An online packing heuristic for the three-dimensional container loading problem in dynamic environments and the Physical Internet

Abstract. In this paper, we consider the online three-dimensional container loading problem. We develop a novel online packing algorithm to solve the three-dimensional bin packing problem in the online case where items are not know well in advance and they have to be packed in real-time when they arrive. This is relevant in many real-world scenarios such as automated cargo loading in warehouses. This is also relevant in the new logistics model of Physical Internet. The effectiveness of the online packing heuristic is evaluated on a set of generated data. The experimental results show that the algorithm could solve the 3D container loading problems in online fashion and is competitive against other algorithms both in the terms of running time, space utilization and number of bins.

Keywords: Dynamic optimization, online optimization, dynamic environments, 3D Bin packing problem, 3D container loading problem; online packing heuristic, Physical Internet, Benchmark problems.

1 Introduction

The classic three-dimensional bin packing problem (3D-BPP) is a strong NP-hard combinatorial optimization problem [1, 2], where the primary aim is to load a finite number of items of different sizes using the smallest possible number of bins. In logistics and supply chains, the 3D-BPP is also called the three-dimensional container loading problem (3D-CLP). It has many industrial and commercial applications, such as loading goods to containers and pallets, cargo and ship stowage loading, etc. 3D-CLP has been studied extensively by a lot of researchers with different objective functions and constraints by using diverse methods as surveyed and discussed in [3, 4].

Although many studies have addressed the 3D-CLP, most have focused exclusively on volume utilization and ignored other practical requirements. In real world problems, a number of practical constraints and requirements need to be satisfied, such as loading feasibility, stability, weight balance, operational safety product handling, and the prevention of cargo damage during container shipping. But only a few works have addressed those mentioned requirements [5-7].

Furthermore, most existing works focused on the static (offline) multidimensional CLP. Only few works have addressed the online or dynamic 3D-CLP where knowledge about items' arrival is not known in advance and items have to be packed right after they arrive. The online 3D-CLP has many applications in automatic or robotic cargo loading in warehouse storages. It will also become very common in a new logistics

adfa, p. 1, 2011.

model, the Physical Internet[1], which is considered the future of smarter logistics[2]. This logistics model proposes to move, pack and unpack goods in the same way as information are being transported, pack and unpack in the digital internet. In the Physical Internet logistics model, items will arrive in real-time with not much notification, and they will need to be packed immediately to avoid any delay.

To solve the online/dynamic 3D-CLP, one needs to follow a dynamic optimization approach, in which the problem is solved online when time goes by [8], i.e. packing solutions need to be provided immediately in real-time whenever one or a set of items arrive.

The classical online one-dimensional CLP problem was introduced in [9-11], where items are coming one by one and each must be packed immediately and irrevocably into a bin without any knowledge of future items and the goal is to minimize the maximal number of bins ever used over all times.

The issue we are addressing in this work is to develop an online packing heuristic for 3D-CLP with online arrival of items, and test its performance against other online and static algorithms. The algorithm must make decisions immediately and irrevocably based only on a part of the input without any knowledge of the future part. This is different to the static (offline) case where an algorithm knows the whole information about items and containers before making any decision. The algorithm should guarantee that all items are loaded more realistically in the containers, with one door from the one side, and avoid the problem of blocked items (see

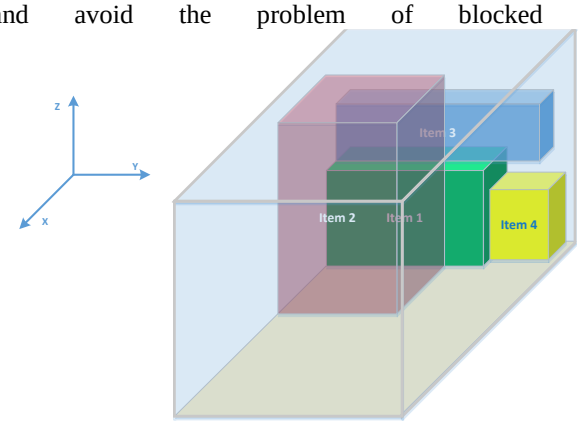

Fig. 3).

The remainder of this paper is organized as follows. In section 2 basic concepts of online 3D-BPP are introduced. In section 3 we discuss about building a packing heuristic for the online case. Section 4 discusses generating problem data and testing the performance of algorithms. Finally, conclusions are given in section 5.

---

[1] B. Montreuil. Toward a Physical Internet: meeting the global logistics sustainability grand challenge. Logistics Research, 3(2):71-87, 2011.

[2] ALICE, "Global Supply Network Coordination and Collaboration research & innovation roadmap," ALICE - Alliance for Logistics Innovation through Collaboration in Europe, 2014.

2 Problem definition

Since the online 3D-CLP in this work is formed from the classical 3D-CLP, it is defined as follows: suppose that IPS is an item packing sequence where item are coming in an online fashion, CLS is a container loading sequence and there are enough containers for the whole IPS.

Let N be the total number of items, and let M be the total number of user containers over all the time of loading process. We assume that all items and containers have the shape of a cube, where the length, width, and height of a container are oriented with the x-axis, y-axis and z-axis respectively in the Cartesian coordinate system. So the point $(0,0,0)$ is the deepest-bottom-left corner of a container. Let L_j, W_j, H_j and c_i be the container length, width, height and the cost of the j-th container, respectively. Let the lowercase letters l_i, w_i, h_i be the length, width, and height of i-th item.

2.1 Objective and constraints

The objective is to minimize the total cost of all used containers:

$$\sum_{i=1}^{M} c_i \rightarrow min \tag{1}$$

If the containers are homogeneous, then the cost of all containers are the same. In this case the objective is to minimize M, the number of containers (bins) to be used. This objective function is similar to maximizing the utilization or minimizing the wasted space:

$$\sum_{i=1}^{M} (L_j * W_j * H_j) - \sum_{i=1}^{M} (l_j * w_j * h_j) \rightarrow min \tag{2}$$

For the constraints of the online problem, this work takes into account the basic geometric constraints, according to [3]: (1) the items are assumed to be placed orthogonally, that is, the edges of the boxes have to be either parallel or perpendicular to those of the containers and within the container's dimension; (2) all items are packed and (3) items do not overlap with each other. More detailed explanation of the similar model can be found in [6, 12, 13]. We also consider a specific constraint that the items can be rotated because it is an important operational factor in both research and the real world. There are, at most, six different rotation types (0 to 5) for each packed item in a container. They are: l-w-h, w-l-h, l-h-w, w-h-l, h-l-w, and h-w-l (see **Fig. 1**).

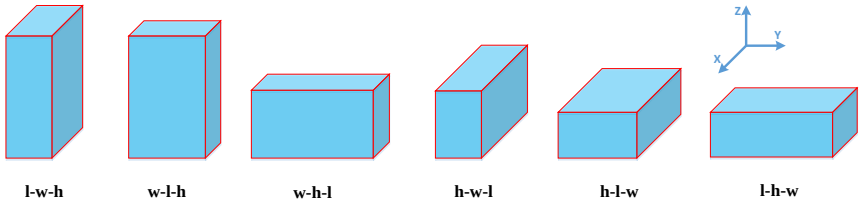

Fig. 1. Six types of item orientation

2.2 Empty maximal spaces

To indicate the exact position where an item can be loaded into a particular container, many concept of coordinates of objects have been used, for example the concept of corner points [14-16] and the concept of extreme points [17, 18]. In this work, we use the concept of empty maximal spaces (EMSs) to represent the free spaces in bins. In this concept, for each container to be used, we list the largest empty orthogonal spaces (that are not inscribed by any other space) available for packing. This concept is used in many recent studies [19-21]. Each empty maximal space is represented by a pair of their vertices with minimum and maximum coordinates: deepest bottom left vertex and highest top right vertex. **Fig. 2** shows four empty maximal spaces in a container where item1 is placed in the middle front of the container. The difference process introduced by [22] is also used in this work to calculate and update the list of EMSs.

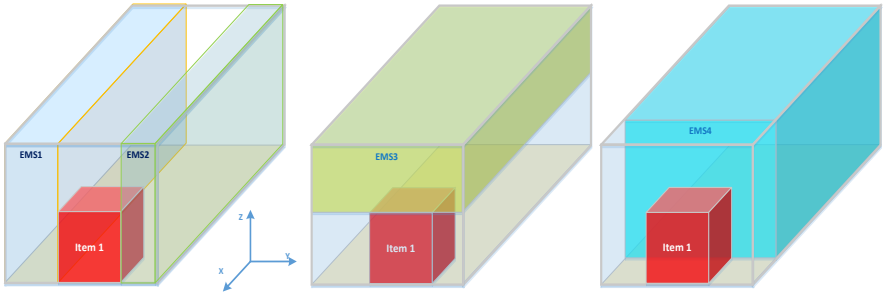

Fig. 2. Empty maximal spaces

3 Online packing heuristic

For the static 3D-CLP, usually, a packing algorithm follows a certain type of heuristic packing strategy. The input of a packing algorithm includes a sequence of packing items (IPS), which can be static or online, and a sequence of loading containers (CLS). The packing algorithm will then convert these sequences into a solution, where each item is assigned an exact position and an exact orientation inside a container. Depending on the type of 3D-CLP and the priority of selection of items and positions, one of many available packing heuristics can be used, such as First Fit Decreasing Algorithm (FFDA), Best-Fit Decreasing Algorithm (BFDA) [23], Bottom-Left-Fill, Bottom-Left-Back-Fill [24], etc.

[25] introduced a packing strategy called Deepest-Bottom-Left with Fill heuristic (DBLF). This heuristic has gained its popularity in various works e.g. [6, 25, 26]. This heuristic always searches for a space with the minimal x (deepest) coordinate to place the current item. And it uses z (bottom) and y (left) coordinates as tie breakers. This heuristic is also used and modified in [27]. [13] showed two drawbacks of DBLF: First, only one coordinate (x) plays the dominant role in choosing the candidate space; second, only one of the two factors: the item or the space, is determined. The other factor is then selected based on certain priority rules. This results in a potential loss of good

combinations that may lead to better solutions (i.e. a solution with a smaller wasted space); In [13], authors proposed a new packing heuristic called Best Match First Packing Heuristic (BMF). In this heuristic, EMSs are sorted in order of smallest coordinate values of the vertices to the deepest-bottom-left point of the container (with coordinates $(x, y, z) = (0,0,0)$). Then BMF searches for the best combination (space-box-orientation), in which the space as close as possible to the deepest-bottom-left corner of the container is chosen to place the current item. The authors showed that BMF outperforms DBLF in terms of utilization when they are combined with a metaheuristic such as GA or DE.

We find that the main difference between the two heuristics is the priority of EMSs, but in both BMF and DBLF, the problem of items being blocked, i.e. an item cannot be loaded through the container's entrance door to its designated locations due to other existing items blocking its way, is not considered. In subsection 3.1 we will propose an online packing heuristic (OnlineBPH) for the online 3D-BPP. The OnlineBPH is inspired from the Deepest-Bottom-Left order in DBLF and the idea of choosing the best combination (space-item-orientation) in BMF. In this new algorithm also we implemented an improvement in space selection to avoid the problem of item being blocked, so that the loading solution provide by packing heuristics can be more realistic and implementable in the real-world.

3.1 Empty maximal space selection

For two EMSs in the same container, we first compare their deepest (the x-value) coordinate values of the two vertices, while the EMS with the smaller value is given higher priority. If they are at the same deep level, we compare the bottom (the z-value) coordinate values and assign the higher priority to the EMS with the smaller one. If the two values are still the same, we compare the y-values. Furthermore, the heuristic always check if the item can be loaded from the door of the container or not by checking the x-value of the selected EMS. The reason behind this is to avoid the blocking problem, where an item can be fit in the space, but the loading process is unfeasible. When the deepest-bottom-left space is selected for an item, but the length dimension of this EMS does not extend to the entrance of the container, then other loaded items can block its loading ways. For example, in **Fig. 3** if the items are loaded in order item1, item2, item3 then item4, then after item2 is loaded, item 3 cannot be placed in the assigned space, because item 2 has blocked its way.

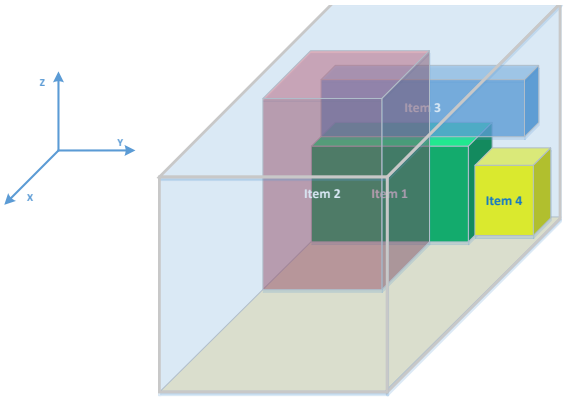

Fig. 3. The blocking problem during loading

3.2 Placement selection

OnlineBPH also inherits two parameters *Kb* and *Ke* from [13] but in a different manner: to determine the placement assignment, for Kb items (if items are arriving one by one then *Kb=1*) and the first *Ke* EMSs in the each opened container, the heuristic finds all the feasible placement assignments with allowed rotations of the items. When one item has several feasible placements in a free EMS, the one that has the smallest margin (from the faces of item to the faces of EMS) is selected. The triad (item, rotation, ems) pair with the largest fill ratio is chosen.

The three main differences between our online packing heuristic (OnlineBPH) and BMF are (1) BMF has information about all items while OnlineBPH only has information about the limited number of items that have arrived. (2) BMF will place an item in the ***first opened container (bin)*** that it has found a suitable space; while OnlineBPH considers all combinations of (bin, item, ems, and rotation) in ***all opened containers*** first before deciding which one is the best. (3) the OnlineBPH always check either the item can be loaded from the door of the container or not by checking the x-value of the selected EMS.

3.3 Pseudo-code of OnlineBPH

Pseudo-code of OnlineBPH is described in **Fig. 4**. At each step OnlineBPH decides packing information of one item in sequence (item will be packed in which container, at which position and in which orientation). For this, there are two phases to determine a placement of one item. First, the algorithm considers all opened containers and the information about the first Ke empty maximal spaces in $EMSs$ list of each opened container, and look ahead Kb item in online IPS Then it selects the best tetrad of (*container, item, orientation, ems)* to pack an item. If there are no feasible tetrad, the second phase is triggered to open a new suitable container for the item. $FindNewSuitableContainer$ returns the container that can fit the item with the largest fill ratio. In the case of identical containers, the next empty container in *CLS* is

chosen. Note that the initial EMS of a container cover its whole space, so containers with different dimensions will have initial EMSs with different sizes.

Input: An online item packing sequence IPS and a container loading sequence CLS;
Output: $Packing\ solution\ PS\ for\ IPS$ or **null** if not found;

```
1.  Let OC be the list of opened containers;
2.  OC ← Ø; PS ← Ø;
3.  while items are arriving or IPS ≠ null do
4.      Let P be a queue of candidate placements;
5.      Kb ← numbers of arrived items;
6.      Update IPS;
7.      itemplaced ← 0;
        // Phase 1: try to put an item to an opened container at the DBL ems;
8.      for each c ∈ OC do
9.          Let EMSc be the sorted list of the ems of container c in the
10.         deepest-bottom-left-first order
11.         j ← 0;
12.         while j < Ke and j < EMSc.size do
13.             for i ← 0 to Kb do
14.                 for each allowed rotation io of item i do
15.                     if IPS[i] can be placed in EMSc[j] with io and
                            is not blocked by others then
16.                         add this placement combination to P;
17.             j = j + 1;
18.     if P ≠ Ø then
19.         add the placement indicted by P[0] to PS;
20.         update IPS and EMS lists;
21.         itemplaced ← 1;
        // Phase 2: Open a new container to load current item to its DBL corner;
22.     if itemplaced = 0 then
23.         c = FindNewSuitableContainer(CLS, i);
24.         if c ≠ null then
25.             EMSc be the initial empty maximal spaces in c;
26.             for each allowed rotation io do
27.                 if IPS[i] can be placed in EMSc with orientation io then
28.                     add this placement combination to P;
29.             if P ≠ Ø then
30.                 move c from CLS to OC;
31.                 add the placement indicted by P[0] to PS;
32.                 update IPS and EMS lists;
33.                 itemplaced ← 1;
34.     if itemplaced = 0 then
35.         return null;
36. return Packing solution PS;
```

Fig. 4. Pseudo-code of packing heuristic for Online 3D-CLP

4 Experiments

4.1 Benchmark problems

Due to the lack of proper data for online 3D-CLP, we follow the approach described in [28] to generate test problems for the case of identical container packing problems. We generated 4 classes (I, II, III and IV) of instances. For classes I, II and III, specific distributions are chosen (Table 2), where l_j, w_j and h_j are length, with and height of j-th generated item, whereas L, W, H are respectively length, with and height of the identical containers.

For class IV, the following four types (types 1, 2, 3, and type 4) of uniformly distribution are defined in the terms of the length L, width W and height H of the containers (**Table 1**). To generate class IV, instances of type 1 are selected with probability 70%, instances of types 2, 3, 4 are selected with probability 10% each.

Table 1. Type of random items in instances

Type of uniformly distribution in dimensions of items	l_j	w_j	h_j
Type 1	$\left[1,\frac{1}{3}L\right]$;	$\left[\frac{2}{3}W,W\right]$;	$\left[1,\frac{1}{2}H\right]$;
Type 2	$\left[\frac{1}{2}L,L\right]$;	$\left[1,\frac{1}{2}W\right]$;	$\left[\frac{2}{3}H,H\right]$;
Type 3	$\left[1,\frac{1}{2}L\right]$;	$\left[\frac{1}{2}W,W\right]$;	$\left[\frac{1}{2}H,H\right]$;
Type 4	$\left[\frac{2}{3}L,L\right]$;	$\left[1,\frac{1}{2}W\right]$;	$\left[1,\frac{1}{2}H\right]$;

To evaluate the performance of algorithms, we classified instances into groups in terms of the problems sizes (number of items): small (less than 50 items), medium (from 50 up to 200 items), or large (more than 200 items). For classes I, II, and III, we generated for each class 5 datasets of instances in sizes of 20, 40 (small), 60, 80 (medium) and 1000 items (large). For class IV, we generated 2 datasets of small size (40 items) and large size (1000 items). In each dataset, there are 100 instances have generated. **Table 2** show the classes of test problems.

Table 2. Classes of test problems

Data set No.	Category	Sizes of containers (L*W*H)	No. of items	No. of instances	Item sizes (l*w*h)
I_20	Small	30*30*30	20	100	uniformly random in [1,10]
I_40	Small	30*30*30	40	100	uniformly random in [1,10]
I_60	Medium	30*30*30	60	100	uniformly random in [1,10]

I_80	Medium	30*30*30	80	100	uniformly random in [1,10]
I_1000	Large	30*30*30	1000	100	uniformly random in [1,10]
II_20	Small	100*100*100	20	100	uniformly random in [1,35]
II_40	Small	100*100*100	40	100	uniformly random in [1,35]
II_60	Medium	100*100*100	60	100	uniformly random in [1,35]
II_80	Medium	100*100*100	80	100	uniformly random in [1,35]
II_1000	Large	100*100*100	1000	100	uniformly random in [1,35]
III_20	Small	100*100*100	20	100	uniformly random in [1,100]
III_40	Small	100*100*100	40	100	uniformly random in [1,100]
III_60	Medium	100*100*100	60	100	uniformly random in [1,100]
III_80	Medium	100*100*100	80	100	uniformly random in [1,100]
III_1000	Large	100*100*100	1000	100	uniformly random in [1,100]
IV_40	Small	100*100*100	40	100	probability 70% of type 1, probability 10% of each types 2, 3, 4
IV_1000	Large	100*100*100	1000	100	probability 70% of type 1, probability 10% of each types 2, 3, 4

4.2 Computational results

All the proposed approach and algorithms have been coded in C and executed on a system with the following configuration: Intel® Core™ i5-4590 CPU @(3.30Ghz, 3.30Ghz) with 8.00 GB RAM, Windows 7 Enterprise 64-bit.

Table 3 and **Table 4** show the average results of OnlineBPH on 100 generated instances for each dataset. We tested the proposed algorithm for both cases of fixed orientation and free (6-ways) orientation of items. To evaluate efficiency of the approach, we selected different combination of parameters Kb and Ke.

Let S be the number of tetrads (*container, item, orientation, ems)*, obviously, S is proportional to $Kb * Ke * OC\ size * number\ of\ allowed\ orientations\ of\ item$. For $Kb = 3$, $Ke = 3$ or $Ke = 5$, the value of S is larger than in the case where $Kb = 1$ or $Ke = 1$. As shown in **Table 3** and **Table 4**, in the case of fixed orientation when $Kb = 3$ and $Ke = 3$ the algorithm gives better results, but in the case of six way orientations, with $Kb = 1$ and $Ke = 1$, the algorithm is more efficient both in utilization and computational time.

To compare the performance of OnlineBPH, we implemented three other algorithms from the literature:

- The online packing algorithm in [28] (Algorithm 1). This online heuristic is based on a layer-building approach;
- Algorithm864, proposed in [16] - a static approximation packing heuristic - where items are sorted by non-increasing volume. In this work the concept of corner points

and a branch & bound procedure are employed to verify whether a set of boxes can be placed into a container. The algorithm also assumes that unlimited identical bins are given.

- A static metaheuristic approach (DE+BMF), introduced in [13] - a differential evolution algorithm (DE) with the Best-Match-First Packing heuristic (BMF). As shown by the authors, this is one of the best combinations of a metaheuristic and a packing heuristic for static 3D container loading problem so far.

Table 3. OnlineBPH performance in case of fixed orientation of items

$(Kb = 1; Ke = 1)$			$(Kb = 1; Ke = 3)$			$(Kb = 3; Ke = 3)$			$(Kb = 3; Ke = 5)$		
Avg. No of bins	Avg. Utl.	Run-ning times	Avg. No of bins	Avg. Utl.	Running times (s)	Avg. No of bins	Avg. Utl.	Run-ning times	Avg. No of bins	Avg. Utl.	Running times (s)
1	**12.69%**	0.002	1	12.69%	0.0017	**1**	**12.69%**	**0.0015**	1	12.69%	0.0014
1	**24.49%**	0.011	1	24.49%	0.0103	**1**	**24.49%**	**0.0097**	1	24.49%	0.0095
1	**37.14%**	0.02	1	37.14%	0.0191	**1**	**37.14%**	**0.0209**	1	37.14%	0.0202
1.03	48.61%	0.03	1.03	48.61%	0.0295	1.02	48.96%	0.0308	**1.01**	**49.24%**	**0.031**
8.09	76.53%	0.28	8.12	76.26%	0.2785	8.05	76.89%	0.3059	**8.13**	**76.17%**	**0.2988**
1	**11.72%**	0.003	**1**	**11.72%**	0.0033	**1**	**11.72%**	0.003	1	11.72%	0.0026
1	**23.51%**	0.022	**1**	**23.51%**	0.0231	**1**	**23.51%**	0.024	1	23.51%	0.0233
1.01	34.34%	0.05	**1**	**34.55%**	0.0508	**1**	**34.55%**	0.0571	1.01	34.34%	0.058
1.03	45.73%	0.079	**1**	**46.65%**	0.1049	1.04	45.55%	0.0848	1.1	43.94%	0.0862
9.51	61.91%	1.051	9.65	61.12%	1.0277	**8.88**	**66.22%**	1.107	9.14	64.30%	1.1053
5.32	47.36%	0.001	5.48	45.95%	0.0002	5.28	47.83%	0.0003	**5.27**	**47.92%**	0.0003
9.54	53.26%	0.001	9.59	52.87%	0.001	**9.51**	**53.44%**	**0.0011**	9.53	53.27%	0.0011
13.56	57.17%	0.002	13.68	56.65%	0.002	**13.52**	**57.30%**	**0.0021**	13.59	57.00%	0.002
17.23	**59.15%**	0.003	17.39	58.57%	0.0023	17.25	59.07%	0.0029	17.29	58.97%	0.0028
173.6	**74.40%**	0.072	173.8	74.32%	0.0717	174.14	74.16%	0.1404	175	73.81%	0.1481
2.61	54.34%	0.009	2.6	54.51%	0.0091	**2.46**	**57.73%**	**0.0087**	2.49	57.03%	0.0088
44.74	76.17%	0.345	44.61	76.39%	0.3214	44.55	76.50%	0.4287	**44.44**	**76.69%**	0.414

Data set No.	Total No. of items	Sizes of containers (L*W*H)
I_20	20	30*30*30
I_40	40	30*30*30
I_60	60	30*30*30
I_80	80	30*30*30
I_1000	1000	30*30*30
II_20	20	100*100*100
II_40	40	100*100*100
II_60	60	100*100*100
II_80	80	100*100*100
II_1000	1000	100*100*100
III_20	20	100*100*100
III_40	40	100*100*100
III_60	60	100*100*100
III_80	80	100*100*100
III_1000	1000	100*100*100
IV_40	40	100*100*100
IV_1000	1000	100*100*100

Firstly, we tested OnlineBPH and Algorithm1 for all instances in an online manner. Then when all information about instances is gathered, the Algorithm864 and DE+BMF is applied to the test cases in an offline (static) manner. The parameters of DE are set as follows: G = 200; Np = 80; F = 0.85; Cr = 0.5; the parameters of BMF Kb = 3; Ke = 3 (as recommended by the authors). The results for 17 classes of instances are showed in **Table 5**. For each dataset (each has 100 instances), if an algorithm cannot solve more than 20 out of 100 instances then the algorithm is given N/A, i.e. no score. If an algorithm takes more than averagely 3600 seconds (1 hour) per instance, it is also given N/A. If an algorithm can solve more than 20 instances but not all 100 instances, the average value of utilization, number of used bins and average time is calculated for the solved cases only, and the scores are given in italic font.

Table 4. OnlineBPH performance in case of free (six ways) orientations of items

$(Kb = 1; Ke = 3)$			$(Kb = 3; Ke = 3)$			$(Kb = 3; Ke = 5)$		
Avg. No of bins	Avg. Utl.	Running times (s)	Avg. No of bins	Avg. Utl.	Running times (s)	Avg. No of bins	Avg. Utl.	Running times (s)
1	**12.69%**	0.002	1	12.69%	0.001	1	12.69%	0.001
1	**24.49%**	0.009	1	24.49%	0.009	1	24.49%	0.008
1	**37.14%**	0.02	1	37.14%	0.02	1	37.14%	0.02
1	**49.55%**	0.032	1	49.55%	0.036	1	49.55%	0.034
7.33	84.63%	0.335	**7.25**	**85.53%**	0.382	7.41	83.75%	0.378
1	**11.72%**	0.003	1	11.72%	0.003	1	11.72%	0.003
1	**23.51%**	0.026	1	23.51%	0.023	1	23.51%	0.023
1	**34.55%**	0.063	1	34.55%	0.066	1	34.55%	0.063
1	**46.65%**	0.105	1.04	45.64%	0.115	1.03	45.91%	0.113
10	58.99%	1.372	10.01	59.08%	1.693	**9.87**	**59.88%**	1.636
4.64	54.64%	4E-04	4.56	55.50%	0.001	4.61	54.96%	6E-04
8.26	61.68%	0.002	8.26	61.68%	0.002	8.3	61.39%	0.002
11.9	65.31%	0.003	11.91	65.14%	0.005	12.05	64.46%	0.004
15.2	67.21%	0.005	15.35	66.54%	0.007	15.37	66.40%	0.007
161	80.39%	0.241	161.66	79.89%	0.604	162.6	79.40%	0.64
2.7	52.68%	0.008	2.71	52.43%	0.007	2.99	47.06%	0.006
44.2	77.06%	0.333	46.23	73.73%	0.679	52	65.57%	0.645

Online Algorithm 1 [28]		OnlineBPH (free rotation)			Static Algorithm864 [16]			Static DE+BMF [13]		
Avg. Utl.	times (s)	Avg. No of bins	Avg. Utl.	times (s)	Avg. No of bins	Avg. Utl.	times (s)	Avg. No of bins	Avg. Utl.	times (s)
12.69%	0.001	1	12.69%	0.0014	1	12.69%	0.001	1	12.69%	8.3
23.93%	0.001	1	24.49%	0.0091	1	24.49%	0.001	1	24.49%	45.73
18.72%	0.001	1	37.14%	0.021	*1*	*N/A*	*0.001*	1	37.14%	106.7
24.77%	0.001	1	49.55%	0.033	N/A	N/A	N/A	1	49.55%	238.7
28.43%	0.001	7.26	85.42%	0.328	N/A	N/A	N/A	N/A	N/A	N/A
11.72%	0.001	1	11.72%	0.003	1	11.72%	0.001	1	11.72%	13.66
18.90%	0.001	1	23.51%	0.023	*1*	*N/A*	*0.001*	1	23.51%	100.6
17.13%	0.001	1	34.55%	0.061	*1*	*N/A*	*0.002*	1	34.55%	303.7
20.21%	0.001	1	46.65%	0.105	N/A	N/A	N/A	1	46.65%	478.7
23.18%	0.001	9.87	59.88%	1.636	N/A	N/A	N/A	N/A	N/A	N/A
31.40%	0.001	4.5	56.59%	0.001	*5.354*	*47.00%*	*0.005*	3.94	64.81%	5.69
31.90%	0.001	8.09	63.02%	0.002	*9.522*	*54.71%*	*0.005*	7.2	71.09%	16.45
32.45%	0.001	11.8	65.77%	0.003	*13.28*	*58.94%*	*0.005*	10.45	73.94%	30.05
32.53%	0.001	15.06	67.79%	0.005	*16.57*	*61.50%*	*0.006*	13.53	75.39%	44.26
32.67%	0.001	159.9	80.75%	0.227	162.5	79.48%	1.076	N/A	N/A	N/A
37.28%	0.001	2.54	55.89%	0.009	*3.103*	*47.53%*	*0.006*	2.14	65.55%	70.06
41.72%	0.001	42.55	80.10%	0.341	*44.32*	*76.89%*	*0.485*	N/A	N/A	N/A

Table 5. A comparison of dynamic Algorithm1, dynamic OnlineBPH, static Algorithm864 and static DE+BMF (N/A means no score due to less than 20 over 100 instances solved, or due to solving time greater than 3600 seconds)

Data set No.	Total No. of items	Sizes of containers (L*W*H)	$(Kb = 1; Ke = 1)$		
			Avg. No of bins	Avg. Utl.	Running times (s)
I_20	20	30*30*30	**1**	**12.69%**	0.001
I_40	40	30*30*30	**1**	**24.49%**	0.009
I_60	60	30*30*30	**1**	**37.14%**	0.021
I_80	80	30*30*30	**1**	**49.55%**	0.033
I_1000	1000	30*30*30	7.26	85.42%	0.328
II_20	20	100*100*100	**1**	**11.72%**	0.003
II_40	40	100*100*100	**1**	**23.51%**	0.023
II_60	60	100*100*100	**1**	**34.55%**	0.061
II_80	80	100*100*100	1.01	46.38%	0.103
II_1000	1000	100*100*100	10.23	57.91%	1.334
III_20	20	100*100*100	**4.5**	**56.59%**	0.001
III_40	40	100*100*100	**8.09**	**63.02%**	0.002
III_60	60	100*100*100	**11.8**	**65.77%**	0.003
III_80	80	100*100*100	**15.06**	**67.79%**	0.005
III_1000	1000	100*100*100	**159.9**	**80.75%**	0.227
IV_40	40	100*100*100	**2.54**	**55.89%**	0.009
IV_1000	1000	100*100*100	**42.55**	**80.10%**	0.341

Data set No.	Total No. of items	Sizes of containers (L*W*H)	Avg. No of bins
I_20	20	30*30*30	1
I_40	40	30*30*30	1.04
I_60	60	30*30*30	1.99
I_80	80	30*30*30	2
I_1000	1000	30*30*30	21.77
II_20	20	100*100*100	1
II_40	40	100*100*100	1.37
II_60	60	100*100*100	2.02
II_80	80	100*100*100	2.38
II_1000	1000	100*100*100	25.23
III_20	20	100*100*100	8
III_40	40	100*100*100	15.87
III_60	60	100*100*100	23.83
III_80	80	100*100*100	31.24
III_1000	1000	100*100*100	395.08
IV_40	40	100*100*100	3.76
IV_1000	1000	100*100*100	81.68

From **Table 5** we observe that in terms of computational time the Algorithm1 is the fastest, closely followed by OnlineBPH and the static Algorithm864. The static DE+BMF is significantly slower than the other three in magnitudes of thousands to hundreds of thousands. The static DE+BMF also takes more than 3600 seconds to solve the large instances with 1000 items (hence the N/A).

In the terms of solution quality (utilization or number of used bins), OnlineBPH and static DE+BMF achieved the best scores for problems of classes I and II (although DE+BMF failed to solve the largest cases with 1000 items). In problems of classes III and IV, DE+BMF is slightly better than OnlineBPH but again it failed in the largest cases while OnlineBPH still succeeded. Online Algorithm 1 performed the worst in terms of solution quality. Static Algorithm864 is bettern than online Algorithm 1, but it struggled to solve all the 100 instances in most datasets, failed to find solutions in some datasets, and its scores are generally worse than that of OnlineBPH and static DE+BMF.

Overall OnlineBPH seems to be the most well-rounded taking into account bother computational time and solution quality. It is generally the second-best in both categories and its scores are not far off the best scores and in many cases match the best scores. It is interesting to see that although it is expected that an optimal online solution cannot be as good as an optimal static/offline solution, OnlineBPH is actually just slightly worse than the best available static solutions (provided by DE+BMF). OnlineBPH's solutions are even better than the static solutions found by Algorithm864.

Here we will try to analyse the reason for the good/bad performance of the algorithms. OnlineBPH is fast because it is an online algorithm, being able to consider just one item at a time. OnlineBPH can produce solutions with good quality because (1) it considers all available containers and choose the most suitable for the current item; and (2) it utilize the EMS concept effectively by taking into account all feasible placements with all possible rotations.

Algorithm1 is fast because like OnlineBPH it is an online algorithm. Algorithm1 provides solutions with the worst quality because it is over simplified. Its layer-building approach is efficient only in the case of weakly heterogeneous items [4]. This is much

less effective than the mechanisms in OnlineBPH, Algorithm864 and DE+BMF. These three compute and consider a much larger number of placement combinations.

In most data sets, Algorithm864 cannot find the solutions for all instances because it does not allow the rotation of items. Due to that, if one of the items' original dimension exceeds the corresponding dimension of the container then algorithm will stop. Algorithm864 also trades the computational quality for computational time to make it fast. That is why a static algorithm like Algorithm864 can still be nearly as fast as online algorithms like Algorithm 1 and OnlineBPH. As a trade-off, the quality of Algorithm864 is worse than OnlineBPH, even that Algorithm864 is a static algorithm.

DE+BMF can provide the best results for the static case because it relies on one of the best packing heuristics, BMF, to pack items into a container, and it relies on an efficient meta-heuristics, DE, to find the optimal sequence of containers. The downside of DE+BMF is that it is very slow. Being a static algorithm it needs to consider all items before making a decision. In addition, the use of a population-based algorithm like DE also slow down the decision making process. The large amount of time needed for DE+BMF to find a solution in the large-scale cases (like the data sets with 1000 items) is simply not realistic in a real-world scenarios.

There is also another issue with Algorithm864 and DE+BMF: these algorithm do not check the problem of item being blocked, so their output may not be used directly for real loading process. Our experiments show that the solutions provided by these algorithms can have a large number of blocked items, meaning that not all items can be loaded into the containers in the sequence provided by the algorithms. This situation is mitigated by OnlineBPH because it always check either the item can be loaded from the door of the container first before selecting an EMS. Due to a lack of space we are not able to provide detailed experimental results on this issue, but this will be further investigated and published in a future publication.

In summary, OnlineBPH seems to be able to provide a good balance of time and utilization. Being an online algorithm it is obviously the only choice if items need to be handled/loaded in real-time or if there is no storage areas and/or buffers for incoming items. However, even in situations where items can be handled offline and there are ample storage areas for incoming items, OnlineBPH can still provide a good alternative to current state-of-the-arts static algorithms like DE+BMF. OnlineBPH is significantly faster; its solutions are just slightly less good in the tested cases; and it eliminates the need of having storage areas.

5 Conclusion

This work presented an online packing heuristic to solve the three-dimensional bin packing problem in dynamic environments. The effectiveness of the online packing heuristic is evaluated on a set of generated data. The experimental results show that the algorithm could solve the 3D container loading problems in online fashion and is competitive against the one of best static algorithms both in the terms of running time, space

utilization and number of bins. The algorithm also avoids the problem of blocked item and allows the loading process in the containers become more realistic.

Acknowledgement

This work is supported by a Newton Institutional Links grant funded by the British Council and a Newton Research Collaborations Programme (3) grant funded by the Royal Academy of Engineering.

The authors thank anonymous reviews for their suggestions and contributions and corresponding editor for his/her valuable efforts.

References

1. Anily, S., J. Bramel, and D. Simchi-Levi, *Worst-case analysis of heuristics for the bin packing problem with general cost structures.* Operations research, 1994. **42**(2): p. 287-298.
2. Scheithauer, G., *Algorithms for the container loading problem*, in *Operations Research Proceedings 1991*. 1992, Springer. p. 445-452.
3. Bortfeldt, A. and G. Wäscher, *Constraints in container loading–A state-of-the-art review.* European Journal of Operational Research, 2013. **229**(1): p. 1-20.
4. Zhao, X., et al., *A comparative review of 3D container loading algorithms.* International Transactions in Operational Research, 2016. **23**(1-2): p. 287-320.
5. Junqueira, L., R. Morabito, and D.S. Yamashita, *Three-dimensional container loading models with cargo stability and load bearing constraints.* Computers & Operations Research, 2012. **39**(1): p. 74-85.
6. Moon, I. and T.V.L. Nguyen, *Container packing problem with balance constraints.* OR Spectrum, 2014. **36**(4): p. 837-878.
7. Liu, D.S., et al., *On solving multiobjective bin packing problems using evolutionary particle swarm optimization.* European Journal of Operational Research, 2008. **190**(2): p. 357-382.
8. Nguyen, T.T., S. Yang, and J. Branke, *Evolutionary dynamic optimization: A survey of the state of the art.* Swarm and Evolutionary Computation, 2012. **6**: p. 1-24.
9. Berndt, S., K. Jansen, and K.-M. Klein, *Fully dynamic bin packing revisited.* arXiv preprint arXiv:1411.0960, 2014.
10. Coffman, J., Edward G, M.R. Garey, and D.S. Johnson, *Dynamic bin packing.* SIAM Journal on Computing, 1983. **12**(2): p. 227-258.
11. Epstein, L. and M. Levy, *Dynamic multi-dimensional bin packing.* Journal of Discrete Algorithms, 2010. **8**(4): p. 356-372.
12. Feng, X., I. Moon, and J. Shin, *Hybrid genetic algorithms for the three-dimensional multiple container packing problem.* Flexible Services and Manufacturing Journal, 2015. **27**(2-3): p. 451-477.

13. Li, X. and K. Zhang, *A hybrid differential evolution algorithm for multiple container loading problem with heterogeneous containers.* Computers & Industrial Engineering, 2015. **90**: p. 305-313.
14. Martello, S., D. Pisinger, and D. Vigo, *The three-dimensional bin packing problem.* Operations Research, 2000. **48**(2): p. 256-267.
15. Lodi, A., S. Martello, and D. Vigo, *Heuristic algorithms for the three-dimensional bin packing problem.* European Journal of Operational Research, 2002. **141**(2): p. 410-420.
16. Martello, S., et al., *Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem.* ACM Transactions on Mathematical Software (TOMS), 2007. **33**(1): p. 7.
17. Baldi, M.M., et al., *The generalized bin packing problem.* Transportation Research Part E: Logistics and Transportation Review, 2012. **48**(6): p. 1205-1220.
18. Crainic, T.G., G. Perboli, and R. Tadei, *Extreme point-based heuristics for three-dimensional bin packing.* Informs Journal on computing, 2008. **20**(3): p. 368-384.
19. Parreño, F., et al., *A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing.* Annals of Operations Research, 2010. **179**(1): p. 203-220.
20. Gonçalves, J.F. and M.G. Resende, *A biased random key genetic algorithm for 2D and 3D bin packing problems.* International Journal of Production Economics, 2013. **145**(2): p. 500-510.
21. Gonçalves, J.F. and M.G.C. Resende, *A parallel multi-population biased random-key genetic algorithm for a container loading problem.* Computers & Operations Research, 2012. **39**(2): p. 179-190.
22. Lai, K. and J.W. Chan, *Developing a simulated annealing algorithm for the cutting stock problem.* Computers & industrial engineering, 1997. **32**(1): p. 115-127.
23. Christensen, S.G. and D.M. Rousøe, *Container loading with multi-drop constraints.* International Transactions in Operational Research, 2009. **16**(6): p. 727-743.
24. Tiwari, S., G. Fadel, and P. Fenyes. *A fast and efficient compact packing algorithm for free-form objects*. in *ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. 2008. American Society of Mechanical Engineers.
25. Karabulut, K. and M.M. İnceoğlu. *A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method*. in *International Conference on Advances in Information Systems*. 2004. Springer.
26. Kang, K., I. Moon, and H. Wang, *A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem.* Applied Mathematics and Computation, 2012. **219**(3): p. 1287-1299.
27. Wang, H. and Y. Chen. *A hybrid genetic algorithm for 3d bin packing problems*. in *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*. 2010. IEEE.
28. Wang, R., T. T. Nguyen, S. Kavakeb. Z. Yang and C. Li. *Benchmarking Dynamic Three-Dimensional Bin Packing Problems Using Discrete-Event Simulation*. in

European Conference on the Applications of Evolutionary Computation. 2016. Springer.