On Variable-Sized Multidimensional Packing

Leah Epstein[*] Rob van Stee[†]

Abstract

The main contribution of this paper is an optimal bounded space online algorithm for variable-sized multidimensional packing. In this problem, hyperboxes must be packed in d-dimensional bins of various sizes, and the goal is to minimize the total volume of the used bins. We show that the method used can also be extended to deal with the problem of resource augmented multidimensional packing, where the online algorithm has larger bins than the offline algorithm that it is compared to. Finally, we give new lower bounds for unbounded space multidimensional bin packing of hypercubes.

1 Introduction

In this paper, we consider the problem of online variable-sized multidimensional bin packing. In the d-*dimensional box packing* problem, we receive a sequence σ of *items* $p_1, p_2, \ldots, p_n$. 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 $(x_1(p), \ldots, x_d(p))$, where $0 \le x_i(p)$ and $x_i(p) + s_i(p) \le 1$ for $1 \le i \le d$. 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.

We focus on the *variable-sized* box packing problem, where bins of various sizes can be used to pack the items, and the goal is to minimize the total volume of all the bins used. To avoid a cumbersome notation, we assume that the available bins are hypercubes, although our algorithm can also be extended to the case where the bins are hyperboxes. The available sizes of bins (edge lengths) are denoted by $\alpha_1 < \alpha_2 < \ldots < \alpha_m = 1$.

Items arrive *online*, which means that each item must be assigned in turn, without knowledge of the next items. We consider *bounded space* algorithms, which have the property that they only have 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 standard measure of algorithm quality for box packing is the *asymptotic performance ratio*, which we now define. For a given input sequence σ, let $\mathrm{cost}_{\mathcal{A}}(\sigma)$ be the number of bins used by

[*]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 Scientifi c Research (NWO), project number SION 612-061-000.

algorithm $\mathcal{A}$ on σ. Let cost(σ) 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

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

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 $\mathcal{R}^\infty_{\mathcal{O}} = \inf_{\mathcal{A}\in\mathcal{O}} \mathcal{R}^\infty_{\mathcal{A}}$. Given $\mathcal{O}$, our goal is to find an algorithm with asymptotic performance ratio close to $\mathcal{R}^\infty_{\mathcal{O}}$. In this paper, we also consider the *resource augmented* box packing problem, where the online algorithm has larger bins at its disposal than the offline algorithm, and the goal is to minimize the (relative) number of bins used. Here all online bins are of the same size, and all the offline bins are of the same size, but these two sizes are not necessarily the same.

Previous Results: The classic online bin packing problem was first investigated by Ullman [17]. The lower bound currently stands at 1.54014, due to van Vliet [18]. 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 $(1+\varepsilon)\Pi_\infty$ [11]. They also showed there is no bounded space algorithm with a performance ratio below Π_∞. Currently the best known unbounded space upper bound is 1.58889 due to Seiden [15].

The variable-sized bin packing problem was first investigated by Friesen and Langston [8]. Csirik [3] proposed the VARIABLE HARMONIC algorithm and showed that it has performance ratio at most Π_∞. Seiden [14] showed that this algorithm is optimal among bounded space algorithms.

The resource augmented bin packing problem was studied by Csirik and Woeginger [6]. They show that the optimal bounded space asymptotic performance ratio is a function $\rho(b)$ of the online bin size b.

The online hyperbox packing problem was first investigated by Coppersmith and Raghavan [2], who give an algorithm based on NEXT FIT with performance ratio $\frac{13}{4} = 3.25$ for $d = 2$. Csirik, Frenk and Labbe [4] give an algorithm based on FIRST FIT with performance ratio $\frac{49}{16} = 3.0625$ for $d = 2$. Csirik and van Vliet [5] 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 [13] also gave a HARMONIC-based algorithm for $d = 2$ and $d = 3$.

Seiden and van Stee [16] improve the upper bound for $d = 2$ to 2.66013. Several lower bounds have been shown [9, 10, 19, 1]. The best lower bound for $d = 2$ is 1.907 [1], 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 [5]. A matching upper bound was shown in [7]. This was done by giving an extension of HARMONIC which uses only bounded space. In this paper a new technique of dealing with items that are relatively small in some of their dimensions was introduced.

Our Results: In this paper, we present a number of results for online multidimensional packing:

- We present a bounded space algorithm for the variable-sized multidimensional bin packing problem. An interesting feature of the analysis is that although we show the algorithm is optimal, 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 a asymptotic performance ratio of $\prod_{i=1}^{d} \rho(b_i)$ where $b_1 \times \cdots \times b_d$ is the size of the online bins.

- Additionally, we give a new general lower bound for unbounded space hypercube packing, where all items to be packed are hypercubes. We improve the bounds from [16].

For the variable-sized packing, the method used in this paper generalizes and combines the methods used in [7] and [14]. The are some new ingredients that we use in this paper. An important difference with the previous papers is that in the current problem, it is no longer immediately clear which bin size should be used for any given item. In [7] all bins have the same size. In [14] bins have one dimension and therefore a simple greedy choice gives the best option. In our case, all the dimensions in which an item is "large" must be taken into account, and the process of choosing a bin is done by considering all these dimensions. This in turn changes the analysis, as not only the packing could have been done in a different way by an offline algorithm, but also items could have been assigned to totally different bins (larger in some dimensions and smaller in others).

2 An optimal algorithm for bounded space 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 [7]. 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. The algorithm uses a parameter ε. The value of M as a function of ε is picked so that $M \geq 1/(1-(1-\varepsilon)^{1/(d+2)})-1$. 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

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

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 [7].

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

Claim 1 *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 $\alpha_j/(F(s_i, \alpha_j) + 1)$ 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 $s = (s_1, \dots, s_d)$ and the corresponding set L and bin size α_j, define an *extended type* $\text{Ext}(s_1, \dots, s_d)$ as follows: an item h is of extended type $\text{Ext}(s_1, \dots, s_d)$ 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 2.1 *Suppose items of extended type* $\text{Ext}(s_1, \dots, s_d)$ *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 1, 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. $\square$

Lemma 2.1 *For all input sequences* σ*,* $\text{cost}_{\text{alg}}(\sigma) \le \sum_{h \in \sigma} w_\varepsilon(h) + O(1)$*.*

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 $s = (s_1, \dots, s_d)$. It is sufficient to consider the subsequence σ of the input that contains only items of this type, since all types are packed independently. 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 $(h_1, \dots, h_d)$ and type $s = (s_1, \dots, s_d)$, let $h' = (h'_1, \dots, h'_d) \in \sigma'$ 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 $\text{Ext}(s_1, \dots, s_d)$. 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

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

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

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

Then Corollary 2.1 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. $\square$

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

$$\max_j \max_{X_j} \sum_{h \in X_j} w_\varepsilon(h)/\alpha_j^d, \tag{1}$$

where the second maximum is taken over all sets X_j that can be packed in a bin of size α_j. Similarly to in [7], 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 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.

3 An optimal algorithm for bounded space resource augmented packing

The resource augmented problem is now relatively simple to solve. In this case, the online algorithm has bins to its disposal that are hypercubes of dimensions $b_1\times b_2\times\ldots\times b_d$. We can use the algorithm from [7] 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 online bins.

The weight of a hyperbox h with components $(h_1,\ldots,h_d)$ and type $s=(s_1,\ldots,s_d)$ is now defined 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 to 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.

3.1 The asymptotic performance ratio

Csirik and Woeginger [6] 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 3.1 *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 is proved in Csirik and Van Vliet [5] for a specific weighting function which is independent of the dimension, and is similar to a result of Li and Cheng [12]. 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 3.2 *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. 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 [5]. 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 the 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. $\square$

As in [5], 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 [6] to d dimensions exactly as in [5], it can be seen that the asymptotic performance ratio of any online bounded space bin packing algorithm can also not be lower than $\prod_{i=1}^{d} \rho(b_i)$.

4 Lower bound for unbounded space hypercube packing

We construct a sequence of items to prove a general lower bound for hypercube packing in d dimensions. In this problem, all items to be packed are hypercubes. Take an integer $\ell > 1$. Let $\varepsilon > 0$ be a number smaller than $1/2^\ell - 1/(2^\ell + 1)$. The input sequence is defined as follows. We use a large integer N. Let

$$x_i = (2^{\ell+1-i} - 1)^d - (2^{\ell+1-i} - 2)^d \qquad i = 1, \ldots, \ell$$
$$x_0 = 2^{\ell d} - (2^\ell - 1)^d$$

In step 0, we let Nx_0 items of size $s_0 = (1+\varepsilon)/(2^\ell + 1)$ arrive. In step $i = 1, \ldots, \ell$, we let Nx_i items of size $s_i = (1+\varepsilon)/2^{\ell+1-i}$ arrive. For $i = 1, \ldots, \ell - 1$, item size s_i in this sequence divides all the item sizes $s_{i+1}, \ldots, s_\ell$. Furthermore, $\sum_{i=0}^{\ell} s_i = (1+\varepsilon)(1 - 1/2^\ell + 1/(2^\ell + 1)) < 1$ by our choice of ε.

The online algorithm receives steps $0, \ldots, k$ of this input sequence for some (unknown) $0 \le k \le \ell$.

A pattern is a multiset of items that fits (in some way) in a unit bin. A pattern is *dominant* if when we increase the number of items of the smallest size in that pattern, the resulting multiset no longer fits in a unit bin. A pattern is *greedy* if the largest item in it appears as many times as it can fit in a bin, and this also holds for all smaller items, each time taking the larger items that are in the pattern into account. Note that not all possible sizes of items need to be present in the bin.

For a pattern P of items that all have sizes in $\{s_0, \ldots, s_\ell\}$, denote the number of items of size s_i by P_i.

Lemma 4.1 *For any pattern P for items with sizes in $\{s_0, \ldots, s_\ell\}$,*

$$P_i \le (2^{\ell+1-i} - 1)^d - \sum_{j=i+1}^{\ell} 2^{(j-i)d} P_j \quad i = 1, \ldots, \ell, \qquad P_0 \le 2^{\ell d} - \sum_{j=1}^{\ell} 2^{(j-1)d} P_j \qquad (2)$$

Proof Suppose (2) does not hold for some i, and consider the smallest i for which it does not hold. Consider an item of size s_j with $j > i$ that appears in P. If there is no such j, we have a contradiction, since at most $(2^{\ell+1-i} - 1)^d$ items of size s_i fit in the unit hypercube for $i = 1, \ldots, \ell$, or at most $2^{\ell d}$ items of size s_0. This follows from Claim 4 in [7].

First suppose $i > 0$. If we replace an item of size s_j with $2^{(j-i)d}$ items of size s_i, the resulting pattern is still feasible: all the new size s_i items can be placed inside the hypercube that this size s_j item has vacated.

We can do this for all items of size s_j, $j > i$ that appear in the pattern. This results in a pattern with only items of size s_i or smaller. Since every size s_j item is replaced by $2^{(j-i)d}$ items of size s_i, the final pattern has more than $(2^{\ell+1-i} - 1)^d$ items of size s_i, a contradiction.

Now suppose $i = 0$. In this case $\lfloor (2^\ell + 1)/2^{\ell+1-j} \rfloor = 2^{j-1}$ items of size s_0 fit in a hypercube of size s_j, and the proof continues analogously. $\Box$

We define a *canonical packing* for dominant patterns. We create a grid in the bin as follows. One corner of the bin is designated as the origin O. We assign a coordinate system to the bin, where each positive axis is along some edge of the bin. The grid points are those points inside the bin that have all coordinates of the form $m_1(1+\varepsilon)/2^{m_2}$ for $m_1, m_2 \in \mathbb{N} \cup \{0\}$.

We pack the items in order of decreasing size. Each item of size s_i ($i = 1, \dots, \ell$) is placed at the available grid point that has all coordinates smaller than $1 - s_i$, all coordinates equal to a multiple of s_i and is closest to the origin. So the first item is placed in the origin. Each item of size s_0 is placed at the available grid point that has all coordinates equal to a multiple of s_1 (and not s_0) and is closest to the origin. Note that for these items we also use grid points with some coordinates equal to $(2^\ell - 1)(1+\varepsilon)/2^\ell$, unlike for items of size s_1. This is feasible because $(2^\ell - 1)(1+\varepsilon)/2^\ell + (1+\varepsilon)/(2^\ell + 1) = (1+\varepsilon)(1 - 1/2^\ell + 1/(2^\ell + 1)) < 1$.

In each step i we can place a number of items which is equal to the upper bound in Lemma 4.1. For $i = 1, \dots, \ell$, this is because a larger item of size s_j takes away exactly $2^{(j-i)d}$ grid points that are multiples of s_i, and originally there are $(2^{\ell+1-i} - 1)^d$ grid points of this form that have all coordinates smaller than $1 - s_i$. For $i = 0$, $2^{(j-1)d}$ grid points that are multiples of s_1 are taken away by an item of size s_j, from an original supply of $2^{\ell d}$. This shows that all patterns can indeed be packed in canonical form.

Lemma 4.2 *For this set of item sizes, any dominant pattern that is not greedy is a convex combination of dominant patterns that are greedy.*

Proof We use induction to construct a convex combination of greedy patterns for a given dominant pattern P. The induction hypothesis is as follows: the vector that describes the numbers of items of the t smallest types which appear in the pattern is a convex combination of greedy vectors for these types. Call such a pattern t-greedy.

The base case is $t = 1$. We consider the items of the smallest type that occurs in P. Since P is dominant, for this type we have that as many items as possible appear in P, given the larger items. Thus P is 1-greedy.

We now prove the induction step. Suppose that in P, items of type i appear fewer times than they could, given the larger items. Moreover, P contains items of some smaller type. Let i' be the largest smaller type in P. By induction, we only need to consider patterns in which all the items of type less than i that appear, appear as many times as possible, starting with items of type i'. (All other patterns are convex combinations of such patterns.)

We define two patterns P' and P'' such that P is a convex combination of them. First suppose $i' > 0$. P' is defined as follows: modify P by removing all items i and adding the largest smaller item that appears in P, of type i', $2^{(i-i')d}$ times per each item i. When creating P', we thus add the maximum amount of items of type i' that can fit for each removed item of type i. P is greedy with respect to all smaller items, and $s_{i'}$ divides s_i. Therefore the multiset P' defined in this way is a pattern, and is $(t+1)$-greedy.

P'' on the other hand is created by adding items of phase i to P and removing items of type i'. In particular, in the canonical packing for P, at each grid point for type i that is not removed due to a higher-type item, we place an item of size s_i and remove all items that overlap with this item. Since all items smaller than s_i appear as many times as possible given the larger items, all the removed items are of the next smaller type i' that appear in P. This holds because the items are packed in order of decreasing size, so this area will certainly be filled with items of type i' if the item of size i is not there.

In P'', the number of items of type i is now maximized given items of higher types. Only type i' items are removed, and only enough to make room for type i, so type i' remains greedy. Thus P''

is $(t+1)$-greedy. Each time that we add an item i, we remove exactly $2^{(i-i')d}$ items of type i'. So by adding an item i in creating P'', we remove exactly the same number of items of type i' as we add when we remove an item i while creating P'. Therefore, P is a convex combination of P' and P'', and we are done.

Now suppose $i' = 0$. (This is a special case of the induction step for $t+1=2$.) In this case, in P' each item of type i is replaced by $2^{(i-1)d}$ items of type 0. In the canonical packing, this is exactly the number of type 1 grid points that become available when removing an item of type i. Thus in P', the number of type 0 items is maximal (since it is sufficient to use type 1 gridpoints for them), and P' is 2-greedy.

Similarly, it can be seen that to create P'', we need to remove $2^{(i-1)d}$ items of type 0 in the canonical packing in order to place each item of type i. Then P'' is 2-greedy, and again P is a convex combination of P' and P''. $\square$

We can now formulate a linear program to lower bound the asymptotic performance ratio of any unbounded space online algorithm as in [16]. We will need the offline cost to pack any prefix of the full sequence. This is calculated as follows.

To pack the items of the largest type k, which have size $(1+\varepsilon)2^{k-\ell-1}$, we need $Nx_k/(2^{\ell+1-k}-1)^d$ bins because there are Nx_k such items. These are all packed identically: using a canonical packing, we pack as many items of smaller types in bins with these items as possible. (Thus we use a greedy pattern.) Some items of types $1,\ldots,k-1$ still remain to be packed. It is straightforward to calculate the number of items of type $k-1$ that still need to be packed, and how many bins this takes. We continue in the same manner until all items are packed.

Solving this linear program for $\ell = 11$ and several values of d gives us the following results.

d	Lower bound
1	1.5
2	1.64062
3	1.66809
4	1.67758
5	1.68405
6	1.68877
7	1.69204
8	1.69432
9	1.69599
10	1.69733

References

[1] 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.

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

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

[4] 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.

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

[6] 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.

[7] Leah Epstein and Rob van Stee. Optimal online bounded space multidimensional packing. In *Proc. of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, pages 207–216. ACM/SIAM, 2004.

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

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

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

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

[12] K. Li and K. H. Cheng. Generalized First-Fit algorithms in two and three dimensions. *International Journal on Foundations of Computer Science*, 1(2):131–150, 1990.

[13] 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.

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

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

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

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

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

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