# Competitive Multi-Dimensional Dynamic Bin Packing via L-Shape Bin Packing *

Prudence W.H. Wong and Fencol C.C. Yung

Department of Computer Science, University of Liverpool, UK
pwong@liverpool.ac.uk, ccyung@graduate.hku.hk

**Abstract.** We study $d$-dimensional dynamic bin packing for general $d$-dimensional boxes, for $d \geq 2$. This problem is a generalization of the bin packing problem in which items may arrive and depart dynamically. Our main result is a $3^d$-competitive online algorithm. We further study the 2- and 3-dimensional problem closely and improve the competitive ratios. Technically speaking, our $d$-dimensional result is due to a space efficient offline single bin packing algorithm, which is a variant of $d$-dimensional NFDH. We introduce an interesting notion of $d$-dimensional L-shape bin and show that effective offline packing into L-shape bin leads to effective online dynamic packing into unit-sized bins.
We also investigate the resource augmentation version of the problem where the online algorithm can use $d$-dimensional bins of size $s_1 \times s_2 \times \cdots \times s_d$ for $s_i \geq 1$ while the optimal offline algorithm uses unit-sized bins. We give conditions for the online algorithm to match the performance of the optimal offline algorithm, i.e., 1-competitive.

## 1 Introduction

Bin packing is a classical combinatorial optimization problem that has been studied since the early 70's and different variants continue to attract researchers' attention (see [5,7,9]). The problem was first studied in one-dimension (1-D) and has been extended to multi-dimension ($d$-D for $d \geq 1$). In $d$-D bin packing, the items are $d$-dimensional with length in the range $(0, 1]$ in each dimension and the bin is a $d$-dimensional bin with all lengths equal to 1. Items are oriented and cannot be rotated. The objective is to pack the items into a minimum number of unit-size bins such that the items do not overlap and do not exceed the boundary of the bin. The bin packing problem is NP-complete [13], even for 1-D.

The problem has been studied both in offline and online setting. In the offline setting, all the items and their sizes are given in advance. In the online setting, items may arrive at arbitrary time; item arrival time and item size are only known when an item arrives. The performance of an online algorithm is measured using competitive analysis [2]. Consider any online algorithm $\mathcal{A}$. Given an input $I$, let $OPT(I)$ and $\mathcal{A}(I)$ be the maximum number of bins used by the optimal offline algorithm and $\mathcal{A}$, respectively. Algorithm $\mathcal{A}$ is said to be $c$-competitive if there exists a constant $b$ such that $\mathcal{A}(I) \leq c\,OPT(I) + b$ for all $I$.

* The work is partly supported by EPSRC Grant EP/E028276/1. We thank Tim Hartnack, Lars Prädel, Isaac K.K. To for discussion at the beginning of the work.

| | 1-D | 2-D | 3-D | $d$-D |
|---|---|---|---|---|
| upper bound | 2.788 [6] | 7.788 [*] | 22.788 [*] | $3^d$ [*] |
| (previous) | | 8.5754 [11] | 35.346 [11] | $2 \times 3.5^d$ [11] |
| lower bound | 2.666 [19] | 3.70301 [11] | 4.85383 [11] | $d+1$ [11] |

**Table 1.** Competitive ratios. Results in this paper are marked with [*].

**Dynamic bin packing.** Most existing work focused on "static" bin packing in the sense that items do not depart. In some potential applications like warehouse storage, a more realistic model takes into consideration of dynamic arrival and departures of items. This natural generalization, known as *dynamic bin packing*, was introduced by Coffman, Garey and Johnson [6]. In this generalization, items arrive over time, reside for some period of time, and may depart at arbitrary time. Each item has to be assigned to a bin from the time it arrives until it departs. The objective is to minimize the maximum number of bins used over all time. Note that migration to another bin is not allowed yet rearrangement of items within a bin is allowed. One can imagine that warehouses (c.f. bins) may be geographically far from each other making migration infeasible but rearrangement within a warehouse is feasible.[1]

The dynamic bin packing problem was first studied in 1-D by Coffman, Garey and Johnson [6] who showed that a modified first-fit algorithm, which we called *FFM*, is 2.788-competitive. This algorithm works by classifying items into large (size larger than $\frac{1}{2}$) and small ones (size $\frac{1}{2}$ or less), then using a dedicated bin for each large item and using first-fit for the small ones. The 2.788 bound is derived from the 1.788-competitive ratio if all items are small [6]. Recently, Chan, Wong and Yung [4] have shown a lower bound that there is no algorithm better than 2.5-competitive and this is further improved to $8/3 \simeq 2.666$ very recently [19] .

Multi-dimensional dynamic bin packing has been studied by Epstein and Levy [11]. They gave a $2 \times 3.5^d$-competitive algorithm for $d$-D dynamic bin packing, i.e., 24.5 for $d = 2$ and 85.75 for $d = 3$. They further presented algorithms specifically for $d = 2$ and $d = 3$ and claimed that they are 8.5754- and 35.346-competitive, respectively. They also gave lower bounds of $d + 1$ for general $d$, 3.70301 and 4.85383 for $d = 2$ and $d = 3$.

Resource augmentation [14] has also been studied in 1-D dynamic bin packing [4] and online static bin packing [1, 10, 12] for various dimensions. In this setting, the online algorithm can use larger bins than the optimal offline algorithm. For 1-D dynamic bin packing, it is shown that using bins of double size is both necessary and sufficient to achieve 1-competitiveness [4].

**Our contribution.** In this paper, we study multi-dimensional dynamic bin packing and give the following results (see Table 1 for a summary).

- For $d$-D where $d \geq 2$, we present a $3^d$-competitive algorithm (Theorem 1).
- For 2-D and 3-D, we further improve the above general ratio. We give 7.788- and 22.788-competitive algorithms, respectively (Theorems 2 and 3).
- We consider resource augmentation and give conditions for the online algorithm to match the offline algorithm, i.e., 1-competitive (Corollary 1).

[1] If rearrangement within a bin is not allowed, one can show that there is no constant competitive deterministic online algorithm.

For the $d$-D result, our algorithm classifies items into large and small items. Roughly speaking, we show that large items can be handled as small items of lower dimensions, hence, we can focus on small items. The main idea is a testing procedure to check whether a new small item can be packed into existing bins. This naturally involves a space efficient offline single bin packing procedure, which is indeed an interesting problem by itself. Multi-dimensional NFDH (next-fit-decreasing-height) is a common strategy to achieve this; in particular, a formula has been given in [15, 16] for the minimum total volume of $d$-D cubes (i.e., all sides are equal) that can be packed without overflowing a bin. However, there is no matching results for $d$-D boxes of general size. We devise a single bin packing procedure using a variant of NFDH, which instead of packing items using the whole bin space, reserves space to accommodate the new item and tries to repack existing items into a so called L-shape space. At first glance, reserving space may be too pessimistic, yet it can be shown that packing boxes using full space may perform only as good as the L-shape approach (we skip the details due to space limit). Using this new packing procedure, we show that the same formula in [15] can be obtained even for packing boxes of general sizes.

**Notations and definitions.** We now give a precise definition of the problem and the necessary notations for further discussion. In general, a $d$-D object (item or bin) is called a $d$-D *cube* if all sides have the same length; and $d$-D *box* otherwise. A packing configuration is said to be *feasible* if all items do not overlap and the packing in each bin does not exceed the boundary of the bin; otherwise, the packing is said to *overflow* and is *infeasible*.

In $d$-D dynamic bin packing, $d$-D items arrive and depart at arbitrary time. When an item arrives, it must be assigned to a unit-sized bin immediately so that the resulting packing is feasible. The item then resides in the assigned bin until it departs, i.e., migration is not allowed. Rearrangement of items within a bin is allowed upon item arrival or departure. The objective is to minimize the maximum number of bins used over all time.

For 2-D packing, we call the two dimensions width and height. For general $d$-D packing, we name the $d$ dimensions $x_1, x_2, \cdots, x_d$ and denote the length of an item $R$ along dimension $x_i$ by $x_i(R)$. When the context is clear, we may also call $x_d$ the height. In the $d$-D packing algorithms, we use the concept of projection of higher dimension item to lower dimension item. We say that an item is *projected along* $x_d$ when the item is projected on the hyperplane of dimensions $x_1, x_2, x_3, \cdots, x_{d-1}$.

Several of our algorithms involve reserving some space for a new item and repacking existing items in a bin to check if the new item can be packed to this bin. If such a repacking is not feasible, it is understood that we restore the packing to the original configuration.

## 2 $d$-Dimensional Dynamic Bin Packing

In this section, we consider $d$-D dynamic bin packing for any $d \geq 2$ and present a $3^d$-competitive online algorithm, called DynamicPack($d$). Roughly speaking,

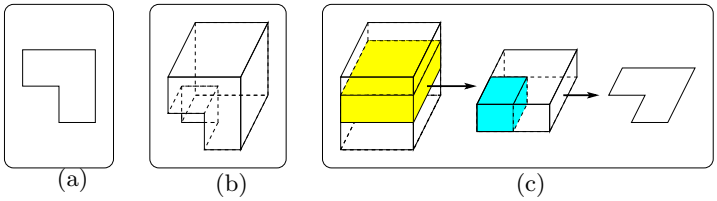


**Fig. 1.** (a) 2-D L-shape bin. (b) 3-D L-shape bin. (c) Removing a box at the bottom left corner from a layer of a 3-D bin and projecting on 2-D gives a 2-D L-shape bin.

when an item $R$ arrives, Algorithm DynamicPack($d$) checks if an existing bin can accommodate $R$ by reserving some space for $R$ and repacking existing items into the remaining space. If there is any bin that such repacking is feasible, $R$ is assigned to this bin. If no such bin exists, open a new bin for $R$. In other words, the algorithm involves a repacking procedure that rearranges items in a given bin. We present this repacking procedure in Section 2.1 and the overall bin assignment algorithm in Section 2.2. Furthermore, the algorithm distinguishes between small and large items. An item $R$ is said to be *small* if $x_i(R) \leq \frac{1}{2}$ for all $1 \leq i \leq d$, and *large* otherwise. In Section 2.2, we show that large items can be handled as small items of lower dimensions and so we can focus on small items.

### 2.1 Repacking Procedure for Small Items into a Single Bin

In this section, we present a procedure to repack small items into a single bin and give a formula for the minimum total volume of small items that can be packed in the bin. This procedure is invoked when a new item arrives, some space is reserved and existing items are repacked into the remaining space. We call the remaining space an L-shape bin. The repacking makes use of a variant of NFDH (next-fit-decreasing-height) approach.

Below we first formally define a $d$-D L-shape bin and describe some property of NFDH. Then we show that if a set of items cannot be packed in a $d$-D L-shape, the volume of these items is at least $2^d$ (Lemma 1).

***d*-D L-shape bin.** For any $d \geq 2$, a *$d$-D L-shape bin* is a unit $d$-D bin with a corner removed: the corner removed is a $d$-D cube with all sides equal to $\frac{1}{2}$. For the sake of reference, we say that the cube removed is the "bottom left-most" corner of the bin. See Figure 1 (a) and (b) for examples. We note the following property about a unit $d$-D bin and a $d$-D L-shape bin.

*Property 1.* Consider a unit $d$-D bin and a $d$-D L-shape bin.

(i) Any small item can be packed into the cube that is removed from the unit bin to form the L-shape bin.
(ii) Take a layer of the unit bin with length $h$ in dimension $x_d$ and length 1 in all other dimensions, if we remove a box of length $h$ in $x_d$ and length $\frac{1}{2}$ in other dimensions from bottom left corner of the layer and project along dimension $x_d$, we obtain a $(d-1)$-D L-shape bin. See Figure 1 (c) for an example.

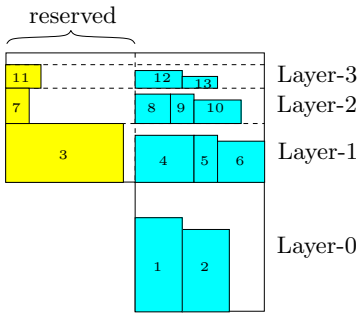


**Fig. 2.** Example for NFDH-LS(2). The number on the items is the order the items are packed. The item overflows from a layer (e.g., 3, 7 and 11) is packed in the first (reserved) partition of the next layer and the height of this item determines the height of the next layer. Packing then continues in the second partition.

**NFDH.** Our procedure to pack into an L-shape bin makes use of the idea of NFDH. In general, the $d$-D version of NFDH first sorts the items in descending order of the length of dimension $x_d$, say $R_1, R_2, R_3, \cdots$ such that $x_d(R_i) \geq x_d(R_{i-1})$. Then the items are packed into layers aligned to dimension $x_d$. The first layer has a "height" equal to $x_d(R_1)$. Items, projected along $x_d$, are then packed into this layer using some $(d-1)$-D packing algorithm until a certain item, say $R_j$, cannot be packed. Then the next layer is constructed with height equals to $x_d(R_j)$. We then observe the following property about NFDH.

*Property 2.* Suppose NFDH has packed $k$ layers with height $h_1 \geq h_2 \geq \cdots \geq h_k$.

(i) All the items in Layer-$i$ have height at least $h_{i+1}$.
(ii) If the $(d-1)$-D packing algorithm guarantees each layer is packed with a $(d-1)$-D volume of at least $V$, then the total $d$-D volume of items in Layer-$i$ $\geq h_{i+1}V$ and the total volume of all items in all layers $\geq \sum_{i=2}^{k} h_i V$.

**Packing an L-shape bin.** We now present the recursive procedure, called NFDH-LS($d$), that packs into a $d$-D L-shape bin. As to be shown in Lemma 1, if NFDH-LS($d$) cannot pack a set $S$ of small items into a $d$-D L-shape bin, the total volume of $S$ is more than $\frac{1}{2^d}$. We first describe the base case NFDH-LS(2).

**Single-bin repacking procedure NFDH-LS(2): packing small items into a 2-D L-shape bin.** We first sort the items in descending order of height. Items are packed into layers as follows (see Figure 2 for an example). *Layer-0* is the bottom square with height and width $\frac{1}{2}$. We pack the items (in order of height) to Layer-0 until a certain item, say $Q$, cannot be packed. Then the height of $Q$ becomes the height of *Layer-1*. Layer-1 is divided into two equal partitions each with width $\frac{1}{2}$. We place $Q$ into the first partition and continue packing the remaining items into the second partition until overflow. In general, the item that overflows from a layer is packed to the first partition of the next layer. The procedure returns whether all items can be packed in the L-shape bin (i.e., whether the last layer can be packed without overflow).

**Single-bin repacking procedure NFDH-LS($d$): packing small items into a $d$-D L-shape bin.** We first sort the items in descending order of $x_d(\cdot)$.

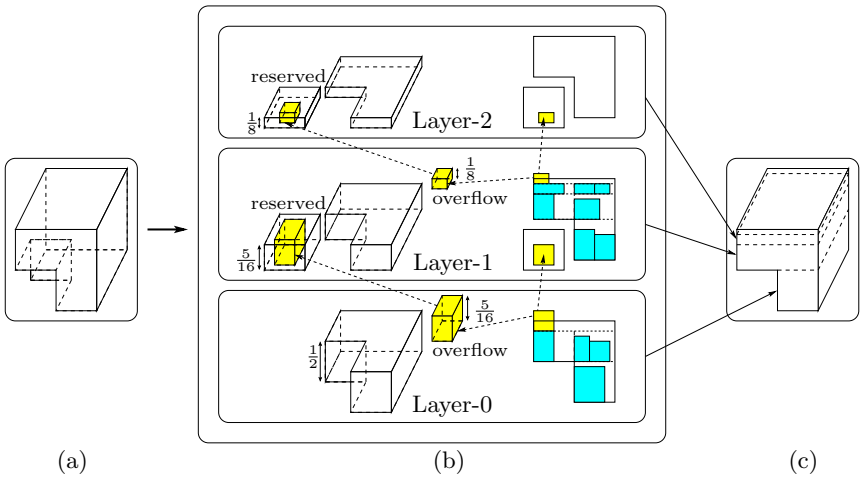


**Fig. 3.** Example for NFDH-LS(3). (a) 3-D L-shape bin. (b) In each layer, the 2-D L-shape on the right is the projection of the layer. The overflow item is packed in the reserved box in the next layer and the height of this item determines the height of the next layer. (c) The layers as shown in the 3-D L-shape.

Items are then packed one by one in the sorted order into layers along dimension $x_d$ (see Figure 3). *Layer-0* has height $\frac{1}{2}$ along dimension $x_d$. Projecting the layers along dimension $x_d$ forms $(d-1)$-D L-shape bins (Property 1 (ii)). Items are packed with dimension $x_d$ aligned to the height of the layer, and then projecting the items along dimension $x_d$, we use NFDH-LS$(d-1)$ to recursively pack the items into Layer-0. For Layer-$i$, if NFDH-LS$(d-1)$ reports feasible packing, we try to include the next item into Layer-$i$; if packing is not feasible with the inclusion of a certain item $R$, we create Layer-$(i+1)$ with $x_d(R)$ as the layer height along dimension $x_d$. We then reserve a box with height $x_d(R)$ along dimension $x_d$ and length $\frac{1}{2}$ along all other dimensions; the box is at the bottom leftmost corner of the layer. Hence, projecting the layer along dimension $x_d$ gives a $(d-1)$-D L-shape bin (Property 1 (ii)). We pack $R$ into the reserved box and continue the process with the remaining items: adding them one by one and use NFDH-LS$(d-1)$ recursively to pack into the $(d-1)$-D L-shape bin.

Whenever NFDH-LS$(d-1)$ reports packing is not feasible with the next item included, a new layer is created. At the end, if all the layers constructed can be packed into the $d$-D L-shape bin without overflow, then NFDH-LS$(d)$ reports a feasible packing, otherwise NFDH-LS$(d)$ reports infeasible packing.

**Lemma 1.** *Let $S$ be a set of items with sides bounded by $\frac{1}{2}$. If* NFDH-LS$(d)$ *cannot pack $S$ into a $d$-D L-shape bin, the total volume of $S$ is more than $\frac{1}{2^d}$.*

*Proof.* We prove the lemma by induction on the dimension. Base case: when $d = 2$, Layer-0 together with the first item on Layer-1 has width greater than $\frac{1}{2}$. For $i > 0$, all items in Layer-$i$ (except the first item in the reserved box) together with the first item in Layer-$(i+1)$ has width greater than $\frac{1}{2}$. All items in Layer-$i$ has a height at least the height of Layer-$(i+1)$ (Property 2 (i)). If not all items

can be packed, the total height of the layers (including the one that cannot fit) is more than 1. Since Layer-0 has height at most $\frac{1}{2}$, the total volume of all items is greater than $(1-\frac{1}{2}) \times \frac{1}{2} = \frac{1}{2^2}$ (Property 2 (ii)).

Assume the lemma is true for dimension $d-1$. For dimension $d$, we can use a similar argument as the base case to show that if not all items can be packed, the total height of all layers in dimension $d$ is at least 1. Using the induction hypothesis, all items in each layer has volume greater than $\frac{1}{2^{d-1}}$. Hence, the total volume of all items is greater than $(1-\frac{1}{2}) \times \frac{1}{2^{d-1}} = \frac{1}{2^d}$ (Property 2 (ii)).

The results in this section can be generalized as follows. Let $S$ be a set of items with sides bounded by $\frac{1}{k}$. There is a packing algorithm such that if $S$ cannot be packed into a $d$-D L-shape bin, then the total volume of $S$ is more than $(1-\frac{1}{k})^d$. As mentioned in the introduction, this bound has been shown for packing $d$-D cubes and we show that it is also true for packing $d$-D boxes.

### 2.2 Bin Assignment Algorithm

We now present a $3^d$-competitive online dynamic bin packing algorithm. We first present an algorithm AFReserveNFDH($d$) for packing small items (AF for any-fit) and then an algorithm DynamicPack($d$) for arbitrary items.

**Bin assignment algorithm AFReserveNFDH($d$) for small items.** When an item $R$ with sides bounded by $\frac{1}{2}$ arrives, we consider every bin in turn to reserve for $R$ the bottom leftmost cube of all sides $\frac{1}{2}$ and check if all the existing items in that bin can be packed into the $d$-D L-shape bin using NFDH-LS($d$). If NFDH-LS($d$) finds a bin that there is feasible packing, $R$ is packed in the reserved bottom leftmost corner of that bin and the other items are packed in the way NFDH-LS($d$) packs them. If there is no such existing bin, open a new bin for $R$. Lemma 2 is a direct consequence of Lemma 1.

**Lemma 2.** *Algorithm* AFReserveNFDH($d$) *is* $2^d$*-competitive for packing* $d$*-D items with sides bounded by* $\frac{1}{2}$.

**Algorithm DynamicPack($d$) for arbitrary sized items.** We distinguish items based on whether each dimension $x_i$ is larger than $\frac{1}{2}$ or at most $\frac{1}{2}$, and classify the items into $2^d$ classes. Consider items of the same class. If $x_i(\cdot) > \frac{1}{2}$ in this class, then in any packing, a line drawn parallel to dimension $x_i$ intercepts at most one item in that class. Suppose there are $z$ dimensions with length at most $\frac{1}{2}$ and $(d-z)$ dimensions larger than $\frac{1}{2}$. Then items in this class can be considered as $z$-D small items. We use AFReserveNFDH($z$) to pack the items by projecting along all the dimensions $x_i$ such that $x_i(\cdot) > \frac{1}{2}$. These items will be packed to align to the facets for those dimensions with $x_i(\cdot) > \frac{1}{2}$.

For the class where all dimensions are larger than $\frac{1}{2}$, we pack each in a separate bin (no two such items can be packed into the same bin by any packing).

**Theorem 1.** *Algorithm* DynamicPack($d$) *is* $3^d$*-competitive.*

*Proof.* For each $z$, we run a couple of Algorithm AFReserveNFDH($z$) and the number is at most $\binom{d}{z}$. By Lemma 2, the competitive ratio of DynamicPack($d$) is at most the sum over all ratios of AFReserveNFDH($z$), i.e., $\sum_{z=1}^{d} \binom{d}{z} 2^z = 3^d$.

## 3 Two- and Three-Dimensional Dynamic Bin Packing

In this section, we improve the results in Section 2 for $d = 2$ and 3 by more careful classification and assignment of items into bins. Similar to Section 2, the algorithms make use of reserve-and-repacking procedures to check if a new item can be assigned to an existing bin.

### 3.1 Two-Dimensional Dynamic Bin Packing

**Repacking Procedures** The procedures described in this section will be reused later and so they are stated in a more general form. We assume we are given a bin with width $u$ and height $v$ throughout Section 3.1. Before we present our two-dimensional repacking procedures, we first note the result by Steinberg [18] for static bin packing. In particular, we will use the following lemma which is implied by Theorem 1.1 in [18].

**Lemma 3 ( [18]).** *Given a bin with width $u$ and height $v$, if all items have width at most $\frac{u}{2}$ and height at most $v$, then any set of these items with total area at most $\frac{uv}{2}$ can fit into the same bin by using Steinberg's algorithm.*

While Steinberg's result gives a condition for a set of items to be fit into a bin, we also need a slightly different condition which bounds the area of existing items in a bin if a new item cannot be packed into the bin. The latter is a typical notion required when the objective is to minimize the number of bins. Below we describe two such packing procedures.

**Procedure 2DRepackNarrow: for items with width in $(0, \frac{u}{3}]$ and height in $(0, v]$.** We divide each bin into two partitions both with height $v$, and the first one with width $\frac{u}{3}$, second with width $\frac{2u}{3}$. Note that any item can fit into the first partition. When an item $R$ arrives, we reserve the first partition for $R$, and repack existing items using the second partition as a bin of width $\frac{2u}{3}$ and height $v$ using Lemma 3. If repacking is feasible, the packing configuration is returned as solution.

**Procedure 2DRepackMedium: for items with width in $(\frac{u}{3}, \frac{u}{2}]$ and height in $(0, v]$.** Note that since the items have width more than $\frac{u}{3}$, in any packing including the optimal one, at most two items can be packed side by side along the width. We divide each bin into two equal partitions each with width $\frac{u}{2}$ and height $v$. When a new item $R$ arrives, we reserve the first partition for $R$ and stack the existing items (in arbitrary order) along the height of the second partition. Similar to 2DRepackNarrow, if repacking is feasible, the packing configuration is returned as solution.

**Lemma 4.** *Consider a bin with width $u$ and height $v$.*

*(i) Consider items with width in $(0, \frac{u}{3}]$ and height in $(0, v]$. If* 2DRepackNarrow *returns an infeasible repacking, the total area of the existing items in the bin is more than $\frac{uv}{3}$.*

*(ii) Consider items with width in $(\frac{u}{3}, \frac{u}{2}]$ and height in $(0, v]$. If* 2DRepackMedium *returns an infeasible repacking, the total height of the existing items in the bin is more than $v$.*

*Proof.* (i) This means that existing items cannot be packed into the second partition (the larger one) of the bin. By Lemma 3, the total area of these items is more than $\frac{1}{2} \times \frac{2uv}{3} = \frac{uv}{3}$; otherwise, the items can be packed in the partition. (ii) This means the total height of the existing items exceed the height of the second partition, i.e., $v$.

**Bin Assignment Algorithm.** Using the above two procedures, we present an algorithm called 2DDynamicPack, which classifies items into three classes: narrow, medium-wide and wide according to their width. An item is said to be *narrow* if its width is in $(0, \frac{1}{3}]$; *medium-wide* if in $(\frac{1}{3}, \frac{1}{2}]$; and *wide* if in $(\frac{1}{2}, 1]$.

**Algorithm 2DDynamicPack.** Classify items into narrow, medium-wide, and wide as they arrive. Items of the same class are assigned to bins independently of other classes.

- **Narrow items.** When a narrow item $R$ arrives, find any bin that existing narrow items in the bin can be repacked using procedure 2DRepackNarrow into the larger partition of the bin and pack $R$ into the smaller partition of the bin. If no such bin exists, open a new bin for $R$.
- **Medium-wide items.** When a medium-wide item $R$ arrives, find any bin that existing medium-wide items in the bin can be repacked using procedure 2DRepackMedium into the second partition of the bin and pack $R$ into the first partition of the bin. If no such bin exists, open a new bin for $R$.
- **Wide items.** Items are packed so that the width of the item is aligned to the width of the bin, and then ignoring the width of the items, use the 1D algorithm FFM to pack according to the height of the items.

**Theorem 2.** *Algorithm* 2DDynamicPack *is* 7.788*-competitive.*

*Proof.* Let $OPT$ denote the maximum number of bins used by the optimal offline algorithm, and $n$, $n_1$, $n_2$, and $n_3$ be that used by 2DDynamicPack for all, narrow, medium-wide and wide items, respectively, i.e., $n \leq n_1 + n_2 + n_3$. When 2DDynamicPack opens the $n_1$-th bin for a new narrow item, the total area of all items is more than $\frac{n_1-1}{3}$ (Lemma 4 (i)). Hence, we have $OPT \geq \lfloor \frac{n_1-1}{3} \rfloor + 1 \geq \frac{n_1}{3}$. When 2DDynamicPack opens the $n_2$-th bin for a new medium-wide item, by Lemma 4 (ii), the total height of all items is more than $n_2 - 1$. Since the width of these items is more than $\frac{1}{3}$, in any packing of the items, every horizontal line drawn intercepts at most two items and hence, the total number of bins used is at least $\lfloor \frac{n_2-1}{2} \rfloor + 1$, i.e., $OPT \geq \frac{n_2}{2}$. Finally, for wide items, using FFM means $OPT \geq \frac{n_3}{2.788}$. In total, $n \leq n_1 + n_2 + n_3 \leq 7.788 OPT$ and the lemma follows.

### 3.2 Upper Bounds for 3-D Dynamic Bin Packing

Recall that the three dimensions are $x_1$, $x_2$ and $x_3$ and the length of an item $R$ along dimension $x_i$ is denoted as $x_i(R)$. Our algorithm, called 3DDynamicPack, classifies the items into four classes according to $x_i$ of the items. An item $R$ is said to be in

- **Class-1** if $x_1(R) > \frac{1}{2}$;

– **Class-2** if $x_1(R) \leq \frac{1}{2}$ and $x_2(R) > \frac{1}{2}$;
– **Class-3** if $x_1(R) \leq \frac{1}{2}$ and $\frac{1}{3} < x_2(R) \leq \frac{1}{2}$; and
– **Class-4** if $x_1(R) \leq \frac{1}{2}$ and $x_2(R) \leq \frac{1}{3}$.

Classes 1 and 2 can be handled rather straightforwardly by using 2-D packing algorithm (details to be given later). Classes 3 and 4 need more attention. We describe two repacking procedures for handling these two classes.

**Procedure 3DRepackClass3: for items with $x_1(R) \leq \frac{1}{2}$ and $\frac{1}{3} < x_2(R) \leq \frac{1}{2}$.** Similar to an observation made in 2DRepackMedium, in any packing including the optimal one, at most two items can be packed side by side along dimension $x_2$. We divide a bin into two equal partitions along dimension $x_2$ both with length $\frac{1}{2}$ (the length along dimension $x_1$ and $x_3$ remains 1). When a new item $R$ arrives, we reserve the first partition for $R$ and check if existing items can be packed into the second partition: project existing items along dimension $x_2$, repack them by Lemma 3, treating them as rectangles with dimension $x_1$ as width and $x_3$ as height. If repacking is feasible, the packing configuration is returned as solution.

**Procedure 3DRepackClass4: for items with $x_1(R) \leq \frac{1}{2}$ and $x_2(R) \leq \frac{1}{3}$.** We first sort the existing items in descending order of $x_1(R)$. Items are then packed into layers constructed along dimension $x_1$, in an NFDH manner (next-fit-decreasing-height). The *first layer* has length $\frac{1}{2}$ along dimension $x_1$. Project the items along dimension $x_1$ and treating $x_2$ as width $x_3$ as height, we pack items into this layer using 2DRepackNarrow, i.e., create two partitions with width $\frac{1}{3}$ and $\frac{2}{3}$, and pack to the larger partition. If some item $Q$ cannot be packed into this layer, we then create *a new layer* with length $x_1(Q)$ along dimension $x_1$ and pack $Q$ into the first partition of this layer. Then we use 2DRepackNarrow similarly to pack items to the second partition. The item that overflows from a layer will be packed into the first partition of the next layer. Repeat this until all existing items are packed or a new layer overflows dimension $x_1$ of the bin. If repacking is feasible, the packing configuration is returned as solution.

**Algorithm 3DDynamicPack.** Classify the items as they arrive into the four classes defined above. Items in each class are assigned to bins independently of other classes.

– **Class-1:** Project the items along dimension $x_1$ treating them as rectangles, and then pack the items using Algorithm 2DDynamicPack.
– **Class-2:** Project each item along dimension $x_2$ and further classify the items into two sub-classes with $0 < x_1 \leq \frac{1}{3}$ and $\frac{1}{3} < x_1 \leq \frac{1}{2}$. For items of the first sub-class, find a bin such that existing items can be repacked using 2DRepackNarrow. If there is no such bin, open a new bin for the new item. Similarly for the second sub-class, use 2DRepackMedium.
– **Class-3:** When a new item $R$ of Class-3 arrives, find any bin that existing items can be repacked using procedure 3DRepackClass3 into the second partition and pack $R$ into the first partition. If there is no such bin, open a new bin for $R$.
– **Class-4:** When a new item $R$ of Class-4 arrives, find any bin that existing items can be repacked using procedure 3DRepackClass4 and pack $R$ into the reserved space in the first layer. If there is no such bin, open a new bin for $R$.

**Theorem 3.** 3DDynamicPack *is* 22.788*-competitive.*

*Proof.* Let $OPT$ be the maximum number of bins used by the optimal offline algorithm, $n$, $n_1$, $n_2$, $n_3$, and $n_4$ be that by 3DDynamicPack for all, Classes 1, 2, 3 and 4 items, respectively, i.e., $n \leq n_1 + n_2 + n_3 + n_4$. When 3DDynamicPack opens the $n_1$-th bin for a new Class-1 item, by Theorem 2, we have $OPT \geq \frac{n_1}{7.788}$.

Let $n_{2,1}$ and $n_{2,2}$ be the maximum number of bins used for the two sub-classes of Class-2 items, i.e., $n_2 \leq n_{2,1} + n_{2,2}$. By Lemma 4 (i) and (ii) and a similar argument as Theorem 2, we have $OPT \geq \frac{n_{2,1}}{3}$ and $OPT \geq \frac{n_{2,2}}{2}$. So, $OPT \geq \frac{n_2}{5}$.

When 3DDynamicPack opens the $n_3$-th bin for a new Class-3 item, by Lemma 3, the total area of all Class-3 items is more than $\frac{n_3-1}{2}$. Furthermore, the length of these items along dimension $x_2$ is more than $\frac{1}{3}$, in any packing of the items, every line drawn parallel to dimension $x_2$ intercepts at most two items. Hence, $OPT \geq \lfloor \frac{n_3-1}{4} \rfloor + 1 \geq \frac{n_3}{4}$.

When 3DDynamicPack opens the $n_4$-th bin for a new Class-4 item, by Lemma 4 (i), in the repacking of each bin, the total area of all Class-4 items in the second partition in each layer and the first partition in the next layer is more than $\frac{1}{3}$. Furthermore, the height of all items in each layer is at least the height of the next layer. Since existing items cannot be repacked into the same bin using the procedure 2DRepackNarrow, the total height of all layers is more than 1 and that of all but the first layer is more than $\frac{1}{2}$ (since the first layer has height $\frac{1}{2}$). Therefore, the total volume of existing items in each bin is more than $\frac{1}{6}$ and the total volume of all existing items is more than $\frac{n_4-1}{6}$. Hence, $OPT \geq \lfloor \frac{n_4-1}{6} \rfloor + 1 \geq \frac{n_4}{6}$. In summary, we have $n \leq n_1 + n_2 + n_3 + n_4 \leq 22.788OPT$.

## 4 Concluding Remarks

In this paper, we have studied multi-dimensional dynamic bin packing. We have presented a general competitive ratio for $d \geq 2$ and improved the ratio further for $d = 2$ and $d = 3$. So far the competitive ratio for multi-dimensional bin packing (both static and dynamic, as well as for both cube and box) grows exponentially with $d$. Yet there is no matching lower bound that also grows exponentially with $d$. It is believed that this is the case [8] and any such lower bound would be of great interest. Furthermore, the general upper bound for $d$-dimension is usually worse than the corresponding 2-D or 3-D upper bound when substituting $d = 2$ or $d = 3$. It would be desirable to have $d$-dimensional packing algorithm that have a more accurate formula to reflect the ratio for lower dimension. As for lower dimension, an obvious open question is to close the gap between the upper bound and lower bound.

Another direction is to consider resource augmentation in which the online algorithm can use $d$-dimensional bins of size $s_1 \times s_2 \times \cdots \times s_d$ for $s_i \geq 1$ while the optimal offline algorithm uses unit-sized bins. As a first step, we give some simple conditions for the online algorithm to match the performance of the optimal offline algorithm, i.e., 1-competitive. The results here are obtained directly from those in Sections 2 and 3.

**Corollary 1.** *Consider the optimal offline algorithm that uses unit sized bins.*
*(i) For 2-D, there is a 1-competitive online algorithm using bins of size* $3 \times 1$.
*(ii) For d-D, there is a 1-competitive online algorithm using bins of size* $\{2\}^d$.
*(iii) For d-D, no online algorithm is 1-competitive using bins of size* $\{2-\epsilon\}^d$ *for any* $\epsilon > 0$.

## References


1. N. Bansal, A. Caprara and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *FOCS 2006*, pages 697–708, 2006.
2. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. W. T. Chan, T. W. Lam, and P. W. H. Wong. Dynamic bin packing of unit fractions items. *Theoretical Computer Science (TCS)*, 409(3):521–529, 2008.
4. W. T. Chan, P. W. H. Wong, and F .C. C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis. *Algorithmica*, 53(2):172–206, 2009.
5. E. G. Coffman, Jr., G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, 1998.
6. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12(2):227–258, 1983.
7. E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, PWS Publishing, pages 46–93, 1996.
8. D. Coppersmith, P. Raghavan Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Operations Research Letters*, 8(1):17–20, 1989.
9. J. Csirik and G. J. Woeginger. On-line packing and covering problems. In A. Fiat and G. J. Woeginger, editors, *On-line Algorithms—The State of the Art*, pages 147–177. Springer. 1996.
10. J. Csirik and G. J. Woeginger. Resource augmentation for online bounded space bin packing. *Journal of Algorithms*, 44(2):308–320, 2002.
11. L. Epstein and M. Levy. Dynamic multi-dimensional bin packing. Manuscript. 2006.
12. L. Epstein and R. van Stee. Online bin packing with resource augmentation. *Discrete Optimization* 4(3-4): 322-333, 2007.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
14. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *J. of ACM*, 47(4):617–643, 2000.
15. Y. Kohayakawa, F. K. Miyazawa, P. Raghavan and Y. Wakabayashi. Multidimensional cube packing. *Algorithmica*, 40(3):173–187, 2004.
16. A. Meir and L. Moser. On packing of squares and cubes. *J. Comb. Theory Series A*, 5(2):116–127, 1968.
17. I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. In *ESA*, pages 290–299, 1994.
18. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.
19. P. W. H. Wong, and F .C. C. Yung. A 2.666-lower bound on dynamic bin packing. Manuscript.