{j_\Delta} \alpha_\Delta^j + j_\Delta(1 - \alpha_\Delta^j)}$	$\frac{1}{\gamma_{j_\Delta} \alpha_\Delta^j + j_\Delta(1 - \alpha_\Delta^j)}$
$k \in \{j_\Delta + 1, \dots, n - 1\}$	$\frac{\gamma_k \alpha^k + k(1 - \alpha^k)}{p(1 - \alpha^n)}$	$\frac{1}{\gamma_k \alpha^k + k(1 - \alpha^k)}$
$n$	$\frac{p(1 - \alpha^n)}{b - b/n - \Delta\alpha^n}$	$\frac{p}{b - b/n - \Delta\alpha^n}$

The following lemma follows directly from Lemma 4 of [13]:

**Lemma 3** *For all  $\sigma$ , we have*

$$\text{cost}_{\mathcal{A}}(\sigma) \leq \max \left\{ \sum_{p \in \sigma} W_{\mathcal{A}}(p), \sum_{p \in \sigma} V_{\mathcal{A}}(p) \right\} + O(1).$$

So the cost to  $\mathcal{A}$  can be upper bounded by the weight of items in  $\sigma$ , and the weight is independent of the order of items in  $\sigma$ .

We give a short intuitive explanation of the weight functions and Lemma 3: Consider the final packing created by an algorithm  $\mathcal{A}$  on some input  $\sigma$ . In this final packing, let  $r$  be the number of bins containing red

items, let  $b_1$  be the number of type  $\Delta(2)$  items, and let  $b_2$  be the number of bins containing blue items of type other than  $\Delta(2)$ . The total number of bins is just  $\max\{r, b_1\} + b_2 = \max\{r + b_2, b_1 + b_2\}$ . We have chosen our weighting functions so that  $\sum_{p \in \sigma} W_{\mathcal{A}}(p) = b_1 + b_2 + O(1)$  and  $\sum_{p \in \sigma} V_{\mathcal{A}}(p) = r + b_2 + O(1)$ . In both  $W_{\mathcal{A}}$  and  $V_{\mathcal{A}}$ , the weight of a blue item of type other than  $\Delta(2)$  is just the fraction of a bin that it occupies.  $W_{\mathcal{A}}$  counts type  $\Delta(2)$  items, but ignores red items.  $V_{\mathcal{A}}$  ignores type  $\Delta(2)$  items, but counts bins containing red items. For a formal proof, we refer the reader to [13].

Let  $f$  be some function  $f : (0, 1] \mapsto \mathbb{R}^+$ .

**Definition 6.1**  $\mathcal{P}(f)$  is the mathematical program: Maximize  $\sum_{x \in X} f(x)$  subject to  $\sum_{x \in X} x \leq 1$ , over all finite sets of real numbers  $X$ . In an abuse of notation, we also use  $\mathcal{P}(f)$  to denote the value of this mathematical program.

Intuitively, given a weighting function  $f$ ,  $\mathcal{P}(f)$  upper bounds the amount of weight that can be packed in a single bin. It is shown in [13] that the performance ratio of  $\mathcal{A}$  is upper bounded by  $\max\{\mathcal{P}(W_{\mathcal{A}}), \mathcal{P}(V_{\mathcal{A}})\}$ . The value of  $\mathcal{P}$  is easily determined using a branch and bound procedure very similar to those in [13, 5].

## 6.2 Choice of values $\alpha^i$ for SMH

To choose the values of  $\alpha^i$  in the algorithm SMH we use the following idea. We would like to balance the total weight of two particular offline packings. The first offline packing contains one item of interval  $\Delta(2)$  and smaller items of type  $i$  (here the weight is maximized by considering the weight function  $W_{\mathcal{A}}$ ). The second offline packing contains only items of type  $i$ , and we use  $V_{\mathcal{A}}$  to determine the maximum weight.

In order to balance these weights, we define the *expansion* of type  $i$  to be the maximum ratio of weight to size of an item of type  $i$ . Let  $EV(i)$  be the expansion according to  $V_{\mathcal{A}}$  and  $EW(i)$  be the expansion according to  $W_{\mathcal{A}}$ . We would like to have

$$EV(i) = 1 + (1 - b/2)EW(i).$$

This implies

$$\alpha^i = \frac{S - b/2}{S - s + 1 - b/2},$$

where  $S$  is the minimal occupied area in a closed bin containing blue items of type  $i$  and  $s$  is the minimal occupied area by red items of interval  $I$  in a closed bin.

Note that this computation is not entirely accurate for all types, as it is not always possible to fill a bin of size 1 or of size  $1 - b/2$  completely with items of the largest expansion. However, the interval which affects the asymptotic performance ratio the most is  $(0, \varepsilon]$ .

## 6.3 Analysis of TMH

The simple structure of TMH allows an analytical solution. For this algorithm, we do not need to solve mathematical programs, but can instead calculate the asymptotic worst case performance directly, as follows.

For all types but the smallest and the largest, the weight of an item of size  $x$  is at most  $x$ . The reason for this is that they are packed according to Harmonic, and TMH can fit at least the same number of items per bin as OPT can. To get a bin of weight more than 1, there must be some items of the first or the last type.

The upper bound of the last interval is  $1 - b/2$ , denoted by  $\varepsilon$ . Only items in this interval can be packed together with a type  $\Delta(2)$  item in one bin.

Recall that the algorithm uses a parameter  $\alpha = \alpha^{j\Delta}$  that determines how the small items are packed. The algorithm maintains the invariant that a fraction  $\alpha$  of the bins containing small items are red and have room for a type  $\Delta(2)$  item. The total size of all the small items in such a bin is at least  $b - 1 - \varepsilon$ . The rest of these bins are blue and contain a volume of at least  $b - \varepsilon$ . There are two cases.

**Case 1** There is no item of type  $\Delta(2)$ . If TMH uses  $k$  bins to pack all items of type  $\Delta(4)$  (the last type), then  $\alpha k$  bins are red and contain a minimum volume of  $b - 1 - \varepsilon$  each;  $(1 - \alpha)k$  bins contain a minimum total volume of  $b - \varepsilon$  of small blue items each. Thus  $k$  bins contain a total volume of at least  $\alpha k(b - 1 - \varepsilon) + (1 - \alpha)k(b - \varepsilon) = k(b - \alpha - \varepsilon)$ , in other words each bin contains on average a volume of at least  $b - \alpha - \varepsilon$ . The worst case is that all the items are small. Since an offline bin can contain one unit of such items, this gives an asymptotic performance ratio of  $1/(b - \alpha - \varepsilon)$ . Note that this is consistent with the definition of the function  $V_{\mathcal{A}}$  for this case.

**Case 2** There is an item of type  $\Delta(2)$ . We are interested in the case that its weight is 1, i.e. in the weights according to the function  $W_{\mathcal{A}}$ . The large item is of size at least  $b/2$ . The weight in a bin that contains such an item is maximized by filling up the bin with items of type  $\Delta(4)$ . The remaining space in the offline bin is exactly  $1 - b/2$ . In this case, TMH only needs “to pay” for the blue bins. It packs a volume of  $k(b - \alpha - \varepsilon)$  using only  $(1 - \alpha)k$  blue bins. The total weight according to  $W_{\mathcal{A}}$  is  $1 + \frac{1 - \alpha}{b - \alpha - \varepsilon}(1 - b/2)$ .

Balancing the weights gives that the best choice is  $\alpha = \frac{2b-2}{4-b}$  and a ratio of  $1/(b - \alpha - \varepsilon) = (2b - 8)/(3b^2 - 10b + 4)$ .

## 7 Results

As mentioned in Section 6, we can determine valid upper and lower bounds on the asymptotic performance ratio for this problem on any interval by sampling a finite number of points. In fact, since we have given an analytical solution for the algorithm TMH, it is not necessary to do any sampling for the upper bound on the interval  $(12/7, 2]$ .

On the remaining intervals, we used a computer program to solve the associated mathematical program  $\mathcal{P}$  for many specific values of  $b$  (we sampled integer multiples of  $\frac{1}{1000}$ ) and whichever algorithm is used for that value of  $b$ .

We also used a computer program to generate lower bounds for 1,000 values of  $b$  between 1 and 2. This program can be found at <http://algo2.iti.uni-karlsruhe.de/vanstee/program/>. There were some values of  $b$  where all lower bound sequences that we used gave a worse lower bound than we had already found for some higher value of  $b$ . However, a lower bound of  $c$  for a value  $b_0$  also implies a lower bound of  $c$  for all values  $1 \leq b \leq b_0$  as stated before. Therefore, whenever we found a lower bound that was worse than one that was found for some higher value of  $b$ , we instead use this higher bound. This explains the small intervals in the graph where the lower bound is constant.

Our results are summarized in the two Figures 1 and 2. The horizontal axis is the size of the online bin, and the vertical axis is the asymptotic performance ratio. For comparison, we have included the graph of the bounded space upper bound (which matches the bounded space lower bound).

It can be seen that for all bin sizes between 1 and 2, we have given substantial improvements on the bounded space algorithm, which was the best known algorithm for this problem so far. In particular, for  $b \geq 1.6$ , the upper bound of our algorithms is less than half as much removed from the (new) lower bound as the previous (bounded space) upper bound was.

The lower bounds from [5] were also significantly improved: for instance for  $b = 6/5$ , the lower bound was improved from less than 1.18 to above 1.34, and for  $b \geq 3/2$ , the previous lower bound was less than 0.8.

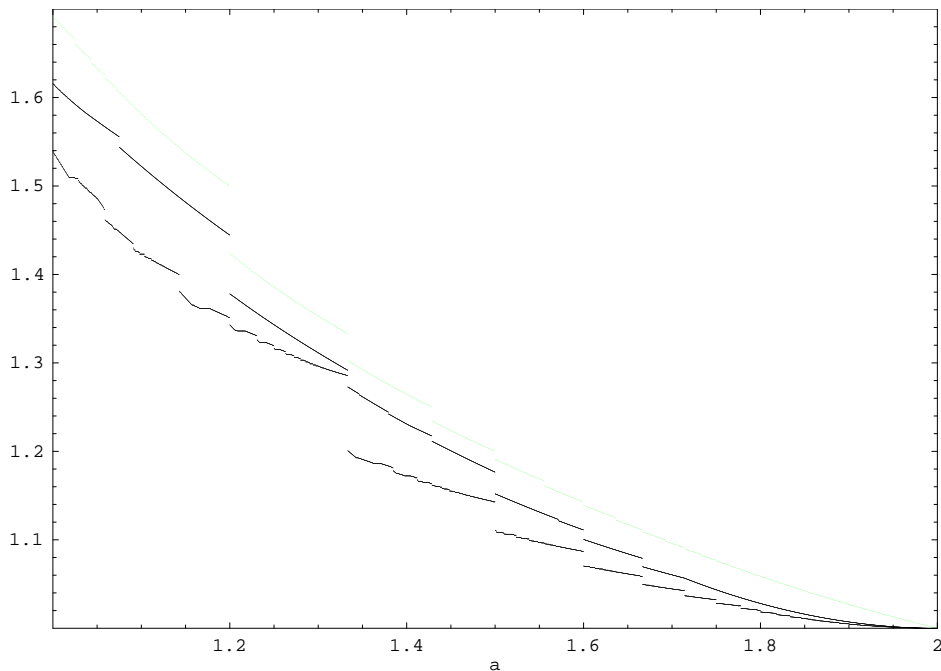


Figure 1: The lower bound (lowest graph), upper bound (middle), and bounded space bound (highest). Horizontal axis is size of online bin, vertical axis is asymptotic performance ratio.

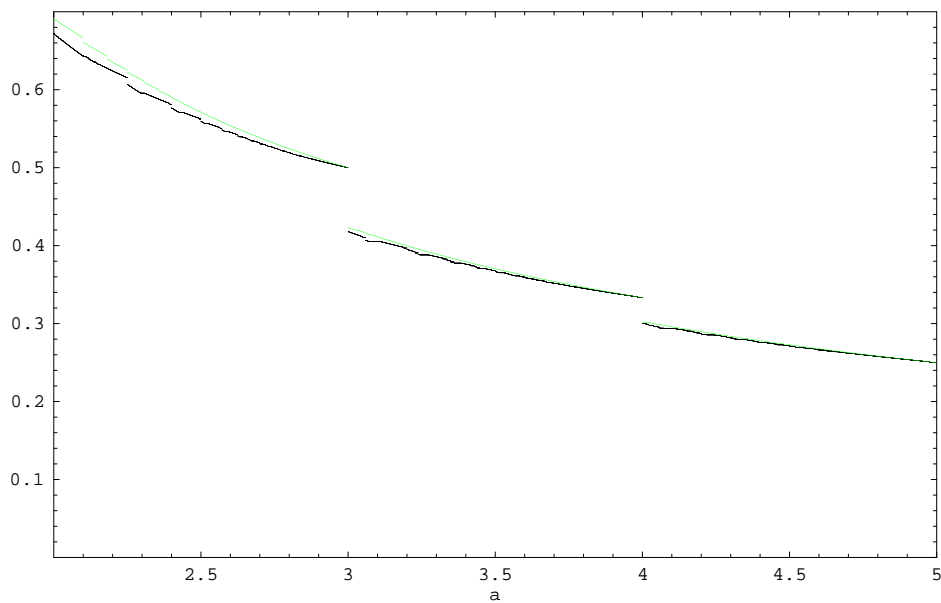


Figure 2: The lower bound (lowest graph) and bounded space bound on  $[2, 5]$ . Axes as in previous figure.

## 8 Conclusions

We have improved all known results for bin packing with resource augmentation. The remaining gap between the upper and lower bound is at most 7% for all  $b \geq 1$ , at most 3% for  $b \geq 1.6$ , and the bounds are nearly tight for  $b \geq 2$ .

The fact that the online algorithm deals with a different bin size than the offline algorithm complicates the design of algorithms. It is in particular critical to deal with sequences that can be packed with very little loss by the offline algorithm, while other sequences like the standard greedy sequence become less important since the optimal offline algorithm can not always pack subsequences of that input well enough to give a good lower bound.

An interesting observation which follows from our research is that in some cases, it can be helpful to pack very small items in the input together with large items in one bin. To our knowledge, this is the first time that this approach has worked for any bin packing problem. It is an open problem whether this can help for standard bin packing too.

## References

- [1] Donna J. Brown. A lower bound for on-line one-dimensional bin packing algorithms. Technical Report R-864, Coordinated Sci. Lab., Urbana, Illinois, 1979.
- [2] 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.
- [3] János Csirik and Gerhard J. Woeginger. On-line packing and covering problems. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 147–177. Springer-Verlag, 1998.
- [4] János Csirik and Gerhard J. Woeginger. Resource augmentation for online bounded space bin packing. *Journal of Algorithms*, 44(2):308–320, 2002.
- [5] Leah Epstein, Steve S. Seiden, and Rob van Stee. New bounds for variable-sized and resource augmented online bin packing. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proc. 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 306–317. Springer, 2002.
- [6] 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.
- [7] David S. Johnson. Fast algorithms for bin packing. *J. Comput. Systems Sci.*, 8:272–314, 1974.
- [8] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47:214–221, 2000.
- [9] C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *J. ACM*, 32:562–572, 1985.
- [10] F. M. Liang. A lower bound for online bin packing. *Inform. Process. Lett.*, 10:76–79, 1980.

- [11] Cynthia Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, pages 163–200, 2002.
- [12] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. Online bin packing in linear time. *J. Algorithms*, 10:305–326, 1989.
- [13] Steve S. Seiden. On the online bin packing problem. *Journal of the ACM*, 49(5):640–671, 2002.
- [14] Jeffrey D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton University, Princeton, NJ, 1971.
- [15] André van Vliet. An improved lower bound for online bin packing algorithms. *Inform. Process. Lett.*, 43:277–284, 1992.