Optimal Online Bounded Space Multidimensional Packing

Leah Epstein[*] Rob van Stee[†]

Abstract

We solve an open problem in the literature by providing an online algorithm for multidimensional bin packing that uses only bounded space. To achieve this, we introduce a new technique for classifying the items to be packed. We show that our algorithm is optimal among bounded space algorithms for any dimension $d > 1$. Its asymptotic performance ratio is $(\Pi_\infty)^d$, where $\Pi_\infty \approx 1.691$ is the asymptotic performance ratio of the one-dimensional algorithm HARMONIC. A modified version of this algorithm for the case where all items are hypercubes is also shown to be optimal. Its asymptotic performance ratio is sublinear in d.

Additionally, for the special case of packing squares in two-dimensional bins, we present a new unbounded space online algorithm with asymptotic performance ratio of at most 2.271. We also present an approximation algorithm for the offline problem with approximation ratio of $16/11$.

1 Introduction

Bin packing is one of the oldest and most well-studied problems in computer science [11, 6]. 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 *items* $p_1, p_2, \dots, p_n$. Each item 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 item must be assigned to a bin and a position $(x_1(p), \dots, x_d(p))$, 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 item 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.

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

- In the *online* version of this problem, each item must be assigned in turn, without knowledge of the next items.
- In the *hypercube packing* problem we have the restriction that all items are hypercubes, i.e. an item has the same size in every dimension.
- 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, and 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 $\text{cost}_{\mathcal{A}}(\sigma)$ be the number of bins used by algorithm $\mathcal{A}$ on σ. Let $\text{cost}(\sigma)$ be the minimum possible number of bins used to pack items in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

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

Let $\mathcal{O}$ be some class of box packing algorithms (for instance online algorithms). The *optimal asymptotic performance* ratio for $\mathcal{O}$ is defined to be $\mathcal{R}_{\mathcal{O}}^{\infty} = \inf_{\mathcal{A}\in\mathcal{O}} \mathcal{R}_{\mathcal{A}}^{\infty}$. Given $\mathcal{O}$, our goal is to find an algorithm with asymptotic performance ratio close to $\mathcal{R}_{\mathcal{O}}^{\infty}$.

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. lea@idc.ac.il. Research supported by Israel Science Foundation (grant no. 250/01).

[†]Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands. Rob.van.Stee@cwi.nl. Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-061-000.

Previous Results: The classic online bin packing problem was first investigated by Ullman [26]. He showed that the FIRST FIT algorithm has performance ratio $\frac{17}{10}$. This result was then published in [16]. Johnson [17] showed that the NEXT FIT algorithm has performance ratio 2. Yao showed that REVISED FIRST FIT has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [31]. Brown and Liang independently improved this lower bound to 1.53635 [3, 22]. The lower bound currently stands at 1.54014, due to van Vliet [27]. 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 presented an algorithm called HARMONIC, which uses $m > 1$ classes and uses bounded space. For any $\varepsilon > 0$, there is a number m such that the HARMONIC algorithm that uses m classes has a performance ratio of at most Π_∞ [20]. They also showed there is no bounded space algorithm with a performance ratio below Π_∞. Bounded space algorithms for bin packing were also considered by Woeginger in [30], and by van Vliet in [29]. Currently the best known upper bound is 1.58889 due to Seiden [24].

Offline bin packing has also received a great deal of attention, for a survey see [6]. The most prominent results are as follows: Garey, Graham and Ullman [16] were the first to study the approximation ratios of both online and offline algorithms. Fernandez de La Vega and Lueker [12] presented the first approximation scheme for bin packing. Karmarkar and Karp [18] gave an algorithm which uses at most $\text{cost}(\sigma) + \log^2(\text{cost}(\sigma))$ bins.

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 [5]. Caprara [4] presented an algorithm with approximation ratio Π_∞.

The online problem was first investigated by Coppersmith and Raghavan [7], who give an algorithm based on NEXT FIT with performance ratio $\frac{13}{4} = 3.25$ for $d = 2$. Csirik, Frenk and Labbe [9] give an algorithm based on FIRST FIT with performance ratio $\frac{49}{16} = 3.0625$ for $d = 2$. Csirik and van Vliet [10] present an algorithm with performance ratio $(\Pi_\infty)^d$ for all $d \geq 2$ (2.85958 for $d = 2$). Even though this algorithm is based on HARMONIC, it was not clear how to change it to bounded space. Li and Cheng [21] also gave a HARMONIC-based algorithm for $d = 2$ and $d = 3$.

Seiden and van Stee [25] improve the upper bound for $d = 2$ to 2.66013. Several lower bounds have been shown [14, 15, 28, 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 bounded space algorithms, a lower bound of $(\Pi_\infty)^d$ is implied by [10].

For online square packing, even less is known. The following results are known for $d = 2$: Coppersmith and Raghavan [7] show an upper bound of $43/16 = 2.6875$ and a lower bound of $4/3$. The upper bound is improved to $395/162 < 2.43828$ by Seiden and van Stee [25]. For $d = 3$, Miyazawa and Wakabayashi [23] show an upper bound of 3.954.

For the offline problem, Ferreira, Miyazawa and Wakabayashi give a 1.988-approximation algorithm [13]. This was improved to $14/9$ by Kohayakawa et al. [19] and Seiden and van Stee [25] independently. However, the result in [19] is more general in that the authors give two approximation algorithms for any dimension $d \geq 2$ with asymptotic performance bounds of $2 - (1/2)^d$ and $2 - (2/3)^d$.

Finally, the algorithm in [4] is shown to have an approximation ratio in $(1.490, 1.507)$ for this problem, if a certain conjecture holds.

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

- We begin by presenting a bounded space algorithm for the packing of hypercubes. An interesting feature of the analysis is that although we show the algorithm is optimal, we do not know the exact asymptotic performance ratio. The asymptotic performance ratio is $\Omega(\log d)$ and $O(d/\log d)$.
- We then extend this algorithm to a bounded space algorithm for general hyperbox packing and show that this algorithm is also optimal, with a asymptotic performance ratio of $(\Pi_\infty)^d$. This solves the ten-year old open problem of how to pack hyperboxes using only bounded space.
- We present a new unbounded space algorithm for two dimensional square packing with a asymptotic performance ratio of 2.2709, which cannot be attained by a bounded space algorithm.
- We improve the approximation algorithm for square packing from [25] and show that the resulting algorithm has an approximation ratio of at most $16/11 < 1.45455$. Independently and in parallel, Bansal and Sviridenko [1] and Correa and Kenyon [8] presented approximation schemes for this problem.

For the online results, we will use the technique of weighting functions. This technique was originally introduced for one-dimensional bin packing algorithms [26]. In [25], it was demonstrated how to use the analysis for one-dimensional algorithms to get results for higher dimensions. In contrast, in the current paper we will define weighting functions directly for multidimensional algorithms, without using one-dimensional algorithms as subroutines.

New Technique: To construct the bounded space algorithm we adapt some of the ideas used in previous work. Specifically, the algorithm of [10] also required a scheme of partitioning bins into sub-bins, and of sub-bins into smaller and smaller sub-bins. However, in order to keep a constant number of bins active, we had to introduce a new method of classifying items. Our key improvement is that there is not one single class of "small" items like all the standard algorithms have, but instead we partition the items into an infinite number of classes that are grouped into a finite number of groups. The hypercube packing algorithm uses an easier scheme for the same purpose. This is a more direct extension of the method used in [7].

2 An optimal algorithm for bounded space packing of hypercubes

In this section we define the algorithm for hypercubes, denoted by ALG_ε. In the next section we extend it to deal with hyperboxes. Let the *size* of hypercube q, $s(q)$ be the length of each side of the hypercube.

The algorithm has a parameter $\varepsilon > 0$. Let $M \geq 10$ be an integer parameter such that $M \geq 1/(1-(1-\varepsilon)^{1/(d+1)})-1$. We distinguish between "small" hypercubes (of size smaller or equal to $1/M$) and "big" hypercubes (of size larger than $1/M$). The packing algorithm will treat them in different ways.

All *large* hypercubes are packed using a multidimensional version of HARMONIC [20]. The hypercubes are assigned a type according to their size: type i items have a size in the interval $(1/i+1, 1/i]$ for $i = 1, \ldots, M-1$. The bins that are used to pack items of these types all contain items of only one type. We use the following algorithm to pack them. A bin is called *active* if it can still receive items, otherwise it is *closed*.

Algorithm ASSIGNLARGE(i) At all times, there is at most one active bin for each type. Each bin is partitioned into i^d hypercubes (sub-bins) of size $1/i$ each (the sub-bins create a grid of i strips in each dimension). Each such sub-bin can contain exactly one item of type i. On arrival of a type i item it is assigned to a free sub-bin (and placed anywhere inside this sub-bin). If all sub-bins are taken, the previous active bin is closed, a new active bin is opened and partitioned into sub-bins.

The *small* hypercubes are also assigned types depending on their size, but in a different way. Consider an item q of size $s(q) \leq 1/M$. Let k be the largest non-negative integer such that $2^k s(q) \leq 1/M$. Clearly $2^k s(q) > 1/(2M)$. Let i be the integer such that $2^k s(q) \in (1/i+1, 1/i]$, $i \in \{M, \ldots 2M-1\}$. The item is defined to be of type i. Each bin that is used to pack small items contains only small items with a given type i. Note that items of very different sizes may be packed together in one bin. We now describe the algorithm to pack a new small item of type i for $i = M, \ldots, 2M-1$. A sub-bin which received a hupercube is said to be *used*. A sub-bin which is not used and not cut into smaller sub-bins is called *empty*.

Algorithm ASSIGNSMALL(i) The algorithm maintains a single active bin. Each bin may during its use be partitioned into sub-bins which are hypercubes of different sizes of the form $1/(2^j i)$. When an item q of type i arrives we do the following. Let k be the integer such that $2^k s(q) \in (1/i+1, 1/i]$.

1. If there is an empty sub-bin of size $1/(2^k i)$, then the item is simply assigned there and placed anywhere within the sub-bin.
2. Else, if there is no empty sub-bin of any size $1/(2^j i)$ for $j > k$ inside the current bin, the bin is closed and a new bin is opened and partitioned into sub-bins of size $1/i$. Then the procedure in step 3 is followed, or step 1 in case $k = 0$.
3. Take an empty sub-bin of size $1/(2^j i)$ for a minimum $j > k$. Partition it into 2^d identical sub-bins (by cutting into two identical pieces, in each dimension). If the resulting sub-bins are of size larger than $1/(2^k i)$, take *one* of them and partition it in the same way. This is done until sub-bins of size $1/(2^k i)$ are reached. The new item is assigned into one such sub-bin.

Finally, the main algorithm only determines the type of newly arriving items and assigns them to the appropriate algorithms. The total number of active bins is at most $2M-1$. In order to perform a competitive analysis, we prove the following claims.

CLAIM 1. *For a given i $\geq M$, consider an active bin. At all times, the number of empty sub-bins of each size except $1/i$ is at most $2^d - 1$.*

Proof. Note that the number of empty sub-bins of size $1/i$ decays from i^d to zero during the usage of the bin. Consider a certain possible size r of a sub-bin. When a sub-bin of some size r is created, it is due to partition of a larger sub-bin. This means that there were no empty sub-bins of size r before the partition. Afterwards, there are at most $2^d - 1$ of them for each size that has been created during the partitioning (for the smallest size into which the sub-bin is partitioned, 2^d sub-bins created, but one is immediately used). □

CLAIM 2. *For a given $i \geq M$, when a bin is about to be closed, the total volume of empty sub-bins in the bin is at most $1/i^d$.*

Note that the above claims bound the volume of sub-bins that are not used at all. There is some waste of volume also due to the fact that each item does not fill its sub-bin totally. We compute this waste later.

Proof. For $i \geq M$, when a bin is to be closed, there are no empty sub-bins of size $1/i$. There are at most $2^d - 1$ empty sub-bins of each other size by Claim 1. This gives a total unused volume of at most $(2^d - 1)\sum_{k\geq 1}(2^k i)^{-d} = 1/i^d$. $\square$

CLAIM 3. *The occupied volume in each closed bin of type $i \geq M$ is at least $1 - \varepsilon$.*

Proof. A hypercube which was assigned into a sub-bin of size $1/(2^k i)$ always has size of at least $1/(2^k(i+1))$. Therefore the ratio of occupied space and existing space in each used sub-bin is at least $i^d/(i+1)^d$. When a bin is closed, the total volume of used sub-bins is at least $1 - 1/i^d$ by Claim 2. Therefore the occupied volume in the bin is at least $i^d/(i+1)^d(1 - 1/i^d) = (i^d - 1)/(i+1)^d$. We use $i \geq M$ and $M^d \geq M + 1$ to get $(i^d - 1)/(i+1)^d \geq (M^d - 1)/(M+1)^d \geq (\frac{M}{M+1})^{d+1} \geq 1 - \varepsilon$. $\square$

Now we are ready to analyze the performance. We define a weighting function for ALG_ε [26]. Each item with p type $1 \leq i \leq M - 1$ has weight $w_\varepsilon(p) = 1/i^d$. Each item p' of higher type has weight $w_\varepsilon(p') = (s(p))^d/(1-\varepsilon)$ which is the volume of the item divided by $(1 - \varepsilon)$. We begin by showing that this weighting function is valid for our algorithm.

LEMMA 2.1. *For all input sequences σ, $\text{cost}_{\text{ALG}_\varepsilon}(\sigma) \leq \sum_{i=1}^{n} w_\varepsilon(p_i) + O(1)$.*

Proof. Each closed bin of type $1 \leq i \leq M - 1$ contains i^d items. All sub-bins are used when the bin is closed, and thus it contains a total weight of 1. Each closed bin of type $M \leq i \leq 2M - 1$ has occupied volume of at least $1 - \varepsilon$ by Claim 3, and therefore the weights of the items in such a bin sum up to at least 1. A constant number of bins is active (at most $2M - 1$). Thus the total number of bins used by ALG_ε for a given input sequence σ is upper bounded by the total weight of the items plus a constant. $\square$

By this Lemma, for any given $\varepsilon > 0$, the asymptotic performance ratio of our algorithm can be upper bounded by the maximum amount of weight that can be packed in a single bin: for a given input sequence σ (with fixed weight w), the offline algorithm minimizes the number of bins that it needs to pack all items in σ by packing as much weight as possible in each bin. If it needs k bins, the performance ratio on this input is w/k, which is also the average weight per offline bin.

Therefore we need to find the worst case offline bin, i.e. an offline bin which is packed with a maximum amount of weight. However, for the case of cubes, we only have $M + 1$ different types of items. All large items of type i have the same weight. All small items have the same ratio of weight to volume. Therefore the exact contents of a bin are not crucial. In order to define a packed bin, we only need to know how many items there are of each type, and the volume of the small items. To maximize the weight we can assume that the large items are as small as possible (without changing their type), and the rest of the bin is filled with small items.

Formally, we define a *pattern* as a tuple $q = \langle q_1, \ldots, q_{M-1}\rangle$, where there exists a feasible packing into a single bin containing q_i items of type i for all $1 \leq i \leq M-1$. This generalizes the definition from [24]. The weight of a pattern q is at most

$$(2.1)\ \ w_\varepsilon(q) = \sum_{i=1}^{M-1} \frac{q_i}{i^d} + \frac{1}{1-\varepsilon}\left(1 - \sum_{i=1}^{M-1} \frac{q_i}{(i+1)^d}\right).$$

Note that for any given pattern the amounts of items of types $M, \ldots, 2M - 1$ are unspecified. However, as mentioned above, the weight of such items is always their volume divided by $1 - \varepsilon$. Therefore (2.1) gives an upper bound for the total weight that can be packed in a single bin for a given pattern q. Summarizing, we have the following Theorem.

THEOREM 2.1. *The asymptotic performance ratio of ALG_ε is upper bounded by $\max_q w_\varepsilon(q)$, where the maximum is taken over all patterns q that are valid for ALG_ε.*

In order to use the Theorem, we need the following geometric Claim. We immediately formulate it in a general way so that we can also apply it in the next section.

CLAIM 4. *Given a packing of hyperboxes into bins, such that each component j is bounded in an interval $(1/(k_j + 1), 1/k_j]$, where $k_j \geq 1$ is an integer, then each bin has at most $\prod_{j=1}^{d} k_j$ hyperboxes packed in it.*

Proof. We prove the claim by induction on the dimension. Clearly for $d = 1$ the claim holds. To prove the claim for $d > 1$, the induction hypothesis means that a hyperplane of dimension $d - 1$ through the bin which is parallel to one of the sides (the side which is the projection of the bin on the first $d - 1$ dimensions) can meet at most $\prod_{j=1}^{d-1} k_j$ hyperboxes. Next, take the projection of the hyperboxes and the bin on the last axis. We get short intervals of length in $(1/(k_d + 1), 1/k_d]$ (projections of hypercubes) on an main interval of length 1 (the projection of the bin). As mentioned above, each point of the main interval can have the projection of at most $\prod_{j=1}^{d-1} k_j$ items. Consider the short intervals as an interval graph. The size of the largest clique is at most $\prod_{j=1}^{d-1} k_j$. Therefore, as interval graphs are perfect, we can colour the short intervals using $\prod_{j=1}^{d-1} k_j$ colours. Note that the number of intervals of each independent set is at most k_d (due to length), and so the total number of intervals is at most $\prod_{j=1}^{d} k_j$. $\square$

LEMMA 2.2. *Let $\alpha = \liminf_{\varepsilon\to 0} \max_q w_\varepsilon(q)$, where the maximum is taken over all patterns q that are valid for ALG_ε. Then the asymptotic performance ratio of any bounded space algorithm is at least α.*

Proof. We show that there is no bounded space algorithm with an asymptotic performance ratio strictly below α. For any $\varepsilon' > 0$, there exists an $\varepsilon \in (0, \varepsilon')$ such that $\mathcal{R}^\infty(\text{ALG}_\varepsilon) \le (1+\varepsilon')\alpha$. Consider the pattern q for which $w_\varepsilon(q)$ is maximal. We write $w_\varepsilon(q) = (1+\varepsilon'')\alpha$ for some $\varepsilon'' \in [0, \varepsilon']$.

Note that a pattern does not specify the precise sizes of any of the items in it. Based on q, we define a set of hypercubes that can be packed together in a single bin. For each item of type i in q, we take a hypercube of size $1/(i+1)+\delta$ for some small $\delta > 0$. Take $V_\delta = 1 - \sum_{i=1}^{M-1} q_i(1/(i+1)+\delta)^d$. We add a large amount of small hypercubes of total volume V_δ, where the sizes of the small hypercubes are chosen in such a way that they can all be packed in a single bin together with the large hypercubes prescribed by q. By the definition of a pattern, such a packing is feasible for δ sufficiently small.

Define the following input for a bounded space algorithm. Let N be a large constant. The sequence contains M phases. The last phase contains a volume NV_δ of small hypercubes. Phase i ($1 \le i \le M-1$) contains Nq_i hypercubes of size $1/(i+1)+\delta$. After phase i, almost all hypercubes of this phase must be packed into closed bins (except a constant number of active bins). Each such bin may contain up to i^d items, which implies that in each phase i, $Nn_i/i^d - O(1)$ bins are closed. The last phase contributes at least $V_\delta - O(1)$ extra bins. The cost of the online algorithm is $\sum_{i=1}^{M-1} Nq_i/i^d + V_\delta - O(M)$. But the optimal offline cost is simply N. Taking $\delta = 1/N$ and letting N grow without bound, N becomes much larger than M and the asymptotic performance ratio of any bounded space on-line algorithm is lower bounded by $\sum_{i=1}^{M-1} q_i/i^d + V_0$. Note that the weight of this set of hypercubes according to our definition of weights tends to $\sum_{i=1}^{M-1} q_i/i^d + V_0/(1-\varepsilon) = w_\varepsilon(q) = (1+\varepsilon'')\alpha$ as $\delta \to 0$. Therefore $\sum_{i=1}^{M-1} q_i/i^d + V_0 \ge (1-\varepsilon)(1+\varepsilon'')\alpha \ge (1-\varepsilon')\alpha$. $\square$

This Lemma implies that our algorithm is the best possible bounded space algorithm. More precisely, for every $\varepsilon' > 0$, there exists an $\varepsilon \in (0, \varepsilon')$ such that $\mathcal{R}^\infty(\text{ALG}_\varepsilon) \le (1+\varepsilon')\alpha$, and no bounded space algorithm has an asymptotic performance ratio below $(1-\varepsilon')\alpha$. This also implies that our weighting function cannot be improved and determines the asymptotic performance ratio exactly. We show that the asymptotic performance ratio of our algorithm as a function of the dimension is $O(d/\log d)$ when we choose M correctly.

LEMMA 2.3. *By choosing M correctly, ALG_ε has an asymptotic performance ratio of $O(d/\log d)$.*

Proof. Take $M = 2d/\log d$. The occupied area in bins of small types is at least $(\frac{M}{M+1})^{d+1}$ by the proof of Claim 3. This is greater than $(\frac{M+1}{M})^{-d} = (1+1/M)^{-d} = (1+(\log d)/(2d))^{-d}$, which tends to $e^{-(\log d)/2} = (e^{\log d})^{-1/2} = 1/\sqrt{d}$ for $d \to \infty$.

Suppose the input is I. Denote by I_i the subsequence of items of type i ($i = 1, \ldots, M$), where we consider all the small types as a single type. Then we have $\text{ALG}(I_i) = \text{OPT}(I_i) \le \text{OPT}(I)$ for $i = 1, \ldots, M-1$, since if items of only one type arrive, our algorithm packs them perfectly. Moreover, $\text{ALG}(I_M) = O(\sqrt{d})\cdot\text{OPT}(I_M) = O(\sqrt{d})\cdot\text{OPT}(I)$ for $i = M$. Thus $\text{ALG}(I) = \sum_{i=1}^{M} \text{ALG}(I_i) \le (M-1)\text{OPT}(I) + O(\sqrt{d})\text{OPT}(I) = O(d/\log d)\text{OPT}(I)$. $\square$

We can show that the asymptotic performance ratio of this problem is $\Omega(\log d)$ by considering the following lower bound construction. We use $\lceil \log d \rceil$ phases. In phase i, $N((2^i-1)^d - (2^i-2)^d)$ items of size $2^{-i}(1+\varepsilon)$ arrive, where $\varepsilon < 2^{-\lceil \log d \rceil} \le 1/d$. OPT can place all these items in just N bins by using the following packing scheme. Each bin is packed identically, so we just describe the packing of a single bin. The first item is placed in a corner of the bin. We assign coordinates to the bin so that this corner is the origin and all positive axes are along edges of the bin. (The size of the bin in each dimension is 1.)

Consider any coordinate axis. We reserve the space between $(1-2^{1-i})(1+\varepsilon)$ and $(1-2^{-i})(1+\varepsilon)$ for items of phase i. Note that this is exactly the size of such an item. By doing this along every axis, we can place all $(2^i-1)^d - (2^i-2)^d$ items of phase i. (There would be room for $(2^i-1)^d$ items if we used all the space until $(1-2^{-i})(1+\varepsilon)$ along each axis; we lose $(2^i-2)^d$ items because the space until $(1-2^{1-i})(1+\varepsilon)$ is occupied.)

The minimum number of bins that any bounded space online algorithm needs to place the items of phase i is $N((2^i-1)^d - (2^i-2)^d)/(2^i-1)^d = N(1-(\frac{2^i-2}{2^i-1})^d)$ items. Note that the contribution of each phase i to the total number of bins required to pack all items is strictly decreasing in i. Consider the contribution of the last phase, which is phase $\lceil \log d \rceil$. Since $\lceil \log d \rceil \le 1 + \log d$, it is greater than $N(1-(\frac{2d-2}{2d-1})^d) = N(1-(1-\frac{1}{2d-1})^d) \ge N(1-e^{-1/2}) > 0.39N$ for all $d \ge 2$. Thus all $\lceil \log d \rceil$ terms all contribute at least $0.39N$, and the total number of bins required is at least $0.39N(\lceil \log d \rceil)$. This implies a lower bound of $\Omega(\log d)$ on the asymptotic performance ratio of this problem.

Finally, we give an overview of the upper and lower bounds we can show for small dimensions. The upper bounds were found by formulating a linear program and solving it (using Matlab). The lower bounds were found by constructing bins with high weights. We omit the details of these constructions. All results for $d > 1$ improve upon existing results.

d	Lower bound	Upper bound
1	1.69103	1.69103
2	2.35656	2.3722
3	2.94457	3.0672
4	3.36547	3.7595
5	3.74913	4.4052
6	4.04789	5.0126
7	4.27726	5.7084

3 An optimal algorithm for bounded space packing of hyperboxes

Next we describe how to extend the algorithm for hypercubes to handle hyperboxes instead of hypercubes. This algorithm also uses the parameter ε. The value of M as a function of ε is picked so that $M \geq 1/(1-(1-\varepsilon)^{1/(d+2)})-1$. Similarly to the previous algorithm, the hyperboxes are classified into types. A arriving hyperbox b of dimensions $(b_1, b_2, \ldots, b_d)$, is classified as one of $(2M-1)^d$ types depending on its components: a type of a hyperbox is the vector of the types of its components.

There are $2M-1$ types of components. A component larger than $1/M$ has type i if $1/(i+1) < b_i \leq 1/i$, and is called large. A component smaller than $1/M$ has type i, where $M \leq i \leq 2M-1$, if there exists a non-negative integer f_i such that $1/(i+1) < 2^{f_i} b_i \leq 1/i$. Such components are called small.

Each of the $(2M-1)^d$ types is packed separately and independently of the other types. The algorithm keeps one active bin for each type $(s_1, \ldots, s_d)$. When such a bin is opened, it is split into $\prod_{i=1}^{d} s_i$ identical sub-bins of dimensions $(1/s_1, \ldots, 1/s_d)$. On arrival of a hyperbox p, after classification into a type, a sub-bin has to be found for it. If there is no sub-bin in the current bin that is larger than p in every dimension, we close the bin and open a new one. Otherwise, we take an empty sub-bin that has minimum volume among all sub-bins that can contain p.

Now consider the components of p one by one. If the i-th component is large, the sub-bin has the correct size in this dimension: its size is $1/s_i$ whereas the component is in $(1/(s_i-1), 1/s_i]$.

If the i-th component is small, the size of the sub-bin in the i-th dimension may be too large. Suppose its size is $1/(2^{f'} s_i)$ whereas the hyperbox has size $1/(2^f s_i)$ in this dimension for some $f > f'$. In this case, we divide the sub-bin into two equal parts by cutting halfway (across the i-th dimension). If the new sub-bins have the proper size, take one of the two smallest sub-bins that were created, and continue with the next component. Otherwise, take one of the new sub-bins and cut it in half again, repeating until the size of a created sub-bin is $1/(2^f s_i)$.

Thus we ensure that the sub-bin that we use to pack the item p has the proper size in every dimension. We then place this item anywhere inside the sub-bin.

We now generalize the proofs from the previous section for this algorithm.

CLAIM 5. *Consider a type $(s_1, \ldots, s_d)$, and its active bin. For every vector $(f_1, \ldots, f_d) \neq 0$ of nonnegative integers such that $f_i = 0$ for each large component i, there is at most one empty sub-bin of size $(1/(2^{f_1} s_1), \ldots, 1/(2^{f_d} s_d))$.*

Proof. Note that the number of sub-bins of size $(1/s_1, \ldots, 1/s_d)$, is initialized to be $\prod_{i=1}^{d} s_i$, and decays until it reaches the value zero. The cutting process does not create more than a single empty sub-bin of each size. This is true for all the sub-bins created except for the smallest size that is created in any given process. For that size we create two identical sub-bins. However, one of them is filled right away.

Furthermore, no sub-bins of existing sizes are created due to the choice of the initial sub-bin. The initial sub-bin is chosen to be of minimum volume among the ones that can contain the item, and hence all the created sub-bins (all of which can contain the item) are of smaller volume than any other existing sub-bin that can contain the item. $\square$

CLAIM 6. *The occupied volume in each closed bin of type $(s_1, \ldots, s_d)$ is at least*

$$(1-\varepsilon) \prod_{i \in L} s_i/(s_i+1),$$

where L is the set of large components in this type.

Proof. To bound the occupied volume in closed bins, note that a sub-bin which was assigned an item is full by a fraction of at least

$$\prod_{i=1}^{d} \frac{s_i}{s_i+1} \geq \left(\frac{M}{M+1}\right)^{d-|L|} \prod_{i \in L} \frac{s_i}{s_i+1}.$$

Considering sub-bins that were empty when the bin was closed, by Claim 5 there may be one empty sub-bin of each size $(1/(2^{f_1} s_1), \ldots, (1/(2^{f_d} s_d))$, with the restrictions that f_i is a nonnegative integer for $i = 1, \ldots, d$, $f_i = 0$ for each large component i, and there exists some $i \in \{1, \ldots, d\}$ such that $f_i \neq 0$.

If there are no small components, there can be no empty sub-bins because large components never cause splits into sub-bins, so all sub-bins are used when the bin is closed. This gives a bound of $\prod_{i \in L} s_i/(s_i+1)$.

If there is only one small component, the total volume of all empty sub-bins that can exist is $1/(s_1 \ldots s_d) \cdot (\frac{1}{2} + \frac{1}{4} + \ldots) \leq 1/(s_1 \ldots s_d) \leq 1/M$, since one of the components is small (type is at least M) and all other components have type at least 1. The occupied volume is at least $(1-1/M)\cdot\frac{M}{M+1}\prod_{i\in L}(s_i/(s_i+1)) \geq (\frac{M}{M+1})^{d+2}\prod_{i\in L}(s_i/(s_i+1))$. This holds for any $d\geq 2$ and $M\geq 2$.

If there are $r\geq 2$ small components, the total volume of empty sub-bins is at most $(2^r-1)/(s_1s_2\dots s_d)\leq(2^r-1)/M^r\leq 2^r/M^r$. (We get the factor 2^r-1 by enumerating over all possible choices of the values f_i.) We get that each bin is full by at least a fraction of

$$\begin{aligned}&\left(1-\frac{2^r}{M^r}\right)\left(\frac{M}{M+1}\right)^r\prod_{i\in L}\frac{s_i}{s_i+1}\\ =\ &\frac{M^r-2^r}{(M+1)^r}\prod_{i\in L}\frac{s_i}{s_i+1}\geq\left(\frac{M}{M+1}\right)^{r+2}\prod_{i\in L}\frac{s_i}{s_i+1}\\ \geq\ &\left(\frac{M}{M+1}\right)^{d+2}\prod_{i\in L}\frac{s_i}{s_i+1}.\end{aligned}$$

The penultimate inequality holds for $M^r-2^r\geq M^{r+2}/(M+1)^2$, which holds for any $r\geq 2$ and $M\geq 4$.

Using $(\frac{M}{M+1})^{d+2}\geq 1-\varepsilon$ we get the Claim. □

We now define a weighting function for our algorithm. The weight of a hyperbox p with components $(b_1,\dots,b_d)$ and type $(s_1,\dots,s_d)$ is defined as

$$w_\varepsilon(p)=\frac{1}{1-\varepsilon}\prod_{i\notin L}b_i\prod_{i\in L}\frac{1}{s_i},$$

where L is the set of large components in this type.

LEMMA 3.1. *For all input sequences σ, $\mathrm{cost}_{\mathrm{alg}}(\sigma)\leq\sum_{i=1}^n w_\varepsilon(p_i)+O(1)$.*

Proof. In order to prove the claim, it is sufficient to show that each closed bin contains items of total weight of at least 1. Consider a bin filled with hyperboxes with type $(s_1,\dots,s_d)$. It is sufficient to consider the subsequence σ of the input that contains only items of this type, since all types are packed independently. We build an input σ' for which both the behavior of the algorithm and the weights are the same as for σ, and show the claim holds for σ'. Let $\delta<1/M^3$ be a very small constant.

For a hyperbox $p\in\sigma$ with components $(b_1,\dots,b_d)$ and type $(s_1,\dots,s_d)$, let $p'=(b'_1,\dots,b'_d)\in\sigma'$ be defined as follows. For $i\notin L$, $b'_i=b_i$. For $i\in L$, $b'_i=1/(s_i+1)+\delta<1/s_i$. As p and p' have the same type, they require a sub-bin of the same size in all dimensions. Therefore the algorithm packs σ' in the same way as it packs σ. Moreover, according to the definition of weight above, p and p' have the same weight.

Let $v(p)$ denote the volume of an item p. For $p\in\sigma$, we compute the ratio of weight and volume of the item p'. We have

$$\begin{aligned}\frac{w_\varepsilon(p')}{v(p')}&=\frac{1}{1-\varepsilon}\prod_{i\notin L}b'_i\prod_{i\in L}\frac{1}{s_i}\Big/\prod_{i=1}^d b'_i\\ &=\frac{1}{1-\varepsilon}\prod_{i\in L}\frac{1}{s_ib_i}>\frac{1}{1-\varepsilon}\prod_{i\in L}\frac{s_i+1}{s_i+M^2\delta}.\end{aligned}$$

As δ tends to zero, this bound approaches the inverse of the number in Claim 6. This means that the total weight of items in a closed bin is no smaller than 1. □

Just like in Section 2, this Lemma implies that the asymptotic worst case ratio is upper bounded by the maximum amount of weight that can be packed in a single bin. We now prove a technical Lemma that implies that this weighting function is also "optimal" in that it determines the true asymptotic performance ratio of our algorithm.

DEFINITION 1. *The* pseudo-volume *of a hyperbox $H=(b_1,\dots,b_d)$ is defined as $\prod_{i\notin L}b_i$, where L is the set of large components of H.*

Suppose that for a given set of hyperboxes X, we can partition the dimensions into two sets, A and B, such that for each dimension i in A, we have that the i-th components of all hyperboxes in X are bounded in an interval $(1/(k_j+1),1/k_j]$. There are no restrictions on the dimensions in B. (Thus such a partition can always be found by taking $A=\emptyset$.)

For a hyperbox $H\in X$, define the *generalized pseudo-volume* of the components in B by $v(H,B)=\prod_{i\in B}b_i$, where b_i is the i-th component of H. Define the total generalized pseudo-volume of all hyperboxes in a set X by $v(X,B)=\sum_{H\in X}v(H,B)$.

CLAIM 7. *For a given set X of hyperboxes, for sufficiently large N, any packing of X into bins requires at least $v(X,B)(1-\frac{1}{N})^{|B|}/\prod_{i\in A}k_i$ bins, where A and B form a partitioning of the dimensions as described above.*

Proof. We prove the claim by induction on the number of dimensions in B. For $|B|=0$, we find that the total generalized pseudo-volume of X is simply the number of hyperboxes in X (since the empty product is 1) and thus the claim is true using Claim 4.

Assume the claim is true for $|B|=0,\dots,r-1$. Suppose $|A|=d-r<d$. Take any dimension $i\in B$. We replace each hyperbox H, with component b_i in dimension i, by $\lfloor Nb_i\rfloor$ hyperboxes that have $\frac{1}{N}$ as their i-th component, and are identical to H in all other components. Here N is taken sufficiently large, such that $\frac{1}{N}<b_i$. Clearly, the new input X' is no harder to pack, as we split each item into parts whose sum is smaller than or equal to the original items. The total generalized pseudo-volume of the hypercubes in X' is at most a factor of $1-\frac{1}{N}$ smaller than that of X.

So if we write $B' = B \backslash \{i\}$, we have $v(X', B') \cdot \frac{1}{N} \geq v(X, B)(1 - \frac{1}{N})$. By induction, it takes at least

$$v(X', B') \cdot (1 - \frac{1}{N})^{r-1} / \prod_{j \in A \cup \{i\}} k_j$$

bins to pack the modified input X'. Using that $k_i = N$, this is $v(X, B) \cdot (1 - \frac{1}{N})^r / \prod_{j \in A} k_j$ bins. □

Letting $\gamma = 1 - (1 - \frac{1}{N})^d$, we get that the required number is at least $v(X, B)(1 - \gamma) / \prod_{j \in A} k_j$ bins, where $\gamma \to 0$ as $N \to \infty$. In the remainder, we will take A to be the dimensions where the components of the hyperboxes in X are large, and B the dimensions where they are small. Note that this choice of A satisfies the constraints on A above, and that this reduces the generalized pseudo-volume to the (normal) pseudo-volume defined before. We are ready to prove the following Lemma.

LEMMA 3.2. *Let $\varepsilon > 0$. Suppose the maximum amount of weight that can be packed in a single bin is α_ε. Then our algorithm has an asymptotic performance ratio of α_ε, and the asymptotic performance ratio of any bounded space algorithm is at least $(1 - \varepsilon)\alpha_\varepsilon$.*

Proof. The first statement follows from Lemma 3.1. We show a lower bound of value which tends to α_ε on the asymptotic performance ratio of any bounded space algorithm.

Consider a packed bin for which the sum of weights is α_ε. Partition the hyperboxes of this bin into M^d types in the following way. Each component is either of a type in $\{1, \ldots, M-1\}$ or small (i.e. of a type i, $i \leq M$). Let N' be a large constant. The sequence consists of phases. Each phase consists of one item from the packed bin, repeated N' times. The optimal offline cost is therefore N'. Using Claim 7 we see that the amount of bins needed to pack a phase which consists of an item p repeated N' times is simply $N' w(p)(1 - \gamma)(1 - \varepsilon)$. Therefore the cost of an on-line algorithm is at least $N' \alpha_\varepsilon (1 - \gamma)(1 - \varepsilon) - O(1)$, which makes the asymptotic performance ratio arbitrarily close to $(1 - \varepsilon)\alpha_\varepsilon$. □.

Furthermore, we can determine the asymptotic performance ratio of our algorithm for hyperbox packing. Comparing to the unbounded space algorithm from [10] we can see that all the weights we defined are smaller than or equal to the weights used in [10]. So the asymptotic performance ratio is not higher. However, it can also not be lower due to the general lower bound for bounded space algorithms. This means that both algorithms have the same asymptotic performance ratio, namely $(\Pi_\infty)^d$, where $\Pi_\infty \approx 1.691$ is the asymptotic performance ratio of the algorithm HARMONIC [20].

4 An unbounded space algorithm for square packing

We define a two-dimensional version of Modified Harmonic that we call MH^2. It uses seven item types (intervals in $(0, 1]$), denoted by $1, 1a, 2, 2a, 3, 4, 5$ in order of decreasing size. The algorithm uses a variable $\Delta \in (1/3, 0.385)$. The upper bound for type i ($i = 1, \ldots, 5$) is $1/i$. The upper bound for type $1a$ is $1 - \Delta$, and for $2a$ it is Δ.

MH^2 packs all items of size at most $1/5$ (type 5) using the algorithm from Section 2 for small items. That is, we divide the items into 5 subtypes, and for each item of subtype i we call the function ASSIGNLARGE(i) (we take $M = 5$).

Items of type $2a$ are partially put three to a bin, in such a way that a type $1a$ item could be put in the same bin with them, and partially put four to a bin, in the four corners. These items are colored red and blue, respectively. We use a second parameter $\alpha \in (1/5, 1/4)$ to denote the fraction of type $2a$ items that are colored red. The values of α and Δ will be chosen in such a way that the asymptotic performance ratio of MH^2 is minimized. We will find $\alpha \approx 0.2094$ and $\Delta \approx 0.3803$.

Items of types $1, 2, 3,$ and 4 are packed in bins that only contain items of one type. Full bins contain $1, 4, 9,$ and 16 items, respectively. By the proof of Claim 3, full bins containing items of type 5 have occupied area of at least $(M^d - 1)/(M + 1)^d = 24/36 = 2/3$.

We give an overview of the types, weights and expansions in Table 1.

Table 1: Item weights and expansions for MH^2

Type	Max. size	W_1	W_2	E_1	E_2
1	1	1	1	$\frac{1}{(1-\Delta)^2}$	$\frac{1}{(1-\Delta)^2}$
$1a$	$1-\Delta$	1	0	4	0
2	$1/2$	$1/4$	$1/4$	$1/(4\Delta^2)$	$1/(4\Delta^2)$
$2a$	Δ	$\frac{1-\alpha}{4}$	$\frac{3+\alpha}{12}$	$\frac{9-9\alpha}{4}$	$\frac{9+3\alpha}{4}$
3	$1/3$	$1/9$	$1/9$	$16/9$	$16/9$
4	$1/4$	$1/16$	$1/16$	$25/16$	$25/16$
5	$1/5$	$\frac{3}{2}x$	$\frac{3}{2}x$	$3/2$	$3/2$

We omit the proof of the following Theorem.

THEOREM 4.1. *Denote by w_i the maximum amount of weight that can be packed into a single bin according to measure W_i ($i = 1, 2$). Then the asymptotic performance ratio of MH^2 is upper bounded by $\max(w_1, w_2)$.*

The expansion of type 5 items (small items) is $3/2$. To simplify the calculations, we will assume that it is instead $25/16$. This can only make the asymptotic performance ratio worse, so we will calculate an upper bound for the asymptotic performance ratio. Using this value enables us to ignore the items of type 4, since they also have expansion $25/16$: we will assume that after taking some items from types $1, 1a, 2, 2a$, and 3 we can fill up the rest of the bin entirely with items of expansion $25/16$ (this is the worst case).

To find a bin with maximal weight, we need to try all possible combinations of items of types $1, 1a, 2, 2a$, and 3 and fill up the rest with "sand" (small items). The weight is maximized by minimizing the size of all the large items that we use (since the weight they contribute does not depend on their size, and it maximizes the area available for other large items and for small items). We take $\alpha = \frac{25}{36}(9\Delta^2 - 1)$. This choice of α ensures that if we have a bin with a type 2 item (of minimal size), and we replace it with a type $2a$ item (of minimal size) and fill the remaining area with sand, the total weight is unchanged: we have $1/4 = \frac{1-\alpha}{4} + \frac{25}{16}(\Delta^2 - 1/9)$.

We use the following Lemma from [25]. This Lemma can also be proved using Claim 4.

LEMMA 4.1. *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 be packed with it.*

THEOREM 4.2. MH^2 *maintains a asymptotic performance ratio of at most 2.270876.*

Proof. We look for a bin with maximal weight.

Case 1. First of all, suppose that there is no item of type 1 or $1a$ in the bin. For the remaining items, $W_1(p) \le W_2(p)$ for all items p. Thus we only need to consider W_2. The type with the highest expansion is type $2a$, which fits at most four times in a bin. All other types have expansion at most $16/9$ by our choice of Δ, so the total weight of the bin in this case is at most $(3+\alpha)/3 + (1-4/9)16/9 < 2.1$.

Suppose there is an item of type 1 or 1a. By Lemma 4, at most 5 items of type $2, 2a$, or 3 can be packed with them. As usual we assume that the rest of the bin is filled with items of expansion $25/16$. Since the items of type $2, 2a$ and 3 have expansion greater than $25/16$, the weight is maximized by taking 5 of them.

Case 2. Suppose there is an item of type 1. Then there is no item of type $1a$; for all other items, $W_2(p) \ge W_1(p)$, so we only need to consider $W_2(p)$. In this case items of type 2 do not fit in the bin. If we place i items of type $2a$ in the bin, this leaves room for at most $5-i$ items of type 3. We assume that $5-i$ such items can always be placed. (This is a worst-case assumption.) Since the weight of type $2a$ items is higher than that of type 3 items, and the total number of type $2a$ items and type 3 items is assumed to be independent of the number of type $2a$ items, we maximize the number of type $2a$ items. However, at most 4 items of size strictly greater than $1/3$ fit in one bin, so at most 3 items of type $2a$ can be placed in this bin together with the item of type 1. This leaves two items of type 3.

The total weight is at most $1+3/4+\alpha/4+2/9+\frac{25}{16}(1-(1-\Delta)^2-1/3-1/8)$.

Case 3. Now suppose there is an item of type $1a$. Then the weight of the bin is maximal if we use W_1 (using W_2, the total weight is now lower than in Case 1).

There are at most 5 items of type $2, 2a$, or 3. By the remarks above Lemma 4, we do not need to distinguish between bins that only differ in that one of them has a type 2 item where the other has a type 2a item and more sand. Since type 2 and 2a items have higher weight than items of type 3 (because $\Delta < 0.385$), we take 3 of them, and 2 of type 3 (here we ignore that items of type 3 do not necessarily fit in a bin that already contains one item of size greater than $1/2$ and three items of size greater than Δ, just like in Case 2).

The total weight is at most $1+3/4+2/9+\frac{25}{16}(1-1/4-3\Delta^2-2/16)$.

We take Δ to be the solution of $27\Delta^2+18\Delta-43/4=0$ that is in the interval $(1/3, 0.385)$. Then the weights in Cases 2 and 3 above are the same, and they are 2.270876. This is an upper bound for the asymptotic performance ratio of MH^2. $\square$

5 A $16/11$ approximation for offline square packing

We present an approximation algorithm for the packing of squares in two-dimensional bins.

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.

We pack small items only into bins that contain a single large item of size at least $1/2$, and into bins that contain exactly four items (in such bins, the smallest two items have total size at least $2/3$). In this way, we ensure that such bins (if any exist) contain an amount of items of total area nearly 1, i.e. little space is wasted (unused) in such bins. The key to bounding the approximation ratio is to show that the remaining bins cannot hold too many small items in the optimal offline solution.

Let A be a set of item sizes. 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 $1 \le j \le m$, define P_j to be the number of items of size a_j in P. 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 $1 - \sum_i P_i a_i^2$.

The following Lemma extends the corresponding Lemma from [25]. It is key to proving the approximation ratio of $16/11$. The proof is omitted.

LEMMA 5.1. *The waste of any dominant pattern which contains at least five items is at most* $5/11 \approx 0.4545$.

6 Conclusions

An open question left by this paper is what the asymptotic performance ratio of the bounded space hypercube packing problem is. We can show that it is $\Omega(\log d)$, and we conjecture that it is $\Theta(\log d)$.

It should be possible to improve upon the algorithm in section 4. However, at the moment we lack a good way of enumerating all (dominant) patterns in a two-dimensional bin if there are many item types. This is an interesting open problem.

References

[1] Nikhil Bansal and Maxim Sviridenko. New approximability and inapproximability results for 2-dimensional packing. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms*. ACM/SIAM, 2004. To appear.

[2] David Blitz, André van Vliet, and Gerhard J. Woeginger. Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms. Unpublished manuscript, 1996.

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

[4] Alberto Caprara. Packing 2-dimensional bins in harmony. In *Proc. 43th IEEE Symp. on Found. of Comp. Science*, pages 490–499, 2002.

[5] Fan R. K. Chung, Michael R. Garey, and David S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66–76, 1982.

[6] 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.

[7] Don Coppersmith and Prabhakar Raghavan. Multidimensional online bin packing: Algorithms and worst case analysis. *Operations Research Letters*, 8:17–20, 1989.

[8] Jose Correa and Claire Kenyon. Approximation schemes for multidimensional packing. In *Proceedings of the 15th ACM/SIAM Symposium on Discrete Algorithms*. ACM/SIAM, 2004. To appear.

[9] János Csirik, Johannes B. G. Frenk, and Martine Labbe. Two dimensional rectangle packing: on line methods and results. *Discrete Applied Mathematics*, 45:197–204, 1993.

[10] János Csirik and André van Vliet. An on-line algorithm for multidimensional bin packing. *Operations Research Letters*, 13(3):149–158, Apr 1993.

[11] 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.

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

[13] Carlos E. Ferreira, Flavio K. Miyazawa, and Yoshiko Wakabayashi. Packing squares into squares. *Pesquisa Operacional*, 19(2):223–237, 1999.

[14] Gabor Galambos. A 1.6 lower bound for the two-dimensional online rectangle bin packing. *Acta Cybernetica*, 10:21–24, 1991.

[15] Gabor Galambos and André van Vliet. Lower bounds for 1-, 2-, and 3-dimensional online bin packing algorithms. *Computing*, 52:281–297, 1994.

[16] 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.

[17] David S. Johnson. Fast algorithms for bin packing. *Journal Computer Systems Science*, 8:272–314, 1974.

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

[19] Yoshiharu Kohayakawa, Flavio K. Miyazawa, Prabhakar Raghavan, and Yoshiko Wakabayashi. Multidimensional cube packing. In Jayme Szwarcfiter and Siang Song, editors, *Electronic Notes in Discrete Mathematics*, volume 7. Elsevier Science Publishers, 2001.

[20] C. C. Lee and D. T. Lee. A simple online bin packing algorithm. *Journal of the ACM*, 32:562–572, 1985.

[21] K. Li and K. H. Cheng. A generalized harmonic algorithm for on-line multi-dimensional bin packing. manuscript, 1990.

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

[23] Flavio K. Miyazawa and Yoshiko Wakabayashi. Cube packing. *Theoretical Computer Science*, 297(1-3):355–366, 2003.

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

[25] Steve S. Seiden and Rob van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3):261–293, 2003.

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

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

[28] André van Vliet. *Lower and upper bounds for online bin packing and scheduling heuristics*. PhD thesis, Erasmus University, Rotterdam, The Netherlands, 1995.

[29] André van Vliet. On the asymptotic worst case behaviour of harmonic fit. *Journal of Algorithms*, 20:113–136, 1996.

[30] Gerhard J. Woeginger. Improved space for bounded-space online bin packing. *SIAM Journal on Discrete Mathematics*, 6:575–581, 1993.

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