Optimal Online Algorithms for Multidimensional Packing Problems*

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.

Furthermore, we extend the techniques used in these algorithms to give optimal algorithms for online bounded space variable-sized packing and resource augmented packing.

1 Introduction

Bin packing is one of the oldest and most well-studied problems in computer science [12, 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* $h_1, h_2, \ldots, h_n$. Each item h has a fixed *size*, which is $s_1(h) \times$

*Preliminary versions of different parts of this paper appeared in Proc. Symp. Discr. Alg. 2004 (SODA 2004) and Proc. Eur. Symp. Alg. 2004 (ESA 2004).

†Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il. Research supported by Israel Science Foundation (grant no. 250/01).

‡Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany. vanstee@ira.uka.de. Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-061-000. Work performed while this author was at CWI, The Netherlands.

$\cdots \times s_d(h)$. I.e., $s_i(h)$ is the size of h 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 $(x_1(h), \ldots, x_d(h))$, where $0 \leq x_i(h)$ and $x_i(h) + s_i(h) \leq 1$ 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.
- In *variable-sized* bin packing, bins of various sizes are available to be used for packing and the goal is to minimize the total size of all the bins used.
- In *resource-augmented* bin packing, the online algorithm has larger bins at its disposal than the offline algorithm, and the goal is to minimize the number of bins used.

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 consider online 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 or bounded space 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 $\mathcal{R}_{\mathcal{O}}^{\infty}$.

Previous Results: The classic (one-dimensional) online bin packing problem was first investigated by Ullman [33]. He showed that the First Fit algorithm has performance ratio $\frac{17}{10}$. This result was then published in [20]. Johnson [22] 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 $\frac{3}{2}$ [38]. Brown and Liang independently improved this lower bound to 1.53635 [3, 28]. The lower bound currently stands at 1.54014, due to van Vliet [34]. 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 $(1 + \varepsilon)\Pi_\infty$ [25]. 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 [37], and by Van Vliet in [36]. Currently the best known unbounded space upper bound is 1.58889 due to Seiden [31].

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 [20] were the first to study the approximation ratios of both online and offline algorithms. Fernandez de La Vega and Lueker [15] presented the first (asymptotic) approximation scheme for bin packing. Karmarkar and Karp [23] gave an algorithm which uses at most $\text{cost}(\sigma) + \log^2(\text{cost}(\sigma))$ bins.

The on-line one-dimensional variable-sized bin packing problem was first investigated by Friesen and Langston [17]. Csirik [9] proposed the Variable Harmonic algorithm and showed that it has performance ratio at most Π_∞. Seiden [30] showed that this algorithm is optimal among bounded space algorithms.

The on-line one-dimensional resource augmented bin packing problem was studied by Csirik and Woeginger [13]. They showed that the optimal bounded space asymptotic performance ratio is a function $\rho(b)$ of the size b of the bins of the online algorithm.

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_∞ for $d = 2$.

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 [10] gave an algorithm based on First Fit with performance ratio $\frac{49}{16} = 3.0625$ for $d = 2$. Csirik and van Vliet [11] presented 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 [27] also gave a HARMONIC-based algorithm for $d = 2$ and $d = 3$.

Seiden and van Stee [32] improved the upper bound for $d = 2$ to 2.66013. Several lower bounds have been shown [18, 19, 35, 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 [11].

For online square packing, even less is known. The following results are known for $d = 2$: Coppersmith and Raghavan [7] showed an upper bound of $43/16 = 2.6875$ and a lower bound of $4/3$ (which holds for all $d \geq 2$). The upper bound was improved to $395/162 < 2.43828$ by Seiden and van Stee [32]. For $d = 3$, Miyazawa and Wakabayashi [29] showed an upper bound of 3.954. For the offline problem, Ferreira, Miyazawa and Wakabayashi give a 1.988-approximation algorithm [16]. A sequence of results improved this result [24, 32, 4], recently culminating in an APTAS by Bansal and Sviridenko [1] and Correa and Kenyon [8] independently.

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 an 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 bounded space algorithm for the variable-sized multidimensional bin packing problem. As for the first algorithm above, we do not know the exact asymptotic performance ratio.
- We then give an analogous algorithm for the problem of resource augmented online bin packing. This algorithm is also optimal, with an asymptotic performance ratio of $\prod_{i=1}^{d} \rho(b_i)$ where $b_1 \times \cdots \times b_d$ is the size of the bins that the online algorithm uses.

For the online results, we will use the technique of weighting functions. This technique was originally introduced for one-dimensional bin packing algorithms [33, 21]. In [32], 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 [11] 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].

1.1 The Harmonic algorithm

In this section we briefly discuss the important one-dimensional HARMONIC algorithm [25]. In the next sections we adapt it to the multi dimensional cases using our novel techniques.

The fundamental idea of these algorithms is to first classify items by size, and then pack an item according to its class (as opposed to letting the exact size influence packing decisions).

For the classification of items, we need to partition the interval $(0, 1]$ into subintervals. The standard HARMONIC algorithm uses $M-1$ subintervals of the form $(1/(i+1), 1/i]$ for $i = 1, \ldots, M-1$ and one final subinterval $(0, 1/M]$. Each bin will contain only items from one subinterval (type). Items in subinterval i are packed i per bin for $i = 1, \ldots, M-1$ and the items in interval M are packed in bins using NEXT FIT(i.e. a greedy algorithm that opens a new active bin whenever an item does not fit into the current active bin, and never uses the previous bins).

2 Packing 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 h, $s(h)$ 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 [25]. 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 h of size $s(h) \leq 1/M$. Let k be the largest non-negative integer such that $2^k s(h) \leq 1/M$. Clearly $2^k s(h) > 1/(2M)$. Let i be the integer such that $2^k s(h) \in (1/(i+1), 1/i]$, $i \in \{M, \dots 2M-1\}$. 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 $i = M, \dots, 2M-1$. A sub-bin which received a hypercube 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 h of type i arrives we do the following. Let k be the integer such that $2^k s(h) \in (1/(i+1), 1/i]$.

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 maximum $j < k$. 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 of type i. At all times, the number of empty sub-bins in it 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 such a bin. Consider a certain possible size r of a sub-bin in it. 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 of type i 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 of type i is to be closed, there are no empty sub-bins of size $1/i$ in it. 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 $(2^d - 1)\sum_{k\geq 1}(2^k i)^{-d} = 1/i^d$. □

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 $(i^d-1)/(i+1)^d \geq (M^d-1)/(M+1)^d \geq (\frac{M}{M+1})^{d+1} \geq 1-\varepsilon$. □

Now we are ready to analyze the performance. We define a weighting function for ALG_ε [33]. Each item p with 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, $\text{cost}_{\text{ALG}_\varepsilon}(\sigma) \leq \sum_{h\in\sigma} w_\varepsilon(h) + 2M - 1$.*

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. At most $2M-1$ bins are active. 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 $2M-1$. □

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 [31]. The weight of a pattern q is at most

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

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 (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 component* j *of each hyperbox is bounded in an interval* $(1/(k_j+1), 1/k_j]$*, where* $k_j \geq 1$ *is an integer for* $j = 1, \ldots, d$*, 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 color the short intervals using $\prod_{j=1}^{d-1} k_j$ colors. 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 $\prod_{j=1}^{d} k_j$. □

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 $\mathcal{R}^\infty(\text{ALG}_\varepsilon) \leq (1+\varepsilon')\alpha$. 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 ($1 \leq i \leq M - 1$) 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, $Nq_i/i^d - O(1)$ 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 $\sum_{i=1}^{M-1} q_i/i^d + V_0 \geq (1 - \varepsilon)(1 + \varepsilon'')\alpha \geq (1 - \varepsilon')\alpha$. □

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 $\mathcal{R}^\infty(\text{ALG}_\varepsilon) \leq (1 + \varepsilon')\alpha$, 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. However, we have no general formula for this ratio. We do have the following bounds.

Theorem 2.2 *There exists a value of M such that the asymptotic performance ratio of* ALG_ε *is $O(d/\log d)$. Any bounded space algorithm (in particular* ALG_ε*) has an asymptotic performance*

ratio of $\Omega(\log d)$.

Proof. We first show the upper bound. 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 $\textsc{alg}(I_i) = \textsc{opt}(I_i) \leq \textsc{opt}(I)$ for $i = 1, \ldots, M-1$, since if items of only one type arrive, our algorithm packs them perfectly. Moreover, $\textsc{alg}(I_M) = O(\sqrt{d}) \cdot \textsc{opt}(I_M) = O(\sqrt{d}) \cdot \textsc{opt}(I)$ for $i = M$. Thus $\textsc{alg}(I) = \sum_{i=1}^{M} \textsc{alg}(I_i) \leq (M-1)\textsc{opt}(I) + O(\sqrt{d})\textsc{opt}(I) = O(d/\log d)\textsc{opt}(I)$.

We now prove the lower bound. Consider the following lower bound construction. (This lower bound can also be shown using the weighting function.) We use $\lceil \log d \rceil$ phases. In phase i, $N((2^i-1)^d - (2^i-2)^d)$ items of size $2^{-i}(1+\delta)$ arrive, where $\delta < 2^{-\lceil \log d \rceil} \leq 1/d$. 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 $(1-2^{1-i})(1+\delta)$ and $(1-2^{-i})(1+\delta)$ 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 $(1-2^{-i})(1+\delta)$ along each axis; we lose $(2^i-2)^d$ items because the space until $(1-2^{1-i})(1+\delta)$ is occupied.)

The minimum number of bins that any bounded space online algorithm needs to place the items of phase i is $N((2^i-1)^d-(2^i-2)^d)/(2^i-1)^d = N(1-(\frac{2^i-2}{2^i-1})^d)$. 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 $\lceil \log d \rceil \leq 1 + \log d$, it is greater than $N(1-(\frac{2d-2}{2d-1})^d) = N(1-(1-\frac{1}{2d-1})^d) \geq N(1-e^{-1/2}) > 0.39N$ for all $d \geq 2$. 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. □

In [14], we give specific upper and lower bounds for dimensions $2, \ldots, 7$.

3 Packing 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. An arriving hyperbox h of dimensions $(h_1, h_2, \ldots, h_d)$ 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 $1/(i+1) < h_i \leq 1/i$, 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 $1/(i+1) < 2^{f_i} h_i \leq 1/i$. 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 h, 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 h 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 h.

Now consider the components of h 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 $\in (1/(2^f(s_i+1)), 1/(2^f s_i)]$ 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 h 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. □

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 $(2^r - 1)/(s_1 s_2 \ldots s_d) \leq (2^r-1)/M^r \leq 2^r/M^r$. (We get the factor $2^r - 1$ by enumerating over all possible choices of the values f_i.) We get that the fraction of each bin that is filled is at least

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

The first 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 $(h_1, \ldots, h_d)$ and type $(s_1, \ldots, s_d)$ is defined as

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

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

Lemma 3.1 *For all input sequences σ, $\mathrm{cost}_{\mathrm{alg}}(\sigma) \leq \sum_{h\in\sigma} w_\varepsilon(h) + O(1)$.*

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 $(s_1, \ldots, s_d)$. 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 $h \in \sigma$ with components $(h_1, \ldots, h_d)$ and type $(s_1, \ldots, s_d)$, let $h' = (h'_1, \ldots, h'_d) \in \sigma'$ be defined as follows. For $i \notin L$, $h'_i = h_i$. For $i \in L$, $h'_i = 1/(s_i + 1) + \delta < 1/s_i$. As h and h' 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, h and h' have the same weight.

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

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

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 $h = (h_1, \ldots, h_d)$ is defined as $\prod_{i\notin L} h_i$, 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, S and T, such that for each dimension j in S, we have that the j-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 T. (Thus such a partition can always be found by taking $S = \emptyset$.)

For a hyperbox $h \in X$, define the *generalized pseudo-volume* of the components in T by $\tilde{v}(h, T) = \prod_{j\in T} h_j$, where h_j is the jth component of h. Define the total generalized pseudo-volume of all hyperboxes in a set X by $\tilde{v}(X, T) = \sum_{h\in X} \tilde{v}(h, T)$.

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

Proof. We prove the claim by induction on the number of dimensions in T. For $|T|=0$, 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 $|T|=0,\ldots,r-1$. Suppose $|S|=d-r<d$. Take any dimension $i\in T$. We replace each hyperbox h, with component h_i in dimension i, by $\lfloor N^2h_i\rfloor$ hyperboxes that have $\frac{1}{N^2}$ as their i-th component, and are identical to h in all other components. Here N is taken sufficiently large, such that $\frac{1}{N}<h_i$. 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 $1-\frac{1}{N^2h_i}\geq 1-\frac{1}{N}$ smaller than that of X. So if we write $T'=T\backslash\{i\}$, we have $\tilde{v}(X',T')\cdot\frac{1}{N^2}\geq\tilde{v}(X,T)(1-\frac{1}{N})$. By induction, it takes at least

$$\tilde{v}(X',T')\cdot(1-\frac{1}{N})^{r-1}/\prod_{j\in S\cup\{i\}}k_j$$

bins to pack the modified input X'. Using that $k_i=N^2$, this is $\tilde{v}(X,T)\cdot(1-\frac{1}{N})^r/\prod_{j\in S}k_j$ bins. □

Letting $\gamma=1-(1-\frac{1}{N})^d$, we get that the required number is at least $\tilde{v}(X,T)(1-\gamma)/\prod_{j\in S}k_j$ bins, where $\gamma\to 0$ as $N\to\infty$. In the remainder, we will take S to be the dimensions where the components of the hyperboxes in X are large, and T the dimensions where they are small. Note that this choice of S satisfies the constraints on S 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 $N'w_\varepsilon(p)(1-\gamma)(1-\varepsilon)$. 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 $(1-\varepsilon)\alpha_\varepsilon$. $\square$.

Furthermore, we can determine the asymptotic performance ratio of our algorithm for hyperbox packing. Comparing to the unbounded space algorithm in [11] we can see that all the weights we defined are smaller than or equal to the weights used in [11]. 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 [25].

4 Variable-sized packing

In this section we consider the problem of multidimensional packing where the bins used can have different sizes. We assume that all bins are hypercubes, with sides $\alpha_1 < \alpha_2 < \ldots < \alpha_m = 1$. In fact our algorithm is more general and works for the case where the bins are hyperboxes with dimensions α_{ij} $(i = 1, \ldots, m,\ j = 1, \ldots, d)$. We present the special case of bins that are hypercubes in this paper in order to avoid an overburdened notation and messy technical details.

The main structure of the algorithm is identical to the one in Section 3. The main problem in adapting that algorithm to the current problem is selecting the right bin size to pack the items in. In the one-dimensional variable-sized bin packing problem, it is easy to see which bin will accommodate any given item the best; here it is not so obvious how to select the right bin size, since in one dimension a bin of a certain size might seem best whereas for other dimensions, other bins seem more appropriate.

We begin by defining types for hyperboxes based on their components and the available bin sizes. Once again we use a parameter ε. The value of M as a function of ε is again picked so that $M \geq 1/(1-(1-\varepsilon)^{1/(d+2)}) - 1$. An arriving hyperbox h of dimensions $(h_1, h_2, \ldots, h_d)$ is classified as one of at most $(2mM/\alpha_1 - 1)^d$ types depending on its components: a type of a hyperbox is the vector of the types of its components. We define

$$T_i = \left\{ \frac{\alpha_i}{j} \middle| j \in \mathbb{N}, \frac{\alpha_i}{j} \geq \frac{\alpha_1}{2M} \right\}, \qquad T = \bigcup_{i=1}^{m} T_i.$$

Let the members of T be $1 = t_1 > t_2 > \ldots > t_{q'} = \alpha_1/M > \ldots > t_q = \alpha_1/(2M)$. The interval I_j is defined to be $(t_{j+1}, t_j]$ for $j = 1, \ldots, q'$. Note that these intervals are disjoint and that they cover $(\alpha_1/M, 1]$.

A component larger than α_1/M has type i if $h_i \in I_i$, and is called large. A component smaller than α_1/M has type i, where $q' \leq i \leq q-1$, if there exists a non-negative integer f_i such that $t_{i+1} < 2^{f_i} h_i \leq t_i$. Such components are called small. Thus in total there are $q - 1 \leq 2mM/\alpha_1 - 1$ component types.

Bin selection We now describe how to select a bin for a given type. Intuitively, the size of this bin is chosen in order to maximize the number of items packed relative to the area used. This is done as follows.

For a given component type s_i and bin size α_j, write $F(s_i, \alpha_j) = \max\{k \mid \alpha_j/k \geq t_{s_i}\}$. Thus for a large component, $F(s_i, \alpha_j)$ is the number of times that a component of type s_i fits in an interval of length α_j. This number is uniquely defined due to the definition of the numbers t_i. Basically, the general classification into types is too fine for any particular bin size, and we use $F(s_i, \alpha_j)$ to get a less refined classification which only considers the points t_i of the form α_j/k.

Denote by L the set of components in type $s = (s_1, \ldots, s_d)$ that are large. If $L = \emptyset$, we use a bin of size 1 for this type. Otherwise, we place this type in a bin of any size α_j which maximizes[1]

$$\prod_{i \in L} \frac{F(s_i, \alpha_j)}{\alpha_j}. \tag{2}$$

Thus we do not take small components into account in this formula. Note that for a small component, $F(s_i, \alpha_j)$ is not necessarily the same as the number of times that such a component fits into any interval of length α_j. However, it is at least M for any small component.

When such a bin is opened, it is split into $\prod_{i=1}^{d} F(s_i, \alpha_j)$ identical sub-bins of dimensions $(\alpha_j/F(s_1, \alpha_j), \ldots, \alpha_j/F(s_d, \alpha_j))$. These bins are then further sub-divided into sub-bins in order to place hyperboxes in "well-fitting" sub-bins, in the manner which is described in Section 3.

Similarly to in that section, the following claim can now be shown.

Claim 8 *The occupied volume in each closed bin of type $s = (s_1, \ldots, s_d)$ is at least*

$$V_{s,j} = (1 - \varepsilon)\alpha_j^d \prod_{i \in L} \frac{F(s_i, \alpha_j)}{F(s_i, \alpha_j) + 1},$$

where L is the set of large components in this type and α_j is the bin size used to pack this type.

We now define a weighting function for our algorithm. The weight of a hyperbox h with components $(h_1, \ldots, h_d)$ and type $s = (s_1, \ldots, s_d)$ is defined as

$$w_\varepsilon(h) = \frac{1}{1 - \varepsilon} \left(\prod_{i \notin L} h_i \right) \left(\prod_{i \in L} \frac{\alpha_j}{F(s_i, \alpha_j)} \right),$$

where L is the set of large components in this type and α_j is the size of bins used to pack this type.

In order to prove that this weighting function works (gives a valid upper bound for the cost of our algorithm), we will want to modify components s_i to the smallest possible component such that $F(s_i, \alpha_j)$ does not change. (Basically, a component will be rounded to

[1] For the case that the bins are hyperboxes instead of hypercubes, we here get the formula $\prod_{i \in L}(F(s_i, \alpha_{ij})/\alpha_{ij})$, and similar changes throughout the text.

$\alpha_j/(F(s_i,\alpha_j)+1)$ plus a small constant.) However, with variable-sized bins, when we modify components in this way, the algorithm might decide to pack the new hyperbox differently. (Remember that $F(s_i,\alpha_j)$ is a "less refined" classification which does not take other bin sizes than α_j into account.) To circumvent this technical difficulty, we will show first that as long as the algorithm keeps using the same bin size for a given item, the volume guarantee still holds.

For a given type $s=(s_1,\ldots,s_d)$ and the corresponding set L and bin size α_j, define an *extended type* $\mathrm{Ext}(s_1,\ldots,s_d)$ as follows: an item h is of extended type $\mathrm{Ext}(s_1,\ldots,s_d)$ if each large component $h_i \in (\frac{\alpha_j}{F(s_i,\alpha_j)+1}, \frac{\alpha_j}{F(s_i,\alpha_j)}]$ and each small component h_i is of type s_i.

Corollary 4.1 *Suppose items of extended type* $\mathrm{Ext}(s_1,\ldots,s_d)$ *are packed into bins of size* α_j. *Then the occupied volume in each closed bin is at least* $V_{s,j}$.

Proof. In the proof of Claim 8, we only use that each large component h_i is contained in the interval $(\frac{\alpha_j}{F(s_i,\alpha_j)+1}, \frac{\alpha_j}{F(s_i,\alpha_j)}]$. Thus the proof also works for extended types.

Lemma 4.1 *For all input sequences* σ, $\mathrm{cost}_{\mathrm{alg}}(\sigma) \le \sum_{h\in\sigma} w_\varepsilon(h) + O(1)$.

Proof. In order to prove the claim, it is sufficient to show that each closed bin of size α_j contains items of total weight of at least α_j^d. Consider a bin of this size filled with hyperboxes of type $s=(s_1,\ldots,s_d)$. It is sufficient to consider the subsequence σ of the input that contains only items of this type, since all types are packed independently. This subsequence only uses bins of size α_j so we may assume that *no other sizes of bins are given*. 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 $h\in\sigma$ with components $(h_1,\ldots,h_d)$ and type $s=(s_1,\ldots,s_d)$, let $h'=(h'_1,\ldots,h'_d)\in\sigma'$ be defined as follows. For $i\notin L$, $h'_i=h_i$. For $i\in L$, $h'_i=\alpha_j/(F(s_i,\alpha_j)+1)+\delta < \alpha_j/F(s_i,\alpha_j)$.

Note that h' is of extended type $\mathrm{Ext}(s_1,\ldots,s_d)$. Since only one size of bin is given, the algorithm packs σ' in the same way as it packs σ. Moreover, according to the definition of weight above, h and h' have the same weight.

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

$$\begin{aligned}\frac{w_\varepsilon(h')}{v(h')} = \frac{w_\varepsilon(h)}{v(h')} &= \frac{1}{1-\varepsilon}\left(\prod_{i\notin L} h'_i\right)\left(\prod_{i\in L}\frac{\alpha_j}{F(s_i,\alpha_j)}\right)\Bigg/\prod_{i=1}^{d} h'_i \\ &= \frac{1}{1-\varepsilon}\prod_{i\in L}\frac{\alpha_j}{F(s_i,\alpha_j)h'_i} > \frac{1}{1-\varepsilon}\prod_{i\in L}\frac{F(s_i,\alpha_j)+1}{F(s_i,\alpha_j)+M\frac{\alpha_j}{\alpha_1}\delta}.\end{aligned}$$

Here we have used in the last step that a component with a large type fits less than M times in a (one-dimensional) bin of size α_1, and therefore less than $M\frac{\alpha_j}{\alpha_1}$ times in a bin of size $\alpha_j \ge \alpha_1$.

As δ tends to zero, this bound approaches $\alpha_j^d / V_{s,j}$. We find

$$w_\varepsilon(h) \geq \alpha_j^d \frac{v(h')}{V_{s,j}} \qquad \text{for all } h \in \sigma.$$

Then Corollary 4.1 implies that the total weight of items in a closed bin of size α_j is no smaller than α_j^d, which is the cost of such a bin.

Suppose the optimal solution for a given input sequence σ uses n_j bins of size α_j. Denote the ith bin of size α_j by $B_{i,j}$. Then

$$\frac{\sum_{h\in\sigma} w_\varepsilon(h)}{\sum_{j=1}^m \alpha_j^d n_j} = \frac{\sum_{j=1}^m \sum_{i=1}^{n_j} \sum_{h\in B_{i,j}} w_\varepsilon(h)}{\sum_{j=1}^m \alpha_j^d n_j} = \frac{\sum_{j=1}^m \sum_{i=1}^{n_j} \sum_{h\in B_{i,j}} w_\varepsilon(h)}{\sum_{j=1}^m \sum_{i=1}^{n_j} \alpha_j^d} \ .$$

This implies that the asymptotic worst case ratio is upper bounded by

$$\max_j \max_{X_j} \sum_{h\in X_j} w_\varepsilon(h)/\alpha_j^d, \tag{3}$$

where the second maximum is taken over all sets X_j that can be packed in a bin of size α_j. Similarly to Section 3, it can now be shown that this weighting function is also "optimal" in that it determines the true asymptotic performance ratio of our algorithm.

In particular, it can be shown that packing a set of hyperboxes X that have the same type vectors of large and small dimensions takes at least

$$\sum_{h\in X} \prod_{i\notin L} \frac{h_i}{\alpha_j} \Bigg/ \prod_{i\in L} F(s_i, \alpha_j)$$

bins of size α_j, where h_i is the ith component of hyperbox h, s_i is the type of the ith component, and L is the set of large components (for all the hyperboxes in X). Since the cost of such a bin is α_j^d, this means that the total cost to pack N' copies of some item h is at least $N' w_\varepsilon(h)(1-\varepsilon)$ when bins of this size are used. However, it is clear that using bins of another size α_k does not help: packing N' copies of h into such bins would give a total cost of

$$N' \left(\prod_{i\notin L} h_i \right) \left(\prod_{i\in L} \frac{\alpha_k}{F(s_i, \alpha_k)} \right) \ .$$

Since α_j was chosen to maximize $\prod_{i\in L}(F(s_i,\alpha_j)/\alpha_j)$, this expression cannot be less than $N' w_\varepsilon(h)(1-\varepsilon)$. More precisely, any bins that are not of size α_j can be replaced by the appropriate number of bins of size α_j without increasing the total cost by more than 1 (it can increase by 1 due to rounding).

This implies that our algorithm is optimal among online bounded space algorithms.

5 Resource augmented packing

The resource augmented problem is now relatively simple to solve. In this case, the online algorithm has bins at its disposal that are hypercubes of dimensions $b_1 \times b_2 \times \ldots \times b_d$. We can use the algorithm from Section 3 with the following modification: the types for dimension j are not based on intervals of the form $(1/(i+1), 1/i]$ but rather intervals of the form $(b_j/(i+1), b_j/i]$.

Then, to pack items of type $s = (s_1, \ldots, s_d)$, a bin is split into $\prod_{i=1}^{d} s_i$ identical sub-bins of dimensions $(b_1/s_1, \ldots, b_d/s_d)$, and then subdivided further as necessary.

We now find that each closed bin of type $s = (s_1, \ldots, s_d)$ is full by at least

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

where L is the set of large components in this type, and $B = \prod_{j=1}^{d} b_i$ is the volume of the bins of the online algorithm.

We now define the weight of a hyperbox h with components $(h_1, \ldots, h_d)$ and type $s = (s_1, \ldots, s_d)$ as

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

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

This can be shown to be valid similarly as before, and it can also be shown that items can not be packed better. However, in this case we are additionally able to give explicit bounds for the asymptotic performance ratio.

5.1 The asymptotic performance ratio

Csirik and Woeginger [13] showed the following for the one-dimensional case.

For a given bin size b, define an infinite sequence $T(b) = \{t_1, t_2, \ldots\}$ of positive integers as follows:

$$t_1 = \lfloor 1 + b \rfloor \qquad \text{and} \qquad r_1 = \frac{1}{b} - \frac{1}{t_1},$$

and for $i = 1, 2, \ldots$

$$t_{i+1} = \lfloor 1 + \frac{1}{r_i} \rfloor \qquad \text{and} \qquad r_{i+1} = r_i - \frac{1}{t_{i+1}}.$$

Define

$$\rho(b) = \sum_{i=1}^{\infty} \frac{1}{t_i - 1}.$$

Lemma 5.1 *For every bin size $b \geq 1$, there exist online bounded space bin packing algorithms with worst case performance arbitrarily close to $\rho(b)$. For every bin size $b \geq 1$, the bound $\rho(b)$ cannot be beaten by any online bounded space bin packing algorithm.*

The following lemma was proved in Csirik and Van Vliet [11] for a specific weighting function which is independent of the dimension, and is similar to a result of Li and Cheng [26]. However, the proof holds for any positive one-dimensional weighting function w. We extend it for the case where the weighting function depends on the dimension. For a one-dimensional weighting function w_j and an input sequence σ, define $w_j(\sigma) = \sum_{h \in \sigma} w_j(h)$. Furthermore define $W_j = \sup_\sigma w_j(\sigma)$, where the supremum is taken over all sequences that can be packed into a one-dimensional bin.

Lemma 5.2 *Let σ be a list of d-dimensional rectangles, and let Q be a packing which packs these rectangles into a d-dimensional unit cube. Let w_j $(j = 1, \ldots, d)$ be arbitrary one-dimensional weighting functions. For each $h \in \sigma$, we define a new hyperbox h' as follows: $s_j(h') = w_j(s_j(h))$ for $1 \leq j \leq d$. Denote the resulting list of hyperboxes by σ'. Then there exists a packing Q' which packs σ' into a cube of size $(W_1, \ldots, W_d)$.*

Proof. We use a construction analogous to the one in [11]. We transform the packing $Q = Q^0$ of σ into a packing Q^d of σ' in a cube of the desired dimensions. This is done in d steps, one for each dimension. Denote the coordinates of item h in packing Q^i by $(x_1^i(h), \ldots, x_d^i(h))$, and its dimensions by $(s_1^i(h), \ldots, s_d^i(h))$.

In step i, the coordinates as well as the sizes in dimension i are adjusted as follows. First we adjust the sizes and set $s_i^i(h) = w_i(s_i(h))$ for every item h, leaving other dimensions unchanged.

To adjust coordinates, for each item h in packing Q^{i-1} we find the "left-touching" items, which is the set of items g which overlap with h in $d-1$ dimensions, and for which $x_i^{i-1}(g) + s_i^{i-1}(g) = x_i^{i-1}(h)$. We may assume that for each item h, there is either a left-touching item or $x_i^{i-1}(h) = 0$.

Then, for each item h that has no left-touching items, we set $x_i^i(h) = 0$. For all other items h, starting with the ones with smallest i-coordinate, we make the i-coordinate equal to $\max(x_i^i(g) + s_i^i(g))$, where the maximum is taken over the left-touching items of h in packing S^{i-1}. Note that we use the new coordinates and sizes of left-touching items in this construction, and that this creates a packing without overlap.

If in any step i the items need more than W_i room, this implies a chain of left-touching items with total size less than 1 but total weight more than W_i. From this we can find a set of one-dimensional items that fit in a bin but have total weight more than W_i (using weighting function w_i), which is a contradiction.

As in [11], this implies immediately that the total weight that can be packed into a unit-sized bin is upper bounded by $\prod_{i=1}^d W_i$, which in the present case is $\prod_{i=1}^d \rho(b_i)$. Moreover, by extending the lower bound from [13] to d dimensions exactly as in [11], it can be seen that the asymptotic performance ratio of any online bounded space bin packing algorithm can also not be lower than $\prod_{i=1}^d \rho(b_i)$.

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 $\Theta(\log d)$. Giving a explicit expression for the competitive ratio in variable sized packing (as a function of the bin sizes) would be harder. Already in [30] where an optimal one-dimensional bounded space algorithm was given for the variable sized problem, its ratio is unknown. It is interesting to find out whether in the multidimensional case the worst case occurs when only unit sized bins are available.

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*, pages 189–196. ACM/SIAM, 2004.

[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*, pages 179–188. ACM/SIAM, 2004.

[9] Janos Csirik. An online algorithm for variable-sized bin packing. *Acta Informatica*, 26:697–709, 1989.

[10] Janos Csirik, J. B. G. Frenk, and M. Labbe. Two dimensional rectangle packing: On line methods and results. *Discrete Applied Mathematics*, 45:197–204, 1993.

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

[12] 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.

[13] János Csirik and Gerhard J. Woeginger. Resource augmentation for online bounded space bin packing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 296–304, Jul 2000.

[14] Leah Epstein and Rob van Stee. Bounds for online bounded space hypercube packing. Technical Report SEN-E0417, CWI, Amsterdam, 2004.

[15] 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.

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

[17] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15:222–230, 1986.

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

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

[20] 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.

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

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

[23] 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.

[24] 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.

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

[26] K. Li and K. H. Cheng. Generalized First-Fit algorithms in two and three dimensions. *International Journal on Foundations of Computer Science*, 2:131–150, 1990.

[27] Keqin Li and Kam-Hoi Cheng. A generalized harmonic algorithm for on-line multi-dimensional bin packing. Technical Report UH-CS-90-2, University of Houston, January 1990.

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

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

[30] Steve S. Seiden. An optimal online algorithm for bounded space variable-sized bin packing. *SIAM Journal on Discrete Mathematics*, 14(4):458–470, 2001.

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

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

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

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

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

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

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

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