Software ENgineering

SEN

New bounds for multi-dimensional packing

S. Seiden, R. van Stee

REPORT SEN-R0132 DECEMBER 31, 2001

ISSN 1386-369X

New Bounds for Multi-dimensional Packing

Steve Seiden*
Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803, U.S.A.
sseiden@acm.org.

Rob van Stee†
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.
Rob.van.Stee@cwi.nl.

ABSTRACT
New upper and lower bounds are presented for a multi-dimensional generalization of bin packing called box packing. Several variants of this problem, including bounded space box packing, square packing, variable sized box packing and resource augmented box packing are also studied. The main results, stated for $d = 2$, are as follows: A new upper bound of 2.66013 for online box packing, a new $14/9 + \varepsilon$ polynomial time offline approximation algorithm for square packing, a new upper bound of 2.43828 for online square packing, a new lower bound of 1.62176 for online square packing, a new lower bound of 2.28229 for bounded space online square packing and a new upper bound of 2.32571 for online two-sized box packing.

2000 Mathematics Subject Classification: 68W25, 68W40
1998 ACM Computing Classification System: F.2.2
Keywords and Phrases: bin packing, multi-dimensional, online, offline
Note: Work carried out under project SEN4 "Evolutionary Computation and Applied Algorithmics".

1. Introduction

Bin packing is one of the oldest and most well-studied problems in computer science [10, 5]. The study of this problem dates back to the early 1970's, when computer science was still in its formative phase—ideas which originated in the study of the bin packing problem have helped shape computer science as we know it today. The influence and importance of this problem are witnessed by the fact that it has spawned off whole areas of research, including the fields of online algorithms and approximation algorithms. In this paper, we study a natural generalization of bin packing, called box packing.

Problem Definition: Let $d \geq 1$ be an integer. In the *d-dimensional box packing* problem, we receive a sequence σ of *pieces* $p_1, p_2, \ldots, p_N$. We use the words piece and *item* synonymously. Each piece p has a fixed *size*, which is $s_1(p) \times \cdots s_d(p)$. I.e. $s_i(p)$ is the size of p in the ith dimension. We have an infinite number of *bins* each of which is a d-dimensional unit hyper-cube. Each piece must be assigned to a bin and a position $(x_1(p), \ldots, x_d(p))$, where $0 \leq x_i(p)$ and $x_i(p) + s_i(p) \leq 1$ for $1 \leq i \leq d$. Further, the positions must be assigned in such a way that no two items in the same bin overlap. A bin is *empty* if no piece is assigned to it, otherwise it is *used*. The goal is to minimize the number of bins used. Note that for $d = 1$, the box packing problem reduces to exactly the classic bin packing problem.

*This research was partially supported by the Research Competitiveness Subprogram of the Louisiana Board of Regents.

†Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

In this paper, we focus mainly on the case of $d = 2$. This allows for a simplification in notation. We say that item p has *width* $w(p) = s_1(p)$ and *height* $h(p) = s_2(p)$. We note that many of our results are more general, however, we focus on the two-dimensional case in order to avoid the cumbersome notation required by a more general treatment.

There are a number of variants of this problem which are of interest:

- In the *online* version of this problem, each piece must be assigned in turn, without knowledge of the next pieces.
- In the *square packing* problem we have the restriction that $h(p) = w(p)$ for all items p.
- In the *two-sized box packing* problem, bins have one of two sizes, either 1×1 or $1 \times z$. The algorithm chooses the size of a bin when it is allocated. The cost of a bin is equal to its area.
- In the *resource augmented box packing* problem, the algorithm is allowed to have larger bins than the adversary. The cost of each bin is one.
- In the *bounded space* variant, an algorithm has only a constant number of bins available to accept items at any point during processing. The bounded space assumption is a quite natural one, especially so in online box packing. Essentially the bounded space restriction guarantees that output of packed bins is steady, that the packer does not accumulate an enormous backlog of bins which are only output at the end of processing.

The offline versions of these problems are NP-hard, while even with unlimited computational ability it is impossible in general to produce the best possible solution online. We therefore consider both online and offline approximation algorithms.

The standard measure of algorithm quality for box packing is the *asymptotic performance ratio*, which we now define. For a given input sequence σ, let $\mathrm{cost}_{\mathcal{A}}(\sigma)$ be the number of bins used by algorithm $\mathcal{A}$ on σ. Let $\mathrm{cost}(\sigma)$ be the minimum possible number of bins used to pack pieces in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

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

In the case of a randomized algorithm we replace $\mathrm{cost}_{\mathcal{A}}(\sigma)$ with $\mathrm{E}[\mathrm{cost}_{\mathcal{A}}(\sigma)]$ in the preceding definition. Let $\mathcal{O}$ be the set of all online box packing algorithms. The *optimal asymptotic performance* ratio is defined to be

$$R^\infty_{\mathrm{OPT}} = \inf_{\mathcal{A}\in\mathcal{O}} R^\infty_{\mathcal{A}}.$$

Our goal is to find an algorithm with asymptotic performance ratio close to R^∞_{OPT}.

Previous Results: The classic online bin packing problem was first investigated by Johnson [20]. He showed that the Next Fit algorithm has performance ratio 2. Subsequently, it was shown by Johnson, Demers, Ullman, Garey and Graham that the First Fit algorithm has performance ratio $\frac{17}{10}$ [21]. Yao showed that Revised First Fit has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [34]. Brown and Liang independently improved this lower bound to 1.53635 [3, 26]. The lower bound currently stands at 1.54014, due to van Vliet [32]. Define

$$\pi_{i+1} = \pi_i(\pi_i - 1) + 1, \qquad \pi_1 = 2,$$

and

$$\Pi_\infty = \sum_{i=1}^{\infty} \frac{1}{\pi_i - 1} \approx 1.69103.$$

Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to Π_∞ [24]. A sequence of further results has brought the upper bound down to 1.58889 [24, 28, 29, 30].

While box packing is a natural next step from bin packing, the problem seems to be more difficult, and the number of results is smaller. The offline problem was introduced by Chung, Garey and Johnson [4]. The online problem was first investigated by Coppersmith and Raghavan [6], who give an algorithm based of NEXT FIT with performance ratio $\frac{13}{4} = 3.25$ for $d = 2$. Csirik, Frenk and Labbe [8] give an algorithm based on FIRST FIT with performance ratio $\frac{49}{16} = 3.0625$ for $d = 2$. The best result to date is that of Csirik and van Vliet [9]. They present an algorithm based on HARMONIC with performance ratio $(\Pi_\infty)^d$ for all $d \geq 2$ (2.85958 for $d = 2$). Unlike HARMONIC, this algorithm is not a bounded space algorithm. For bounded space algorithms, a lower bound of $(\Pi_\infty)^d$ is implied by [9]. Several lower bounds have been shown [16, 17, 33, 2]. The best lower bound for $d = 2$ is 1.907 [2], while the best lower bound for large d is less than 3.

For online square packing, even less is known. The following results are known for $d = 2$: Coppersmith and Raghavan [6] show an upper bound of $43/16 = 2.6875$ and a lower bound of $4/3$. The upper bound is improved to $100/39 < 2.56411$ by Fujita and Hada [15]. For the offline problem, Ferreira, Miyazawa and Wakabayashi give a 1.988-approximation algorithm [14].

For $d \geq 2$, as far as we know, there are no results for either online two-sized box packing or online resource augmented box packing.

Our Results: In this paper, we present a number of results for online and offline box and square packing:

- We show that if we have a one dimensional online bin packing algorithm $\mathcal{A}$ chosen from a certain class of algorithms (which we define precisely later) and the performance ratio of $\mathcal{A}$ is r, then we can construct an online box algorithm for $d = 2$ with performance ratio arbitrarily close to $r\Pi_\infty$. The class of admissible algorithms includes HARMONIC, REFINED HARMONIC and MODIFIED HARMONIC. Out of these, MODIFIED HARMONIC has the lowest performance ratio, namely 1.61562. This improves the upper bound for online box packing to 2.73220.
- We go on to examine a simple and natural randomized variant of this algorithm. The performance ratio of the randomized variant is shown to be at most 2.66013 for a particular algorithm $\mathcal{A}$. It is possible to de-randomize this algorithm, therefore 2.66013 is also an upper bound on the deterministic performance ratio.
- We show that no online box packing algorithm which uses $d-1$ open bins has finite performance ratio.
- For the offline square packing problem in two dimensions, we give an $14/9 + \varepsilon$ polynomial time approximation algorithm.
- For the online square packing problem in two dimensions, we show an upper bound of 2.43828.
- We show improved lower bounds for square packing for $d \geq 2$. For $d = 2$ we get a lower bound of 1.62176.
- We show the first lower bounds for bounded space online square packing for $d \geq 2$. For $d = 2$ we get a lower bound of 2.28229.
- For online two-sized packing with sizes 1×1 and $1 \times z$ we show that if the algorithm may chose z, then a performance ratio of 2.32571 is possible.
- We show the first upper bound for online resource augmented box packing.

These results are derived using a number of general techniques which most likely will yield further results.

2. The Geometric Next Fit Algorithm

The Next Fit algorithm [19, 20] is one of the simplest possible one-dimensional bin packing algorithms. We shall use Next Fit as a subroutine in our multi-dimensional algorithm.

Next Fit maintains a single *open* bin. If the current item fits into the open bin, it is placed there. Otherwise, the open bin is *closed* and a new open bin is allocated. Obviously, this algorithm is online, runs in linear time and uses constant space.

Let ϵ be a positive real number. We say that an item is *small* if its width is at most ϵ. Otherwise, the item is *large*. We pack items which are large and small differently. Specifically, we pack small items using an algorithm we call Geometric Next Fit, which we shall describe in this section. Geometric Next Fit uses as a sub-routine some online one-dimensional bin packing algorithm $\mathcal{A}$.

Geometric Next Fit is a generalization of Next Fit. The algorithm has a real parameter $\delta \in (0, 1)$. We shall assume that all items processed by Geometric Next Fit are small. We say that a piece p has class i if and only if $\epsilon(1-\delta)^{i-1} < w(p) \leq \epsilon(1-\delta)^i$. Since every item processed by Geometric Next Fit has $w(p) \leq \epsilon$, every item has a class $i \geq 0$. Every item is packed according to its class. We say that class i is *active* if we have processed at least one small item with class i.

An i *slice* is a box of height 1 and width $\epsilon(1-\delta)^i$. The *size* of an i slice is defined to be $\epsilon(1-\delta)^i$. We divide each bin used by the algorithm into several slices. In particular, we always have one bin which we call *fully open*. When we need to allocate a slice, we first try to allocate it from this bin. Let S be the sum of the sizes of the slices already allocated in the fully open bin. If the size of the new slice is at most $1 - S$, we allocate the slice in the fully open bin. Otherwise, we allocate a new bin and allocate the new slice within it. The new bin becomes fully open, while the old fully open bin becomes either *open* or *closed* as shall be explained later in our exposition. Intuitively, we are using Next Fit to allocate slices in bins.

We run an independent copy of $\mathcal{A}$ for each active class. We denote the copy of $\mathcal{A}$ associated with class i as $\mathcal{A}_{n+1,i}$, for reasons that shall become clear in Section 4. Pieces of class i are packed into i slices using algorithm $\mathcal{A}_{n+1,i}$. We can imagine that $\mathcal{A}_{n+1,i}$ does not know that items have a width, it treats height as size, and treats slices as bins.

As mentioned before, a bin other than the one fully open bin is either open or closed. Such a bin is defined to be open if it contains an open slice with respect to some algorithm $\mathcal{A}_{n+1,i}$. I.e. If there is the possibility that some algorithm $\mathcal{A}_{n+1,i}$ will place an item in an i slice contained in it. Otherwise, it is closed.

Note that while Next Fit is a bounded space algorithm, Geometric Next Fit is not, even if $\mathcal{A}$ is a bounded space algorithm. This is because Geometric Next Fit may have an unbounded number of active classes. In fact, it could potentially have $\Omega(N)$ active classes. During processing, we need to be able to find the class for the current item quickly. We can implement Geometric Next Fit to run in time $O(N \log N)$ by keeping track of the active classes using a balanced binary search tree.

In essence, Geometric Next Fit packs items into slices using $\mathcal{A}$, then packs slices into bins using Next Fit. In general, if we have some algorithms $\mathcal{A}$ and $\mathcal{B}$ for the one-dimensional bin packing problem, then we can use them to construct an algorithm for two dimensions as follows. We somehow classify items. We use $\mathcal{A}$ to pack items of the same class into slices, and then use $\mathcal{B}$ to pack slices into bins. In Section 4, we use this algorithm design paradigm to construct an algorithm for packing large items.

3. The Improved Harmonic Class of Algorithms

We shall also use an algorithm from the class of algorithms called Super Harmonic by Seiden [30]. These algorithms are all based on the ideas first used in Harmonic. More specifically, we use an algorithm from a subset of Super Harmonic which we call Improved Harmonic. This class includes Harmonic, Refined Harmonic and Modified Harmonic. We describe the class of Improved Harmonic algorithms in this section. We assume that the reader is familiar with the definition of Super Harmonic, and with the analysis of Super Harmonic given in [30].

The fundamental idea of all SUPER HARMONIC algorithms is to first classify items by size, and then pack an item according to its class (as opposed to directly letting size influence packing decisions). An instance $\mathcal{A}$ of the IMPROVED HARMONIC algorithm is described by a number of parameters: an integer $n_{\mathcal{A}} \geq 4$, a real number $\Delta_{\mathcal{A}} \in [\frac{1}{3}, \frac{1}{2}]$ and real numbers $\alpha^1_{\mathcal{A}} \ldots \alpha^n_{\mathcal{A}} \in [0,1]$. We define

$$\begin{aligned} t^1_{\mathcal{A}} &= 1 \\ t^2_{\mathcal{A}} &= 1 - \Delta \\ t^3_{\mathcal{A}} &= \tfrac{1}{2} \\ t^4_{\mathcal{A}} &= \Delta \\ t^i_{\mathcal{A}} &= \frac{1}{i-2} \qquad \text{for } 5 \leq i \leq n+1 \\ t^{n+2}_{\mathcal{A}} &= 0 \end{aligned}$$

We further define $\epsilon_{\mathcal{A}} = t^{n+1}_{\mathcal{A}}$. To facilitate the classification of items, we define the interval $I^j_{\mathcal{A}}$ to be $(t^{j+1}_{\mathcal{A}}, t^j_{\mathcal{A}}]$ for $j = 1, \ldots, n_{\mathcal{A}} + 1$. Note that these intervals are disjoint and that they cover $(0, 1]$. An IMPROVED HARMONIC algorithm $\mathcal{A}$ assigns each piece a *type* depending on its size. An item of size s has type $\tau_{\mathcal{A}}(s)$ where

$$\tau_{\mathcal{A}}(s) = j \Leftrightarrow s \in I^j_{\mathcal{A}}.$$

When it is clear that we are discussing algorithm $\mathcal{A}$, we shall drop the $\mathcal{A}$ subscript.

In addition to being assigned a type, each item of type $i \leq n$ is assigned a color, *red* or *blue*. The algorithm uses two sets of counters, $e_1, \ldots, e_n$ and $s_1, \ldots, s_n$, all of which are initially zero. The total number of type i items is s_i, while the number of type i red items is e_i. For $1 \leq i \leq n$, the invariant $e_i = \lfloor \alpha^i s_i \rfloor$ is maintained.

$\beta_i = \lfloor 1/t^i \rfloor$ is the number of type i items which fit in a bin. Blue items of type i are placed β_i in a bin, as in HARMONIC.

Δ is the amount of space left when a type 2 item is placed in a bin. If possible, we would like to use this space to pack red items.

Since items of types 1, 2 and 3 may not fit in a bin with a type 2 item, we require that $\alpha^1 = \alpha^2 = \alpha^3 = 0$. Items of type $i \geq 4$ are guaranteed to fit in a bin with a type 2 item. Define $\gamma_i = \lfloor \Delta / t^i \rfloor$ for $i \geq 4$. This is the number of red items of type i that the algorithm places together in a bin with a type 2 item.

We explain the method by which red items are packed with type 2 items. When a bin is opened, it is assigned to a *group*. The bin groups are named:

$$\begin{aligned} &1, 3, 4, \ldots, n+1; \\ &(2, ?); \\ &(?, i), \qquad \text{for } \alpha^i \neq 0,\ 1 \leq i \leq n; \\ &(2, i), \qquad \text{for } \alpha^i \neq 0,\ 1 \leq i \leq n. \end{aligned}$$

We call these groups *monochromatic*, *indeterminate blue*, *indeterminate red* and *bichromatic*, respectively.

The monochromatic group i contains bins which hold only blue items of type i. There is one open bin in each of these groups; this bin has fewer than β_i items. The closed bins all contain β_i items.

The bichromatic group $(2, j)$ contains bins which contain a blue item of type 2 along with red items of type j. A closed bin in this group contains one type 2 item and γ_j type j items. There is at most one open bin.

The indeterminate blue group $(2, ?)$ contains bins which hold only a blue item of type 2. These bins are all open.

```
Initialize $e_i \leftarrow 0$ and $s_i \leftarrow 0$ for $1 \le i \le n$.
For each piece $p$:
    $i \leftarrow \tau(p)$.
    If $i = n + 1$ place $p$ using NEXT FIT.
    Else:
        $s_i \leftarrow s_i + 1$.
        If $e_i < \lfloor \alpha^i s_i \rfloor$:
            $e_i \leftarrow e_i + 1$.
            Color $p$ red.
            If there is an open bin in group $(2, i)$ or $(?, i)$ with fewer than $\gamma_i$ type $i$ items, then
            place $p$ in this bin.
            Else if there is some bin in group $(2, ?)$ then place $p$ in it and change the group of
            this bin to $(2, i)$.
            Otherwise, open a new group $(?, i)$ bin and place $p$ in it.
        Else:
            Color $p$ blue.
            If $i = 2$:
                If there is some bin in group $(?, j)$ then place $p$ in it and change the group of
                this bin to $(2, j)$.
                Otherwise, open a new group $(2, ?)$ bin and place $p$ there.
            Else:
                If there is an open bin in group $i$ with fewer than $\beta_i$ items, then place $p$ in this
                bin.
                If not, open a new group $i$ bin and place $p$ there.
```

Figure 1: The IMPROVED HARMONIC Algorithm.

The indeterminate red group $(?, j)$ contains bins which hold only red items of type j. Again, these bins are all open, but only one has fewer than γ_j items.

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 blue items together whenever possible; when this is not possible we place them in indeterminate bins in hope that they can later be so combined. A formal description of IMPROVED HARMONIC is displayed in Figure 1.

An IMPROVED HARMONIC algorithm can be analyzed using the method of weighting systems developed in [30]. The full generality of weighting systems is not required here, so we adopt a slightly different notation than that used in [30], and restrict ourselves to a subclass of weighting systems. However, the definitions here are consistent with those in [30].

A weighting system for a IMPROVED HARMONIC 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. Specifically, we have

$$W_{\mathcal{A}}(x) \quad = \quad \begin{cases} 1 & \text{if } x \in I_2, \\ \dfrac{1-\alpha^i}{\beta_i} & \text{if } x \in I_i \text{ with } i \in \{1,3,\dots,n\}, \\ \dfrac{x}{1-\epsilon} & \text{if } x \in I_{n+1}. \end{cases}$$

$$V_{\mathcal{A}}(x) \quad = \quad \begin{cases} 0 & \text{if } x \in I_2, \\ \dfrac{1-\alpha^i}{\beta_i} + \dfrac{\alpha^i}{\gamma_i} & \text{if } x \in I_i \text{ with } i \in \{1,3,\dots,n\}, \\ \dfrac{x}{1-\epsilon} & \text{if } x \in I_{n+1}. \end{cases}$$

Given these definitions, the following lemma follows directly from Lemma 4 of [30]:

Lemma 3.1 *If $\mathcal{A}$ is an* Improved Harmonic *algorithm then for all σ,*

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

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

Let f be some function $f : (0,1] \mapsto \mathbb{R}^+$. Define $\mathcal{P}(f)$ to be the mathematical program: Maximize

$$\sum_{x\in X}^{n} f(x)$$

subject to $\sum_{x\in X} x \le 1$, over all finite sets of real numbers X. It is further shown in [30] that the performance ratio of $\mathcal{A}$ is upper bounded by the maximum of the values of $\mathcal{P}(W_{\mathcal{A}})$ and $\mathcal{P}(V_{\mathcal{A}})$. Intuitively, given a weighting function f, $\mathcal{P}(f)$ upper bounds the amount of weight that can be packed in a single bin.

Example 1 Harmonic (H) is the Improved Harmonic algorithm with $\alpha^i = 0$ for $1 \le i \le n$ and Δ any value in $[\frac{1}{2}, \frac{1}{3}]$. (The choice of Δ is irrelevant, since there are no red items). In this case, we have $W_{\text{H}}(x) \ge V_{\text{H}}(x)$ for all $x \in [0,1]$. The results of [24] imply that the value of $\mathcal{P}_{\text{H}}$ is

$$\Pi_n = \sum_{j=1}^{i} \frac{1}{\pi_j - 1} + \frac{n-1}{(\pi_{i+1}-1)(n-2)}, \tag{3.1}$$

where i is the integer satisfying $\pi_i < n \le \pi_{i+1}$. Note that this is consistent with our earlier definition of Π_∞ since $\Pi_\infty = \lim_{n\to\infty} \Pi_n$.

Example 2 Modified Harmonic (MH) is defined by $n = 39$, $\Delta = 265/684$ and

$$\begin{aligned} \alpha^1 &= \alpha^2 = \alpha^3 = 0; \\ \alpha^4 &= \tfrac{1}{9}; \\ \alpha^5 &= \tfrac{1}{12}; \\ \alpha^6 &= \alpha^7 = 0; \\ \alpha^i &= \frac{39-i}{37(i-1)}, \qquad \text{for } 8 \le i \le 38; \\ \alpha^{39} &= 0. \end{aligned}$$

The results of [28] imply that the value of $\mathcal{P}_{\mathrm{MH}}$ is $\frac{538}{333} < 1.61562$.

4. The $\mathcal{A} \times \mathcal{B}$ Algorithm

Let $\mathcal{A}$ and $\mathcal{B}$ be instances of Improved Harmonic. We construct an algorithm $\mathcal{A} \times \mathcal{B}$ for the two-dimensional box packing problem from $\mathcal{A}$ and $\mathcal{B}$.

$\mathcal{A} \times \mathcal{B}$ operates as follows: We run $n_{\mathcal{B}}+1$ independent copies of $\mathcal{A}$, which we denote $\mathcal{A}_1, \dots, \mathcal{A}_{n_{\mathcal{B}}+1}$. Small items are given to Geometric Next Fit which also uses $\mathcal{A}$ as its sub-routine algorithm. A large item p is given to algorithm $\mathcal{A}_{\tau_{\mathcal{B}}(w(p))}$. I.e. items are processed according to the type of their width. Algorithm $\mathcal{A}_i$ packs the items it receives in slices of height 1 and width $t^i_{\mathcal{B}}$. When $\mathcal{A}$ is used to pack an item into a slice, it considers only the item's height. I.e. it considers the size of the item to be $h(p)$. The size of a slice is defined to be its width. When a new slice is allocated by some $\mathcal{A}_i$, we allocate it from a bin using algorithm $\mathcal{B}$. As far as $\mathcal{B}$ is concerned, the slices are items, and $\mathcal{B}$ knows nothing of the internals of a slice.

We shall require that $\mathcal{A}$ is Harmonic, also denoted by H, while $\mathcal{B}$ is some Improved Harmonic algorithm. It may be possible to get improved performance without these restrictions, but the analysis would seem to be significantly more complicated. We shall also assume that $n_{\mathcal{A}} = n_{\mathcal{B}}$, and henceforth we refer to this common value as n.

We are ready to start analyzing $\mathrm{H} \times \mathcal{B}$. During course of processing, $\mathrm{H}_1, \dots, \mathrm{H}_{n+1}$ make allocation requests for slices to $\mathcal{B}$. Define ς to be the sequence of slices received by $\mathcal{B}$. Define ϱ_i to be the sub-sequence of σ of items p with $\tau_{\mathcal{B}}(w(p)) = i$. From the definition of $\mathrm{H} \times \mathcal{B}$ we have

$$\mathrm{cost}_{\mathrm{H}\times\mathcal{B}}(\sigma) = \mathrm{cost}_{\mathcal{B}}(\varsigma) + \mathrm{cost}_{\mathrm{GNF}}(\varrho_{n+1})$$

We first consider the cost incurred by $\mathcal{B}$: If $s \in \varsigma$ is a slice, we use $w(s)$ to denote its width (size). Define ς_i to be the sub-sequence of ς of slices s with $\tau_{\mathcal{B}}(w(s)) = i$. To start, we have

$$\begin{aligned}
\sum_{s\in\varsigma} W_{\mathcal{B}}(w(s)) &= \sum_{i=1}^{n} \sum_{s\in\varsigma_i} W_{\mathcal{B}}(w(s)) \\
&= \sum_{i=1}^{n} W_{\mathcal{B}}(t^i_{\mathcal{B}})|\varsigma_i| \\
&= \sum_{i=1}^{n} W_{\mathcal{B}}(t^i_{\mathcal{B}})\mathrm{cost}_{\mathrm{H}_i}(\varrho_i) \\
&\le \sum_{i=1}^{n} W_{\mathcal{B}}(t^i_{\mathcal{B}}) \sum_{p\in\varrho_i} W_{\mathrm{H}}(h(p)) + O(1) \\
&= \sum_{i=1}^{n} \sum_{p\in\varrho_i} W_{\mathrm{H}}(h(p)) W_{\mathcal{B}}(t^i_{\mathcal{B}}) + O(1) \\
&= \sum_{i=1}^{n} \sum_{p\in\varrho_i} W_{\mathrm{H}}(h(p)) W_{\mathcal{B}}(w(p)) + O(1).
\end{aligned}$$

Using analogous reasoning, we have $\sum_{s\in\varsigma} V_{\mathcal{B}}(w(s)) \le \sum_{i=1}^{n} \sum_{p\in\varrho_i} W_{\mathrm{H}}(h(p)) V_{\mathcal{B}}(w(p)) + O(1)$. We define $W_{\mathrm{H}\times\mathcal{B}}(x,y) = W_{\mathrm{H}}(x)W_{\mathcal{B}}(y)$ and $V_{\mathrm{H}\times\mathcal{B}}(x,y) = W_{\mathrm{H}}(x)V_{\mathcal{B}}(y)$. In an abuse of notation, we use $W_{\mathrm{H}\times\mathcal{B}}(p)$ to mean $W_{\mathrm{H}\times\mathcal{B}}(h(p), w(p))$ and $V_{\mathrm{H}\times\mathcal{B}}(p)$ to mean $V_{\mathrm{H}\times\mathcal{B}}(h(p), w(p))$. We therefore have

$$\mathrm{cost}_{\mathcal{B}}(\varsigma) \le \max\left\{ \sum_{i=1}^{n} \sum_{p\in\varrho_i} W_{\mathrm{H}\times\mathcal{B}}(p), \sum_{i=1}^{n} \sum_{p\in\varrho_i} V_{\mathrm{H}\times\mathcal{B}}(p) \right\} + O(1).$$

We now consider the number of bins used by Geometric Next Fit. Define $\varrho_{n+1,j}$ to be the sub-sequence of ϱ_{n+1} of items with class j. Only a finite number of these sub-sequences are non-empty.

Let z be the sum of the sizes of the slices allocated by GEOMETRIC NEXT FIT. By definition of the algorithm, each bin other than the one fully open bins contains slices with total size at least $1-\epsilon$. Therefore, we have $\mathrm{cost}_{\mathrm{GNF}}(\varrho_{n+1}) \le z/(1-\epsilon)+1$. We upper bound z as follows:

$$\begin{aligned}
z &= \sum_{j=1}^{\infty} \epsilon(1-\delta)^j \mathrm{cost}_{\mathrm{H}_{n+1,j}}(\varrho_{n+1,j}) \\
&\le \sum_{j=1}^{\infty} \epsilon(1-\delta)^j \left(\sum_{p\in\varrho_{n+1,j}} W_{\mathrm{H}}(h(p)) + O(1) \right) \\
&= \sum_{j=1}^{\infty} \epsilon(1-\delta)^j \sum_{p\in\varrho_{n+1,j}} W_{\mathrm{H}}(h(p)) + O(1)\sum_{j=1}^{\infty} \epsilon(1-\delta)^j \\
&= \sum_{j=1}^{\infty} \sum_{p\in\varrho_{n+1,j}} \epsilon(1-\delta)^j W_{\mathrm{H}}(h(p)) + O(1) \\
&< \frac{1}{1-\delta}\sum_{j=1}^{\infty} \sum_{p\in\varrho_{n+1,j}} w(p) W_{\mathrm{H}}(h(p)) + O(1) \\
&= \frac{1}{1-\delta}\sum_{p\in\varrho_{n+1}} w(p) W_{\mathrm{H}}(h(p)) + O(1).
\end{aligned}$$

Therefore, the number of bins used by GEOMETRIC NEXT FIT is

$$\begin{aligned}
\mathrm{cost}_{\mathrm{GNF}}(\varrho_{n+1}) &\le \frac{1}{(1-\delta)(1-\epsilon)} \sum_{p\in\varrho_{n+1}} w(p) W_{\mathrm{H}}(h(p)) + O(1) \\
&= \frac{1}{1-\delta} \sum_{p\in\varrho_{n+1}} W_{\mathrm{H}\times\mathcal{B}}(p) + O(1).
\end{aligned}$$

Putting these two results together, we get

$$\begin{aligned}
\mathrm{cost}_{\mathrm{H}\times\mathcal{B}}(\sigma) &\le \max\left\{ \sum_{i=1}^{n}\sum_{p\in\varrho_i} W_{\mathrm{H}\times\mathcal{B}}(p), \sum_{i=1}^{n}\sum_{p\in\varrho_i} V_{\mathrm{H}\times\mathcal{B}}(p) \right\} + \frac{1}{1-\delta}\sum_{p\in\varrho_{n+1}} W_{\mathrm{H}\times\mathcal{B}}(p) + O(1) \\
&\le \frac{1}{1-\delta} \max\left\{ \sum_{i=1}^{n+1}\sum_{p\in\varrho_i} W_{\mathrm{H}\times\mathcal{B}}(p), \sum_{i=1}^{n+1}\sum_{p\in\varrho_i} V_{\mathrm{H}\times\mathcal{B}}(p) \right\} + O(1) \\
&\le \frac{1}{1-\delta} \max\left\{ \sum_{p\in\sigma} W_{\mathrm{H}\times\mathcal{B}}(p), \sum_{p\in\sigma} V_{\mathrm{H}\times\mathcal{B}}(p) \right\} + O(1). \qquad (4.1)
\end{aligned}$$

We are using the fact that $W_{\mathrm{H}\times\mathcal{B}}(p) = V_{\mathrm{H}\times\mathcal{B}}(p)$ for all $p \in \varrho_{n+1}$. As is the case with a weighting system for the one dimensional problem, we can conclude that the cost of $\mathrm{H}\times\mathcal{B}$ is upper bounded by the sum of the weights of items in σ.

We now relate the algorithm's cost to that of the optimal offline algorithm. To facilitate this, we need the following definitions: Let $m = \mathrm{cost}(\sigma)$ be the number of bins in the optimal offline solution. For $1 \le i \le m$, let X_i be the multi-set of items contained in the ith bin of the optimal offline solution.

We now rewrite (4.1) as

$$\begin{aligned}
\mathrm{cost}_{\mathrm{H}\times\mathcal{B}}(\sigma) &\leq \frac{1}{1-\delta}\max\left\{\sum_{i=1}^{m}\sum_{p\in X_i} W_{\mathrm{H}\times\mathcal{B}}(p),\ \sum_{i=1}^{m}\sum_{p\in X_i} V_{\mathrm{H}\times\mathcal{B}}(p)\right\} + O(1) \\
&\leq \frac{1}{1-\delta}\max\left\{m\max_{1\leq i\leq m}\sum_{p\in X_i} W_{\mathrm{H}\times\mathcal{B}}(p),\ m\max_{1\leq i\leq m}\sum_{p\in X_i} V_{\mathrm{H}\times\mathcal{B}}(p)\right\} + O(1) \\
&= \frac{1}{1-\delta}\max_{1\leq i\leq m}\left\{\sum_{p\in X_i} W_{\mathrm{H}\times\mathcal{B}}(p),\ \sum_{p\in X_i} V_{\mathrm{H}\times\mathcal{B}}(p)\right\}\mathrm{cost}(\sigma) + O(1).
\end{aligned}$$

We conclude that the performance ratio of $\mathrm{H}\times\mathcal{B}$ is upper bounded by the value of the mathematical program: Maximize

$$\frac{1}{1-\delta}\max\left\{\sum_{p\in X} W_{\mathrm{H}\times\mathcal{B}}(p),\ \sum_{p\in X} V_{\mathrm{H}\times\mathcal{B}}(p)\right\}$$

over all finite multisets of items X which fit in a single bin. We call this optimization problem $\mathcal{Q}$. The reader should note the similarity to the program $\mathcal{P}$ defined earlier.

We show

Lemma 4.1 *Let f and g be functions mapping from $(0,1]$ to $\mathbb{R}^+$. Let F and G be the values of the mathematical programs $\mathcal{P}(f)$ and $\mathcal{P}(g)$, respectively. Then the maximum of*

$$\sum_{p\in X} f(h(p))\,g(w(p))$$

over all finite multisets of items X which fit in a single bin is at most FG.

Proof The proof is similar to the proof of Lemma 2 in [9]. We consider modified multisets of items X' and X''. The items in each set are the same, however, the widths and heights of items are modified as follows: For each item $p\in X$ there is an item $p'\in X'$ with $w(p')=w(p)$ and $h(p')=f(h(p))$. For each item $p\in X$ there is an item $p''\in X''$ with $w(p'')=g(w(p))$ and $h(p'')=f(h(p))$. Note that the area of item p'' is $f(h(p))g(w(p))$. We show that the items in X'' fit in box of height F and width G, which proves the desired result.

Towards this goal, we first show that the items in X' fit in a box of height F and width 1. Since the items in X fit in a bin, there is some pair of functions $\phi : X \mapsto [0,1]$ and $\varphi : X \mapsto [0,1]$ such that $(\phi(p),\varphi(p))$ is the position of p. The positions must obey $\phi(p)+w(p)\leq 1$ and $\varphi(p)+h(p)\leq 1$ for all $p\in X$. Furthermore, the coordinates must be such that the area of the region described by

$$\left\{(x,y)\,\middle|\, \begin{array}{l} x\in[\phi(p),\phi(p)+w(p)]\cap[\phi(q),\phi(q)+w(q)], \\ y\in[\varphi(p),\varphi(p)+h(p)]\cap[\varphi(q),\varphi(q)+h(q)], \end{array}\right\}$$

is zero for all $p,q\in X$. Intuitively, we describe a bin using a coordinate system with origin (0,0) in the bin's lower left corner. For each item p, $\phi(p)$ is the horizontal coordinate of the lower left corner of p, whereas $\varphi(p)$ is the vertical coordinate. The first restriction simply says that each item must be wholly contained in the bin. The second condition prevents two items from overlapping.

We may assume without loss of generality that for all items $p\in X$, either $\phi(p)=0$ or there exists an item q such that $\phi(p)=\phi(q)+w(q)$. I.e. each item is as far to the left as possible. If this were not the case, we can simply move p left until it is true, without effecting the validity of the packing. Similarly, we also assume without loss of generality for all items $p\in X$, either $\varphi(p)=0$ or there exists an item q such that $\varphi(p) = \varphi(q) + h(q)$. We say that $q \leftarrow p$ if and only if $\phi(p) = \phi(q) + w(q)$. We say that p is *leftmost* if $\varphi(p) = 0$ and *rightmost* if there is no q such that $p \leftarrow q$. A *horizontal chain* is a sequence of items $p_1 \leftarrow p_2 \leftarrow \cdots p_\ell$ where p_1 is leftmost and p_ℓ is rightmost. Analogously, we say that $q \downarrow p$ if and only if $\varphi(p) = \varphi(q) + h(q)$ and define *bottommost*, *topmost* and *vertical chain*.

Given ϕ and φ, we construct ϕ' and φ', which pack the items in X' into a bin of height F and width 1. This is done as follows: $\phi'(p')$ is assigned $\phi(p)$ for all $p' \in X'$. For all bottommost items q' we set $\varphi'(q') = 0$. Once we have computed $\varphi'(q')$ for all q' such that $q \downarrow p$ we assign $\varphi'(p')$ the value

$$\max_{q',\, q \downarrow p} \varphi'(q') + h(q').$$

Since the relation $\downarrow$ induces a directed acyclic graph on X, this process terminates. Further, it is easily seen that no items overlap in the resulting packing. Suppose that the packing does not fit in the required box. Only the vertical coordinates of items change, so this means there is some item r' with $\varphi'(r') + h(r') > F$. With loss of generality, r' is topmost. Then by the construction of φ', there is a vertical chain $p'_1 \downarrow \cdots \downarrow p'_\ell = r$ such that

$$\sum_{i=1}^{\ell} h(p'_i) = \sum_{i=1}^{\ell} f(h(p_i)) > F.$$

But since $\sum_{i=1}^{\ell} h(p_i) \leq 1$, this contradicts the fact that $\mathcal{P}(f)$ has value F.

Now, given ϕ' and φ', we construct ϕ'' and φ'', which pack the items in X'' into a bin of height F and width G. The construction is analogous to the one just given: $\varphi''(p'')$ is assigned $\varphi'(p')$ for all $p'' \in X''$. For all leftmost items q'' we set $\phi''(q'') = 0$. Once we have computed $\phi''(q'')$ for all q'' such that $q \leftarrow p$ we assign $\phi''(p'')$ the value

$$\max_{q'',\, q \leftarrow p} \phi''(q'') + w(q'').$$

Once the construction is complete, if there is some topmost item r'' with $\phi''(r'') + w(r'') > G$ then there is a horizontal chain $p''_1 \leftarrow \cdots \leftarrow p''_\ell = r''$ such that

$$\sum_{i=1}^{\ell} w(p''_i) = \sum_{i=1}^{\ell} g(w(p_i)) > G.$$

Since $\sum_{i=1}^{\ell} w(p_i) \leq 1$, this contradicts the fact that $\mathcal{P}(g)$ has value G. ∎

This implies the main theorem of this section:

Theorem 4.1 *For all $\delta > 0$, the asymptotic performance ratio of* H × MH *is at most*

$$\frac{78548}{28749(1-\delta)} < \frac{2.73220}{1-\delta}$$

and at least $137243/50274 > 2.72990$.

Proof First consider the upper bound. We have $n = 39$ and therefore $\Pi_n = 438/259$ by (3.1). By Example 2, $\max\{\mathcal{P}(W_{\mathrm{MH}}), \mathcal{P}(V_{\mathrm{MH}})\} \leq 538/333$. Note that $\mathcal{Q}$ can be decomposed into two mathematical programs which have the form required by Lemma 4.1. In the first we have $f = W_{\mathrm{H}}$ and $g = W_{\mathrm{MH}}$. Hence $\max_{p \in X} W_{\mathrm{H}\times\mathrm{MH}}(p) \leq \mathcal{P}(W_{\mathrm{H}})\mathcal{P}(W_{\mathrm{MH}}) \leq \frac{438}{259} \cdot \frac{538}{333}$. In the second we have $f = W_{\mathrm{H}}$ and $g = V_{\mathrm{MH}}$. Hence $\max_{p \in X} V_{\mathrm{H}\times\mathrm{MH}}(p) \leq \mathcal{P}(W_{\mathrm{H}})\mathcal{P}(V_{\mathrm{MH}}) \leq \frac{438}{259} \cdot \frac{538}{333}$. This gives the bound.

Now consider the lower bound. Let $0 < \varepsilon < 1/5418$ be a real number. Define $s_1 = \frac{1}{2} + \varepsilon$, $s_2 = \frac{1}{3} + \varepsilon$, $s_3 = \frac{1}{7} + \varepsilon$ and $s_4 = \frac{1}{42} - 3\varepsilon$. For $1 \leq i \leq 4$ and $1 \leq j \leq 4$, define an (i,j) item to be one with height s_i and width s_j. Consider an input consisting of N items of each type (i,j), $1 \leq i \leq 4$ and

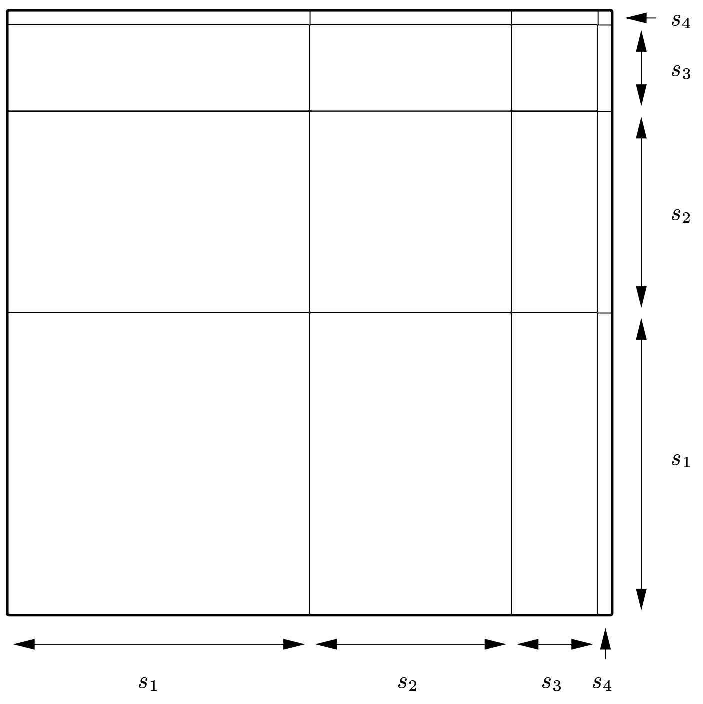

Figure 2: A bin in the optimal offline solution of Theorem 4.1.

$1 \le j \le 4$. Note that the optimal offline cost is N; we can pack each bin so that it contains one item of each type. The packing is illustrated in Figure 2. Consider now the cost to H $\times$ MH. We start by considering the number of slices used by pieces of width s_1. Each of these slices has width $t^2_{\mathrm{MH}} = 419/684$. Items of width s_2, s_3 go into slices of widths 265/684, 1/6 respectively. Let k be the integer such that $\epsilon(1-\delta)^{k-1} < s_4 \le \epsilon(1-\delta)^k$. Items of width s_4 are packed by Geometric Next Fit into slices of width $\epsilon(1-\delta)^k$. For each slice width, the number of slices used is at least $\ell + \ell/2 + \ell/6 + \ell/42 = 71\ell/42$. Now consider the way that Modified Harmonic packs slices into bins. There are at least $71\ell/42$ bins containing slices of width 419/684. Some of these bins also contain slices of width 265/684. There are at least $4/9 \cdot 71\ell/42 = 142\ell/189$ bins containing only slices of width 265/684. There are at least $39/266 \cdot 71\ell/42 = 923\ell/3724$ bins containing only slices of width 1/6. There are at least $71 s_4 \ell/42 = 71\ell/1764 - 71\varepsilon\ell/14$ bins containing only slices of width $\epsilon(1-\delta)^k$. Therefore, for all $\varepsilon > 0$, the performance ratio of H$\otimes$MH is at least $137243/50274 - 71\varepsilon/14$. ■

5. Improving on $\mathcal{A} \times \mathcal{B}$

In the algorithm $\mathcal{A} \times \mathcal{B}$ described in the previous section, the roles played by height and width can be interchanged. In fact, our choice of these roles is completely arbitrary. This insight leads us to consider the following randomized algorithm: Define $-\mathcal{A} \times \mathcal{B}$ to be $\mathcal{A} \times \mathcal{B}$ with the roles of height and width interchanged. Before processing begins, we flip a fair coin. If the result is heads, we run $\mathcal{A} \times \mathcal{B}$. If the result is tails, we run $-\mathcal{A} \times \mathcal{B}$. Call this algorithm $\mathcal{A} \otimes \mathcal{B}$.

As with $\mathcal{A} \times \mathcal{B}$, we require that $\mathcal{A}$ is Harmonic and $\mathcal{B}$ is some Improved Harmonic algorithm. To facilitate the analysis, we will use the following definitions:

$$W_{-\mathrm{H}\times\mathcal{B}}(x,y) = W_{\mathrm{H}}(y)\, W_{\mathcal{B}}(x), \qquad V_{-\mathrm{H}\times\mathcal{B}}(x,y) = W_{\mathrm{H}}(y)\, V_{\mathcal{B}}(x),$$

and

$$\begin{aligned}
WW_{\mathrm{H}\otimes\mathcal{B}}(x,y) &= \frac{W_{\mathrm{H}}(x)\,W_{\mathcal{B}}(y)+W_{\mathrm{H}}(y)\,W_{\mathcal{B}}(x)}{2},\\
WV_{\mathrm{H}\otimes\mathcal{B}}(x,y) &= \frac{W_{\mathrm{H}}(x)\,W_{\mathcal{B}}(y)+W_{\mathrm{H}}(y)\,V_{\mathcal{B}}(x)}{2},\\
VW_{\mathrm{H}\otimes\mathcal{B}}(x,y) &= \frac{W_{\mathrm{H}}(x)\,V_{\mathcal{B}}(y)+W_{\mathrm{H}}(y)\,W_{\mathcal{B}}(x)}{2},\\
VV_{\mathrm{H}\otimes\mathcal{B}}(x,y) &= \frac{W_{\mathrm{H}}(x)\,V_{\mathcal{B}}(y)+W_{\mathrm{H}}(y)\,V_{\mathcal{B}}(x)}{2}.
\end{aligned}$$

We shall abuse notation in a manner analogous to the previous section. Using the results of the previous sections, the performance ratio of the algorithm is the maximum of

$$\begin{aligned}
&\frac{1}{2(1-\delta)}\left(\max\left\{\sum_{p\in X} W_{\mathrm{H}\times\mathcal{B}}(p),\ \sum_{p\in X} V_{\mathrm{H}\times\mathcal{B}}(p)\right\}+\max\left\{\sum_{p\in X} W_{-\mathrm{H}\times\mathcal{B}}(p),\ \sum_{p\in X} V_{-\mathrm{H}\times\mathcal{B}}(p)\right\}\right)\\
&= \frac{1}{1-\delta}\max\left\{\sum_{p\in X}\frac{W_{\mathrm{H}\times\mathcal{B}}(p)+W_{-\mathrm{H}\times\mathcal{B}}(p)}{2},\ \sum_{p\in X}\frac{W_{\mathrm{H}\times\mathcal{B}}(p)+V_{-\mathrm{H}\times\mathcal{B}}(p)}{2},\right.\\
&\qquad\qquad \left.\sum_{p\in X}\frac{V_{\mathrm{H}\times\mathcal{B}}(p)+W_{-\mathrm{H}\times\mathcal{B}}(p)}{2},\ \sum_{p\in X}\frac{V_{\mathrm{H}\times\mathcal{B}}(p)+V_{-\mathrm{H}\times\mathcal{B}}(p)}{2}\right\}\\
&= \frac{1}{1-\delta}\max\left\{\sum_{p\in X} WW_{\mathrm{H}\otimes\mathcal{B}}(p),\ \sum_{p\in X} WV_{\mathrm{H}\otimes\mathcal{B}}(p),\ \sum_{p\in X} VW_{\mathrm{H}\otimes\mathcal{B}}(p),\ \sum_{p\in X} VV_{\mathrm{H}\otimes\mathcal{B}}(p)\right\} \qquad (5.1)
\end{aligned}$$

over all finite multisets of items X which fit in a single bin.

We would like to upper bound the weight in a single bin using Lemma 4.1. However, it is unclear how to 'factor' the weight of an item. We clarify what is meant by this. Define $\Lambda = \{WW, WV, VW, WW\}$. A pseudo-weighting system for $\mathrm{H}\otimes\mathcal{B}$ is a set of pairs of functions $\{(f_\lambda, g_\lambda) \mid \lambda\in\Lambda\}$ such that

$$\lambda_{\mathrm{H}\otimes\mathcal{B}}(x,y) \le f_\lambda(y)\,g_\lambda(x),$$

for all $0\le x\le 1$ and $0\le y\le 1$. In the case of $\mathrm{H}\otimes\mathcal{B}$, it is not immediately clear how the pseudo-weighting functions should be defined. However, we can somewhat simplify matters by choosing

$$g_\lambda(x) = \sup_{0<y\le 1}\frac{\lambda_{\mathrm{H}\otimes\mathcal{B}}(x,y)}{f_\lambda(y)}.$$

The following choices of f yield good results:

$$\begin{aligned}
f_{WW}(x) &= f_{WV}(x) = \frac{W_{\mathrm{H}}(x)+W_{\mathcal{B}}(x)}{2},\\
f_{VW}(x) &= f_{VV}(x) = \frac{W_{\mathrm{H}}(x)+V_{\mathcal{B}}(x)}{2}.
\end{aligned}$$

We now turn our attention to $\mathcal{B}$. The following lemma demonstrates that using Modified Harmonic for $\mathcal{B}$ yields no significant improvement:

Lemma 5.1 *The asymptotic performance ratio of* $\mathrm{H}\otimes\mathrm{MH}$ *is at least* $137243/50274 > 2.72990$.

Proof Consider the lower bound sequence given in the proof of Theorem 4.1. Note that it is invariant under 90 degree rotation. So the cost to $-\mathcal{A}\times\mathcal{B}$ is the same as that to $\mathcal{A}\times\mathcal{B}$. ■

The preceding lemma shows us that MODIFIED HARMONIC is not a good choice because it has the same worst case as HARMONIC. We are therefore led to consider other algorithms, ones which do well when HARMONIC does badly. Through experimentation, we have found that the following instance of IMPROVED HARMONIC does much better than MODIFIED HARMONIC: $n = 39$, $\Delta = 2825022678/7478572741$ and

$$\begin{aligned}
\alpha^1 &= \alpha^2 = \alpha^3 = 0; \\
\alpha^4 &= \frac{37958247020777}{189267718929228}; \\
\alpha^5 &= \frac{1}{25}; \\
\alpha^6 &= \alpha^7 = 0; \\
\alpha^i &= \frac{39-i}{37(i-1)}, \qquad \text{for } 8 \le i \le 38; \\
\alpha^{39} &= 0.
\end{aligned}$$

Call this algorithm STRANGE HARMONIC (SH).

We are now prepared to state the main result of this section:

Theorem 5.1 *For all $\delta > 0$, the asymptotic performance ratio of* H $\otimes$ SH *is at most* $2.66013/(1-\delta)$.

Proof We apply Lemma 4.1 four times to upper the value of (5.1). Using the branch and bound method described in [30], we calculate the following values of $\mathcal{P}$: The values of $\mathcal{P}(f_{WW})$ and $\mathcal{P}(g_{WW})$ are both at most 1.630990197. The square of this is at most 2.66013. The values $\mathcal{P}(f_{WV})$ and $\mathcal{P}(g_{WV})$ are at most 1.63100 and 1.60293, respectively. The product of these values is at most 2.61439. The values $\mathcal{P}(f_{VW})$ and $\mathcal{P}(g_{VW})$ are at most 1.59726 and 1.63775, respectively. The product of these values is at most 2.61592. The values $\mathcal{P}(f_{VV})$ and $\mathcal{P}(g_{VV})$ are at most 1.59726 and 1.66540, respectively. The product of these values is at most 2.66009. ■

It is possible to de-randomize $\mathcal{A} \otimes \mathcal{B}$ without increasing its performance ratio. We briefly sketch how this works: We run both $\mathcal{A} \times \mathcal{B}$ and $-\mathcal{A} \times \mathcal{B}$ independently and concurrently. For each item, we decide whether to give it to $\mathcal{A} \times \mathcal{B}$ or $-\mathcal{A} \times \mathcal{B}$. To make this decision for a piece p we consider $i = \tau_{\mathcal{A}}(h(p))$ and $j = \tau_{\mathcal{B}}(w(p))$. For items with $i \le n$ and $j \le n$, we simply balance the number of items going to each algorithm, for each (i,j). I. e. for each (i,j) we give a new item to the algorithm that has received the least number of items of these types so far. For items with $i = n+1$ or $j = n+1$, we have to be a bit more careful. We instead balance the total size of items going to each algorithm. Hence for each (i,j) with $i = n+1$ or $j = n+1$, we keep track of the total size of items that has been given to the algorithms, and give a new item to the algorithm that has received items of the least total size so far.

6. AN OFFLINE APPROXIMATION ALGORITHM FOR SQUARE PACKING

In the remainder of the paper, we will discuss square packing. From now on, the *size* of an item is defined as the length of its edges.

We begin by studying the off-line problem. We present a $14/9 + \varepsilon$ approximation algorithm called SQUARE SCHEME for the two-dimensional square packing problem for any $\varepsilon > 0$. The algorithm uses several ideas used in the design of approximation algorithms for one-dimensional bin packing [13].

The basic idea is as follows: we classify items as either small or large. Large items are further classified according to their size. For the large items, we create an optimal solution using packing patterns derived from an integer program. Small items are then packed first into bins with certain large items, and then, if necessary, into bins by themselves.

The key is to determine which bins can hold small items. We pack small items only into bins that contain a single large item of size at least $1/2$. The key is to show that the remaining bins cannot hold too many small items in the optimal offline solution.

To begin, we consider the simpler problem of two-dimensional square packing when there are a fixed number m of item sizes. Let the sizes be $A = \{a_1 > a_2 > \cdots > a_m\}$. Call this problem A-restricted square packing.

A *pattern* over A is finite multiset P of squares with sizes in A that fit (in some way) into a unit square. For a pattern P, for $1 \leq j \leq m$, define P_j to be the number of items of size a_j in P. Define the *order* of pattern P to be $\max\{j | P_j > 0\}$. A pattern of order j is *dominant* if when we increase the number of items of size a_j, the resulting multi-set of items no longer fits in a unit square. The *waste* of pattern P is defined to be $1 - \sum_i P_i a_i^2$. Define A^* to be the set of all dominant patterns over A. Note that $|A^*| \leq m^{\lfloor 1/a_m^2 \rfloor}$.

Consider an input which contains b_i items of size a_i for $1 \leq i \leq m$. We can obtain an optimal packing by computing the value of the integer program: Minimize $\sum_{P \in A^*} \phi_P$, subject to

$$\begin{array}{rcll} \phi_P & \geq & 0, & \text{for all } P \in A^*; \\ \sum_{P \in A^*} \phi_P P_j & \geq & b_j, & \text{for } 1 \leq j \leq m. \end{array}$$

over integer variables $\phi_P, P \in A^*$. First we note that by considering only dominant patterns we lose no generality, since the constraints in the linear program simply require that we have enough of each item. Second, note that we can compute the value of this integer program in time $O(m^{|A^*|} m)$ by enumerating all $m^{|A^*|}$ solution vectors ϕ, checking that each is feasible and keeping track of the minimum objective value among the feasible vectors. Call this algorithm for solving A-restricted square packing RESTRICTED BIG SQUARES (RBS).

We now consider the construction of a solution for the unrestricted square packing problem, called *SquareScheme*. We use a parameter Δ, the exact value of which shall be determined later in our exposition. Let σ be the input. From σ, we create two new lists of squares. The first, $\sigma_>$ contains all squares of size greater than Δ. The second, $\sigma_\leq$ contains the remaining squares.

We sort the squares in $\sigma_>$, and then divide them into m groups of approximately the same size. More precisely, we sort $\sigma_>$ to get squares $x_1 \geq x_2 \geq \cdots x_{|\sigma_>|}$. We also use x_i to denote the size of the ith square in this sorted list. Define $k = \lceil |\sigma_>|/m \rceil$, X_i to be the sub-sequence of squares $x_{k(i-1)+1}, \ldots, x_{ki}$ for $1 \leq i \leq m-1$, and X_m to be the sub-sequence of squares $x_{m(i-1)+1}, \ldots, x_{|\sigma_>|}$.

We get an instance of A-restricted square packing by defining $a_i = x_{k(i-1)+1}$ and $b_i = |X_i|$ for $1 \leq i \leq m$. I.e. we round up the size of every square in a group to equal the size the largest square in the group. Call this instance ς. We solve ς using RESTRICTED BIG SQUARES. To obtain a packing for $\sigma_>$, for $1 \leq i \leq m$, we replace each item of size a_i with an item in X_i. We call this algorithm for packing $\sigma_>$ BIG SQUARES (BS).

Lemma 6.1 *For all σ, Δ and m,*

$$\mathrm{cost}_{\mathrm{BS}}(\sigma_>) \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) + k.$$

Proof First note that $\mathrm{cost}_{\mathrm{BS}}(\sigma_>) = \mathrm{cost}_{\mathrm{RBS}}(\varsigma)$. Define another instance ς' of A-restricted square packing by setting $a_i = x_{ki+1}$ and $b_i = |X_i| = k$ for $1 \leq i \leq m-1$. I.e. we round down the size of every square in a group to equal the size the largest square in the next group and throw away all squares in the last group. Clearly,

$$\mathrm{cost}_{\mathrm{RBS}}(\varsigma) \geq \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) \geq \mathrm{cost}_{\mathrm{RBS}}(\varsigma').$$

Suppose we have a packing for ς' generated by RESTRICTED BIG SQUARES. Then, as in [13], we can obtain a packing for ς as follows: For $1 \leq i \leq m-1$, replace each square of size a_i in the solution with some item in X_{i+1}. This is possible since $|X_i| \geq |X_{i+1}|$ for $1 \leq i \leq m-1$. Place each of the k items in X_1 in a separate bin. The cost of this solution is at most $\mathrm{cost}_{\mathrm{RBS}}(\varsigma') + k$. Since RESTRICTED BIG SQUARES is an optimal algorithm, we have therefore have $\mathrm{cost}_{\mathrm{RBS}}(\varsigma) = \mathrm{cost}_{\mathrm{OPT}}(\varsigma) \le \mathrm{cost}_{\mathrm{RBS}}(\varsigma') + k$, and the desired result follows. ∎

Call a pattern which contains a single item of size greater than $\frac{1}{2}$ a *singleton*. To pack the items in $\sigma_{\le}$, we first use those bins packed according to a singleton pattern. To do this, suppose we have a bin which contains only an item of size $y \in (\frac{1}{2}, 1 - \sqrt{\Delta}]$. We place this item in the lower left corner and divide the remaining area into two rectangular areas, one of width 1 and height $1 - y$ and the other of width $1 - y$ and height y. We sort the items in $\sigma_{\le}$ and use the NEXT FIT DECREASING algorithm to pack items in these areas, as described in [27]. If we run out of these rectangular areas, then we continue to use NEXT FIT DECREASING but with newly allocated bins.

We choose

$$\Delta = \frac{\varepsilon^2}{4(4+\varepsilon)^2}, \qquad m = \left\lceil \frac{8(4+\varepsilon)^2}{\varepsilon^3} \right\rceil.$$

These choices guarantees that $\Delta < 1/4$ and

$$\frac{4\sqrt{\Delta}}{1 - 2\sqrt{\Delta}} \le \frac{\varepsilon}{2}, \qquad \frac{1}{\Delta m} \le \frac{\varepsilon}{2}.$$

We now analyze the performance of SQUARE SCHEME. Note that $\Delta|\sigma_>| \le \mathrm{cost}_{\mathrm{OPT}}(\sigma_>)$ and so $|\sigma_>| \le \mathrm{cost}_{\mathrm{OPT}}(\sigma_>)/\Delta \le \mathrm{cost}_{\mathrm{OPT}}(\sigma)/\Delta$. In the case that all small items can be packed in singleton bins, we have

$$\begin{aligned} \mathrm{cost}_{\mathrm{SS}}(\sigma) &= \mathrm{cost}_{\mathrm{BS}}(\sigma_>) \\ &\le \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) + k \\ &\le \mathrm{cost}_{\mathrm{OPT}}(\sigma) + \frac{|\sigma_>|}{m} + 1 \\ &\le \mathrm{cost}_{\mathrm{OPT}}(\sigma) + \frac{\mathrm{cost}_{\mathrm{OPT}}(\sigma)}{\Delta m} + 1 \\ &\le \left(1 + \tfrac{\varepsilon}{2}\right) \mathrm{cost}_{\mathrm{OPT}}(\sigma) + 1 \end{aligned}$$

The other case, where bins must be allocated to hold some small items, proves to be more complicated. The following result, directly adapted from [27], will prove to be very useful:

Lemma 6.2 (Meir & Moser) *Let ς be a list of squares, of which the largest is of size z. ς can be packed in a rectangle of height $x \ge z$ and width $y \ge z$ using* NEXT FIT DECREASING *if the total area of items in ς is at most $z^2 + (x - z)(y - z)$.*

We first bound the amount of available area in non-singleton bins:

Lemma 6.3 *The waste of any dominant pattern which is not a singleton is at most $\frac{5}{9}$.*

Proof There are five cases, depending on the number of items of size greater than $1/3$ in the pattern. There are at most four such items, so there are five cases.

In the first case, there are exactly four items of size greater than $1/3$, and the waste is at most $1 - 4/3^2 = 5/9$.

In the second case, we have a set of items, and only one is larger than $1/3$. Let the size of the largest item be $x > 1/3$. We place it in the lower left corner of the bin. Let $y \le 1/3$ be the size of the next largest item. We use the $1 \times (1 - x)$ rectangle above the largest item to pack items using NEXT FIT DECREASING. Let $z \le y$ be the size of the largest item among those still remaining, if there are any, and 0 otherwise. We now use the $(1 - x) \times x$ rectangles to the right of the largest item to pack any remaining items, also using NEXT FIT DECREASING. The total area that can be packed is at least $x^2 + y^2 + (1-y)(1-x-y) + z^2 + (x-z)(1-x-z)$. Call this expression f. Note that the size of the smallest item in the pattern is at most z. If the total area of items in the pattern is at most $f - z^2$, we have a contradiction, since we can increase the cardinality of the smallest item. So the waste is at most $1 - f + z^2 = 2y - xy - 2y^2 + z - z^2 \leq 5/9$.

In the remaining cases, we assume for a contradiction that the total area of items in some non-singleton dominant pattern is less than 4/9. We show that any set of items with total area 5/9 can be packed into a bin. Since there are less than four items of size greater than 1/3, and the pattern is not a singleton, there is some item of size at most 1/3. The area of this item is at most 1/9. Therefore the cardinality of this size item can be incremented, and we have a contradiction.

In the third case, we have a set of items, none of which are larger than 1/3, and the largest item is of size x. Then by Lemma 6.2 we can use NEXT FIT DECREASING to pack them as long as their total area is at most $x^2 + (1-x)(1-x) = 1 - 2x + 2x^2 \geq 5/9$.

In the forth case, we have a set of items, two of which are larger than 1/3, and the largest is of size x. We place the largest in the lower left corner, and the other adjacent to it to the right. We then use the $1 \times (1-x)$ area above to pack remaining items. Let the size of the largest of these remaining items be y. The total area that can be packed is at least $x^2 + (1/3)^2 + y^2 + (1-y)(1-x-y) = 2/3 - 2y/3 + y^2 \geq 5/9$.

In the fifth case, there are exactly three items larger than 1/3, and the largest is of size x. We place the largest in the lower left corner, and the others adjacent to it to the right and above. We then use the $(1-x) \times (1-x)$ area above and to the right to pack remaining items. Let y be the size of the largest of these items. The total area that can be packed is at least $x^2 + 2(1/3)^2 + y^2 + (1-x-y)(1-x-y) \geq 5/9$.
■

We now bound the amount of usable area in singleton bins:

Lemma 6.4 *For* $0 \leq \Delta \leq 1/4$, NEXT FIT DECREASING *can pack any set of items having total area at most*

$$(1 - 2\sqrt{\Delta})(1 - y^2)$$

in a singleton bin containing an item of size $y \leq 1 - \sqrt{\Delta}$.

Proof The item of size y is placed in the lower left corner. Consider first the area of size $1 \times (1-y)$. By Lemma 6.2, any set of items of total area

$$z^2 + (1-z)(1-y-z)$$

can be packed into this area, where $z \leq \Delta$ is the size of the largest item. This is decreasing in z for $z < (2-y)/4$. We have $z \leq \Delta < 1/4 \leq (2-y)/4$, and therefore, the total area packed is at least

$$\Delta^2 + (1-\Delta)(1-y-\Delta)$$

The ratio of this to the available area is

$$\frac{\Delta^2 + (1-\Delta)(1-y-\Delta)}{1-y}$$

This is strictly decreasing in y for $0 \leq \Delta < 1/4$ and so it is at at most

$$\frac{1 - \sqrt{\Delta} - 2\Delta + \Delta^{3/2} + 2\Delta^2}{1 - \sqrt{\Delta}} \geq 1 - 2\sqrt{\Delta},$$

for $\Delta < 1/4$.

Now consider the area of size $y \times (1-y)$. By similar arguments, the ratio of area used to total area is

$$\frac{\Delta^2 + (y-\Delta)(1-y-\Delta)}{y(1-y)} = 1 - \frac{\Delta(1-2\Delta)}{y(1-y)} \geq \frac{1-2\sqrt{\Delta}+2\Delta^{3/2}}{1-\sqrt{\Delta}} \geq 1-2\sqrt{\Delta},$$

for $\Delta < 1/4$.

The total area available in the bin is $1-y^2$. ■

Define $\psi(y)$ to be the number of singleton bins containing an item of size y in the solution produced by BIG SQUARES and $\Psi = \sum_{1/2<y\leq 1} \psi(y)$. Further define S to be the total area of items in $\sigma_{\leq}$. We first note that

$$\mathrm{cost}_{\mathrm{OPT}}(\sigma) \geq \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) + S - \sum_{1/2<y\leq 1} (1-y^2)\psi(y) - \tfrac{5}{9}(\mathrm{cost}_{\mathrm{OPT}}(\sigma_>) - \Psi)$$

and so

$$-\sum_{1/2<y\leq 1} (1-y^2)\psi(y) \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma) - S - \tfrac{4}{9}\,\mathrm{cost}_{\mathrm{OPT}}(\sigma_>) - \tfrac{5}{9}\Psi.$$

Consider the cost to SQUARE SCHEME. We need to determine the total number of bins allocated to hold only small items. The total available area in singleton bins containing an item of size greater than $1-\sqrt{\Delta}$ is at most $2\sqrt{\Delta}\Psi$. Therefore, into singleton bins containing an item of size at most $1-\sqrt{\Delta}$, the algorithm can pack items with total area at least

$$(1-2\sqrt{\Delta})\left(\sum_{1/2<y\leq 1} (1-y^2)\psi(y) - 2\sqrt{\Delta}\Psi\right).$$

Each of the bins which is allocated to only small items, except for the last, holds items of total area at least $2\Delta^2 + 1 - 2\Delta > 1-2\Delta > 1-2\sqrt{\Delta}$. We therefore have

$$\begin{aligned}
\mathrm{cost}_{\mathrm{SS}}(\sigma) &\leq \mathrm{cost}_{\mathrm{BS}}(\sigma_>) + \frac{1}{1-2\sqrt{\Delta}}\left(S - (1-2\sqrt{\Delta})\left(\sum_{1/2<y\leq 1}(1-y^2)\psi(y) - 2\sqrt{\Delta}\Psi\right)\right) + 1 \\
&= \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) + k + \frac{S}{1-2\sqrt{\Delta}} - \sum_{1/2<y\leq 1}(1-y^2)\psi(y) + 2\sqrt{\Delta}\Psi + 1 \\
&\leq \mathrm{cost}_{\mathrm{OPT}}(\sigma_>) + k + \frac{S}{1-2\sqrt{\Delta}} + \mathrm{cost}_{\mathrm{OPT}}(\sigma) - S - \tfrac{4}{9}\,\mathrm{cost}_{\mathrm{OPT}}(\sigma_>) - \tfrac{5}{9}\Psi + 2\sqrt{\Delta}\Psi + 2 \\
&\leq \left(\frac{14}{9} + \frac{1}{\Delta m} + \frac{4\sqrt{\Delta}}{1-2\sqrt{\Delta}}\right)\mathrm{cost}_{\mathrm{OPT}}(\sigma) + 2 \\
&\leq \left(\tfrac{14}{9} + \tfrac{\varepsilon}{2} + \tfrac{\varepsilon}{2}\right)\mathrm{cost}_{\mathrm{OPT}}(\sigma) + 2 \\
&\leq \left(\tfrac{14}{9} + \varepsilon\right)\mathrm{cost}_{\mathrm{OPT}}(\sigma) + 2.
\end{aligned}$$

Here we have used that $S \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma)$, $\Psi \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma)$ and $k \leq \mathrm{cost}_{\mathrm{OPT}}(\sigma)/(\Delta m)$.

7. LOWER BOUNDS FOR ONLINE BOX AND SQUARE PACKING

In this section, we show lower bounds on bounded space algorithms for box and square packing and unrestricted algorithms for square packing.

To begin, we consider bounded space algorithms for d-dimensional box packing. We say that a bounded space online algorithm $\mathcal{A}$ is in the class BS(i) if at any point in time it has at most i bins open. We have the following negative result:

Theorem 7.1 *No algorithm for d-dimensional box packing in* BS$(d-1)$ *has bounded asymptotic performance ratio.*

Proof Let k and n be a positive integers. We consider an input σ of length dn consisting of d different types of items. An item p of type i has $s_i(p) = 1/k$ and $s_j(p) = 1$ for $j \neq i$. The jth item in σ has type $(j \bmod d) + 1$. The optimal offline cost for the sequence is $d\lceil n/k \rceil$. Note that items of differing types cannot be placed in the same bin. The cost to any algorithm in BS$(d-1)$ is at least n: For every d items processed, the algorithm is forced to close at least one bin, since it can keep only bins containing $d-1$ types open. Therefore, the asymptotic performance ratio of any BS$(d-1)$ algorithm is at least

$$\lim_{n\to\infty} \frac{n}{d\lceil n/k \rceil} = \frac{k}{d},$$

and k can be arbitrarily large. ■

We extend Liang's sequence [26] to d dimensions as follows. The input consists of k phases. The ith phase consists of mb_i hyper-cubes with size $a_i = 1/\pi_i + \varepsilon$, where $b_i = \prod_{j=1}^{i-1}(\pi_j^d - (\pi_j - 1)^d)$. Note that at most $(\pi_i - 1)^d$ items of size $1/\pi_i + \varepsilon$ fit together in a bin. We denote this input sequence by L, and write $A = \{a_1, \dots, a_k\}$. Denote by L_i the input sequence containing only the i smallest items in L. Algorithm $\mathcal{A}$ is given L_i for some $i \in \{1, \dots, k\}$. Note that the smallest item a_k appears first, and the largest item appears last. For a fixed m, define $\chi_i(m)$ to be the optimal offline cost for packing the items in L_i. We have the following lemma.

Lemma 7.1 *For* $1 \leq i \leq k$, $\chi_i(m) = mb_i/(\pi_i - 1)^d$.

Proof We start with the case $i = k$. We show that the cost for packing all items is at most m. Each of the m bins is packed identically, so we merely describe how a single bin is packed. We pack the items by phase. The packing is recursive, in the sense that at each point we have a number of equal sized hyper-cubes into which we place a single item (all of these hyper-cubes lie within the original bin). The number of hyper-cubes available after items from the first $i-1$ phases have been packed is $\prod_{j=1}^{i-1}(\pi_j^d - (\pi_j - 1)^d)$. To start, we have a single hyper-cube of size 1, which is just the bin itself. We place a single phase 1 item, an item of size $1/2 + \varepsilon$, so that one of its corners coincides with one of the corners of the bin. In the remaining space, we allocate $2^d - 1$ hyper-cubes of size $1/2 - \varepsilon$ (leaving a small amount of wasted space along some edges). We now pack items of the second phase, items of size $1/3 + \varepsilon$. Each of these is placed in one of the hyper-cubes left from the first phase, with one of its corners coinciding with a corner of the hyper-cube. In general, to pack the items of phase i, we use the $\prod_{j=1}^{i-1}(\pi_j^d - (\pi_j - 1)^d) = b_i$ hyper-cubes of size $1/(\pi_i - 1) - (i-1)\varepsilon$ left over from the previous phase, and in the remaining space allocate $\prod_{j=1}^{i}(\pi_j^d - (\pi_j - 1)^d) = b_{i+1}$ hyper-cubes of size $1/(\pi_{i+1} - 1) - i\varepsilon$. We illustrate the packing for $d = 2$ and $k = 4$ in Figure 3.

For $1 \leq i < k$, we need $mb_i/(\pi_i - 1)^d$ bins to pack all the items of size a_i, since there are mb_i such items. We show that we can pack all smaller items with them in the same bins, proving the lemma. We use a recursive packing similar to the one for the complete list, reserving hypercubes for the items. However, we now start with the items of size $a_i = 1/\pi_i + \varepsilon$, putting them in hypercubes of size $1/(\pi_i - 1) - (i-1)\varepsilon$. Then, the total number of hypercubes over all the bins for items of size a_i is $(\pi_i - 1)^d mb_i/(\pi_i - 1)^d = mb_i$. This is the same total number as we had above in the packing for L. This implies that if we continue to pack as described for L, we can pack all the items of sizes $a_{i+1}, \dots, a_k$ in these $mb_i/(\pi_i)^d$ bins. ■

We start with the result for bounded space algorithms.

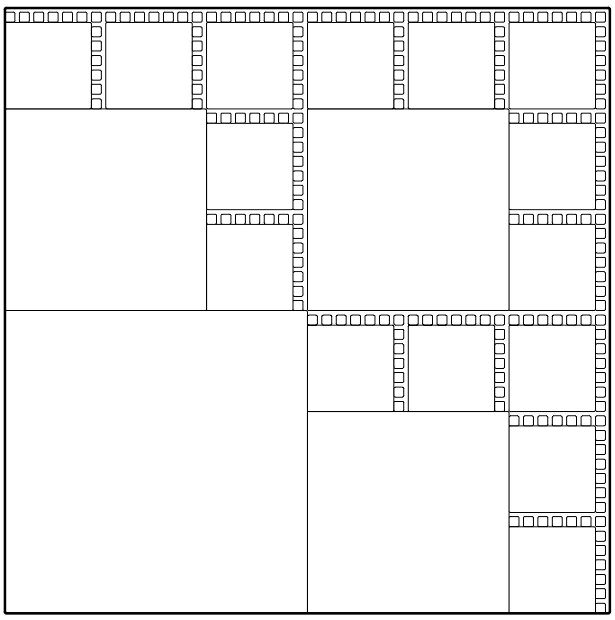

Figure 3: A bin in the optimal offline packing of L with $d = 2$ and $n = 4$.

Theorem 7.2 *The asymptotic performance ratio of any bounded space online square packing algorithm is at least*

$$\sum_{i=1}^{k} \frac{1}{(\pi_i - 1)^d} \prod_{j=1}^{i-1} (\pi_j^d - (\pi_j - 1)^d),$$

for all $k \geq 1$.

Proof Let $\varepsilon < 1/(k(\pi_{k+1} - 1))$ be a small positive real number and m be a large positive integer. We give any bounded space online square packing algorithm the input L defined above. Since only a constant number of items from phase i can be packed with items from previous phases, the number of bins used in phase i is at least $m(\pi_i - 1)^{-d} \prod_{j=1}^{i-1} (\pi_j^d - (\pi_j - 1)^d) - O(1)$. Since k is a constant, the total number of bins used by the algorithm for all phases is

$$m \sum_{i=1}^{k} \frac{1}{(\pi_i - 1)^d} \prod_{j=1}^{i-1} (\pi_j^d - (\pi_j - 1)^d) - O(1).$$

Since the optimal number of bins to pack L is m by Lemma 7.1, we are done. ■

Note that this gives the bound of Lee and Lee for $d = 1$ [24]. Numeric values for this lower bound are calculated in Table 1.

Next we give a lower bound for general (unbounded space) algorithms.

Lemma 7.2 *Consider a packing pattern* $P = (P_1, \ldots, P_k)$ *for the input* L. *Then*

$$P_i \leq (\pi_i - 1)^d - \sum_{j=1}^{i-1} P_j \left(\frac{\pi_i - 1}{\pi_j} \right)^d. \tag{7.1}$$

Proof Suppose (7.1) does not hold for some i, and consider the smallest i for which it does not hold. Consider an item of size a_j with $j < i$ that appears in P. If there is no such j, we have a contradiction, since at most $(\pi_i - 1)^d$ items of size a_i fit in the unit hypercube.

k	$d=1$	$d=2$	$d=3$	$d=4$
1	1	1	1	1
2	1.5	1.75	1.875	1.9375
3	1.66667	2.16666	2.49074	2.68981
4	1.69047	2.27721	2.71872	3.03604
5	1.69102	2.28229	2.73426	3.06715
6	1.69103	2.28229	2.73429	3.06721
7	1.69103	2.28229	2.73429	3.06721

Table 1: Values calculated from Theorem 7.2.

If we replace an item os size a_j with $(\frac{\pi_i-1}{\pi_j})^d$ items of size a_i, the resulting pattern is still feasible: all the new size a_i items can be placed inside the hypercube that this size a_j item has vacated.

We can do this for all items of size a_j, $j < i$ that appear in the pattern. This results in a pattern with only items of size a_i or smaller. Since every size a_j item is replaced by $(\frac{\pi_i-1}{\pi_j})^d$ items of size a_i, the final pattern has more than $(\pi_i - 1)^d$ items of size a_i, a contradiction. ∎

Dominant patterns were defined in the previous section. We now also define greedy patterns.

Definition A pattern $P = (P_1, \dots, P_k)$ is *greedy* if the largest item in it appears as many times as it can fit in a bin, and each successive item that appears, appears as many times as it can be added to the bin given the previous (larger) items in the pattern.

Lemma 7.3 *For the input L, any dominant pattern that is not greedy is a convex combination of dominant patterns that are greedy.*

Proof We use an induction to construct a convex combination of greedy patterns.

Consider a pattern P. Clearly, the smallest item in P appears as many times as it can fit given the larger items, because P is dominant.

Suppose item i appears not as many times as it could, given the larger items. By induction, we only need to consider patterns in which all the smaller items that appear, appear as many times as possible, starting with the largest smaller item. (All other patterns are convex combinations of such patterns.)

We define two patterns P' and P'' such that P is a convex combination of them. P' is defined as follows: modify P by removing all items i and adding smaller items, starting with the largest smaller items that appear in P, and each time adding as many items as possible.

P'' on the other hand is created by adding items of phase i to P and removing smaller items, until the number of items of phase i is maximized, given the larger items. Each time that we add an item i, we remove the maximum amount of smaller items that would fit inside the area of an item i, starting with the largest smaller items. We can do this because we know that the number of these items is maximal (starting with the largest), so the number of items that fit inside the area of an item i is certainly present if the full number of items i is not present.

This implies that by adding an item i in creating P'', we remove exactly the same numbers of smaller items as we add when we remove an item i while creating P'. Therefore, P is a convex combination of P' and P'', and we are done. ∎

Define A^* to be the set of all patterns P with respect to A. Note that A^* is necessarily finite. Given an input sequence of items, and its length, an algorithm is defined by the numbers and types of items it places in each of the bins it uses. Specifically, any algorithm is defined by a function $\Phi : A^* \times \mathbb{N} \mapsto \mathbb{R}_{\geq 0}$. The algorithm uses $\Phi(P, m)$ bins containing items as described by the pattern P. Here we assume that the ratios between the amounts of times that items of sizes a_i and a_j appear are fixed for all $i \neq j$, and hence that the length of the input sequence is determined by a single parameter m.

Consider the function Φ that determines the packing used by online algorithm $\mathcal{A}$ uses for L_k. Since $\mathcal{A}$ is online, the packings it uses for $L_1, \ldots, L_{k-1}$ are completely determined by Φ. We assign to each pattern an *order*, which is defined

$$\operatorname{order}(P) = \max\{i \mid P_i \neq 0\}.$$

Intuitively, the order tells us the first item size a_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 order i, then these bins will contain one or more items after L_i. Define

$$\mathcal{A}_i^* = \{P \in A^* \mid \operatorname{order}(P) \geq i\}.$$

Then if $\mathcal{A}$ is determined by Φ, its cost for L_i is simply

$$\sum_{P \in \mathcal{A}_i^*} \Phi(P, m).$$

Since the algorithm must pack every item, we have the following constraints

$$\sum_{P \in A^*} \Phi(P, m)\, P_i \geq m b_i, \qquad \text{for } 1 \leq i \leq k. \tag{7.2}$$

Define $\chi_i^* = \chi_i / m$ for $i = 1, \ldots, k$. By Lemma 7.1, $\chi_i^* = b_i/(\pi_i - 1)^d$ for $i = 1, \ldots, k$. We can now readily compute a lower bound for online algorithms:

Lemma 7.4 *The optimal value of the linear program: Minimize c subject to*

$$\begin{aligned} c &\geq \frac{1}{\chi_i^*} \sum_{P \in \mathcal{A}_i^*} \phi_P, && \textit{for } 1 \leq i \leq k; \\ b_j &\leq \sum_{P \in A^*} \phi_P\, P_i, && \textit{for } 1 \leq i \leq k; \end{aligned} \tag{7.3}$$

over variables c and $\phi(P), P \in A^*$, is a lower bound on the asymptotic performance ratio of any online bin packing algorithm.

Proof For any fixed m, any algorithm $\mathcal{A}$ has some Φ which must satisfy (7.2). Further, Φ should assign an integral number of bins to each pattern. However, in the LP this integrality constraint is relaxed, and $\sum_{P \in \mathcal{A}_i^*} \phi_P$ is $1/m$ times the cost to $\mathcal{A}$ for L_i as $m \to \infty$. (I. e. we introduce decision variables ϕ_P to represent $\Phi(P, m)/m$.) The value of c is then just the maximum of the performance ratios achieved on $L_1, \ldots, L_k$. ∎

As discussed in [33], we only need to consider dominant patterns. By Lemma 7.3, we can even restrict ourselves to greedy patterns.

Any greedy pattern can be represented by a sequence of k bits, where k is the number of items in the item sequence, and bit i indicates whether or not item i appears in this pattern. Hence the number of these patterns is 2^k. For any given greedy pattern, we can calculate the number of times that each item appears, using Lemma 7.2 and starting with the largest item that appears.

All we have to do then is compute the value of the LP given in Lemma 7.4. Solving this linear program for several values of d and k gives us the results shown in Table 2.

k	$d = 1$	$d = 2$	$d = 3$	$d = 4$
1	1	1	1	1
2	1.33333	1.23076	1.12280	1.06224
3	1.50000	1.48453	1.36801	1.24674
4	1.53900	1.61235	1.57453	1.51173
5	1.54014	1.62175	1.60180	1.55679
6	1.54014	1.62176	1.60185	1.55690

Table 2: Lower bounds for unbounded space algorithms

8. An Upper Bound for Online Square Packing

In this section, we consider the performance of the H $\times$ H algorithm for $d = 2$ when all items are squares. From Theorem 4.1 we have that the performance ratio of this algorithm is at most Π_n^2 (which is approximately 2.85958 for large n), however, we seek to show a better result. By using the fact that all items are squares, we show the following result:

Theorem 8.1 *For $n \geq 7$, the asymptotic performance ratio of* H $\times$ H *is at most* $395/162 < 2.43828$ *for the square packing problem with $d = 2$.*

Before the theorem can be shown we require the following technical lemma:

Lemma 8.1 *If a square of size strictly greater than $1/2$ is packed in the unit square then at most 5 squares of size strictly greater than $1/4$ can packed with it.*

Proof Call the item of size strictly greater than $1/2$ the *big* item, and the others *small* items. We show that without loss of generality, the lower left corner of the big item coincides with the lower left corner of the unit square. First we show that some corner of the big item can be made to coincide with some corner of the unit square. If this is not already true, note that one horizontal edge of the big item is at a distance strictly less than $1/4$ away from a horizontal edge of the unit square. No small item fits in the space between, and so we can move the big item until the two sides coincide, without creating overlap with a smaller item. Now the same can be done in the vertical direction, and one corner coincides. We can now rotate everything until the coinciding corners are in the lower left.

Now we divide the unit square into 4 equal quadrants. The lower left quadrant is entirely occupied by the big item. At most one small item can be contained entirely within each quadrant apart from the lower left quadrant. In addition, no small item can overlap with the lower left quadrant. At most one small item can overlap with both the two top quadrants, and at most one small item can overlap with both the two right quadrants. Putting these facts together, the total number of small items is at most 5. ■

Proof of Theorem 8.1 By the results of Section 4, the asymptotic performance ratio of H $\times$ H is upper bounded by the maximum amount of weight that fits in a single bin. Recall that a square has type $i \leq n$ if its size is in $(1/(i+1), 1/i]$. The weight of a type $i \leq n$ item is just $1/i^2$. Define the *expansion* of a type $i \leq n$ item to be $(i+1)^2/i^2$. Intuitively, the expansion is the maximum ratio of weight to size for a type i item. Recall that $\epsilon = 1/(n-1)$ and so $\epsilon \leq 1/6$. Items of type $n+1$ have size in $s \in (0, \epsilon]$ and weight $s/((1-\delta)(1-\epsilon)^2)$. Define the expansion of such an item to be $1/((1-\delta)(1-\epsilon)^2)$. Since $\epsilon \leq 1/6$ we can pick δ so that the expansion of type $n+1$ items is at most $25/16$.

The proof now proceeds by considering a number of cases, depending on the contents of the bin:

1. The bin contains no type 1 item.

2. The bin contains one type 1 item and $i < 3$ type 2 items.

3. The bin contains one type 1 item, 3 type 2 items and $i \leq 2$ type 3 items.

We first need to show that these cases are the only possible ones. This follows from two facts: 1) at most four items of types 1 and 2 fit in a bin, 2) by Lemma 8, if there is a type 1 item in a bin, no more than five items of types 2 or 3 fit in that bin.

We now bound the total weight of items in each case.

In the first case, note that the expansion of all items is at most $9/4$, while their total area is at most 1. Therefore the total weight in the bin is no more than $9/4 < 395/162$.

In the second case, the weight of the type 1 and 2 items is $1 + i/4$. The remaining area is at most $1 - 1/4 - i/9$. The expansion of all remaining items is at most $16/9$. So the total weight is at most $1 + i/4 + 16/9(1 - 1/4 - i/9) = 7/3 + 17i/324 \leq 395/162$ for $0 \leq i \leq 2$.

In the third case, using similar arguments to those in the previous ones, the total weight is at most $1 + 3/4 + i/9 + 25/16(1 - 1/4 - 3/9 - i/16) = 461/192 + 31i/2304 < 395/162$ for $0 \leq i \leq 2$. ■

Note that the lower bound of Theorem 7.2 for $d = 2$ also applies to $\mathrm{H} \times \mathrm{H}$, so its performance ratio is at least 2.28229.

9. Conclusions

We have improved the upper bound for online two-dimensional bin packing from 2.85958 to 2.66013, and also for the special case of two-dimensional packing of squares, both online (2.56411 to 2.43828) and offline (1.988 to $14/9 + \varepsilon$). Furthermore, we have improved the lower bound for square and hypercube packing, and given the first lower bounds for bounded space square and hypercube packing. Our results also imply the following results:

- $\mathrm{H} \times \mathcal{B}$ generalizes easily to $d > 2$, but the bound we get is $r(\Pi_\infty)^{d-1}$. It is not clear what the performance of $\mathrm{H} \times \mathrm{MH}$ is, but it is at most $2.66013(\Pi_\infty)^{d-2}$.
- In the two-sized bin packing problem, bins have one of two sizes, either 1 or $z < 1$. In the two-sized two-dimensional box packing problem, bins have one of two sizes, either 1×1 or $1 \times z$. The algorithm chooses the size of a bin when it is opened. The cost of a bin is equal to its area. The Variable Harmonic (VH) algorithm [7, 31], is an optimal bounded space algorithm for the two-sized bin packing problem. It performance ratio is a function $R^\infty_{\mathrm{VH}}(z)$. For the two-sized two-dimensional box packing problem, the performance ratio of $\mathrm{H} \times \mathrm{VH}$ is $R^\infty_{\mathrm{VH}}(z)\,\Pi_\infty$. The results of [31] imply that for $z = 5/7$ this is at most 2.32571.
- In the resource augmented box packing problem, the algorithm is allowed to have larger bins than the adversary. The cost of each bin is one. In [11], for the one-dimensional case where the algorithm is allowed bins of size z, it is shown that the performance ratio of Harmonic is a function $R^\infty_{\mathrm{H}}(z)$. It is easy to show that the performance ratio of $\mathrm{H} \times \mathrm{H}$ is $(R^\infty_{\mathrm{H}}(z))^2$ for the two-dimensional case where the algorithm uses $z \times z$ bins.

There are also a number of open problems. A main problem is that the analysis does not go through if we chose $\mathcal{B}$ to be Harmonic++. The problem is that we rely on Lemma 4.1. To get better results, we need some stronger way of bounding the weight in a bin.

Related open problems are the two-dimensional box packing problem with rotations [15], the vector packing problem [18, 23], and strip packing [1, 12, 25]. Furthermore, in the two-sized two-dimensional bin packing problem it is also possible to have bin sizes of 1×1 and $z \times z$. It is unclear what the performance ratio is, but the techniques given here should be applicable.

References

1. Baker, B. S., and Schwartz, J. S. Shelf algorithms for two-dimensional packing problems. *SIAM Journal on Computing 12* (1983), 508–525.
2. Blitz, D., van Vliet, A., and Woeginger, G. J. Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms. Unpublished manuscript, 1996.
3. Brown, D. J. A lower bound for on-line one-dimensional bin packing algorithms. Tech. Rep. R-864, Coordinated Sci. Lab., University of Illinois at Urbana-Champaign, 1979.
4. Chung, F. R. K., Garey, M. R., and Johnson, D. S. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods 3* (1982), 66–76.
5. Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-hard Problems*, D. Hochbuam, Ed. PWS Publishing Company, 1997, ch. 2.
6. Coppersmith, D., and Raghavan, P. Multidimensional online bin packing: Algorithms and worst case analysis. *Operations Research Letters 8* (1989), 17–20.
7. Csirik, J. An on-line algorithm for variable-sized bin packing. *Acta Informatica 26*, 8 (1989), 697–709.
8. Csirik, J., Frenk, J., and Labbe, M. Two-dimensional rectangle packing: on-line methods and results. *Discrete Applied Mathematics 45*, 3 (Sep 1993), 197–204.
9. Csirik, J., and van Vliet, A. An on-line algorithm for multidimensional bin packing. *Operations Research Letters 13*, 3 (Apr 1993), 149–158.
10. Csirik, J., and Woeginger, G. On-line packing and covering problems. In *On-Line Algorithms—The State of the Art*, A. Fiat and G. Woeginger, Eds., Lecture Notes in Computer Science. Springer-Verlag, 1998, ch. 7.
11. Csirik, J., and Woeginger, G. Resource augmentation for online bounded space bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming* (Jul 2000), pp. 296–304.
12. Csirik, J., and Woeginger, G. J. Shelf algorithms for on-line strip packing. *Information Processing Letters 63* (1997), 171–175.
13. de la Vega, W. F., and Lueker, G. S. Bin packing can be solved within $1 + \epsilon$ in linear time.

Combinatorica 1, 4 (1981), 349–355.

14. Ferreira, C. E., Miyazawa, F. K., and Wakabayashi, Y. Packing squares into squares. *Pesquisa Operacional 19*, 2 (1999), 223–237.
15. Fujita, S., and Hada, T. Two-dimensional on-line bin packing problem with rotatable items. In *6th Annual International Conference on Computing and Combinatorics* (2000), pp. 210–20.
16. Galambos, G. A 1.6 lower-bound for the two-dimensional online rectangle bin-packing. *Acta Cybernetica 10*, 1-2 (1991), 21–24.
17. Galambos, G., and van Vliet, A. Lower bounds for 1, 2 and 3-dimensional online bin packing algorithms. *Computing 52* (1994), 281–297.
18. Garey, M., Graham, R., Johnson, D., and Chi-Chih Yao, A. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A 21*, 3 (Nov 1976), 257–298.
19. Johnson, D. S. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.
20. Johnson, D. S. Fast algorithms for bin packing. *Journal Computer Systems Science 8* (1974), 272–314.
21. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing 3* (1974), 256–278.
22. Karmarkar, N., and Karp, R. M. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (1982), pp. 312–320.
23. Kou, L., and Markowsky, G. Multidimensional bin packing algorithms. *IBM Journal of Research and Development 21*, 5 (Sep 1977), 443–448.
24. Lee, C., and Lee, D. A simple on-line bin-packing algorithm. *Journal of the ACM 32*, 3 (Jul 1985), 562–572.
25. Li, K., and Cheng, K. H. Heuristic algorithms for online packing in three dimensions. *Journal of Algorithms 13* (1992), 589–605.
26. Liang, F. M. A lower bound for online bin packing. *Information Processing Letters 10* (1980), 76–79.
27. Meir, A., and Moser, L. On packing of squares and cubes. *Journal of Combinatorial Theory 5* (1968), 126–134.
28. Ramanan, P., Brown, D., Lee, C., and Lee, D. On-line bin packing in linear time. *Journal of Algorithms 10*, 3 (Sep 1989), 305–326.
29. Richey, M. B. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics 34* (1991), 203–227.
30. Seiden, S. On the online bin packing problem. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming* (Jul 2001), pp. 237–249.
31. Seiden, S. S. An optimal online algorithm for bounded space variable-sized bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming* (Jul 2000), pp. 283–295.
32. van Vliet, A. An improved lower bound for online bin packing algorithms. *Information Processing Letters 43*, 5 (Oct 1992), 277–284.
33. van Vliet, A. *Lower and upper bounds for online bin packing and scheduling heuristics.* PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

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