Semi-online three-dimensional container loading problems

Hiroo Saito[1,a)] Yukio Asari[1,b)]

概要：We consider an online version of the three-dimensional container loading problem. In the online case sizes of the future items are not known in advance and items must be packed in the container right after their arrival. We introduce semi-online container loading problems relaxing one of the online conditions and propose deepest-bottom-left based Monte Carlo Tree Search and heuristics that sorts temporary buffered items, respectively. The computational results show that the algorithms improve the density (volume utilization) against the genuine online case.

1. Introduction

The container loading problem, also known as the three-dimensional packing problem, is a widely-studied NP-hard combinatorial optimization problem [5]. This problem is a modelling of packing small items into a container without overlap as much as possible and there are many applications in logistics and supply chains. There are numerous problem variants (see a survey [13]) for practical application such as item orientation, item balance, weight, multiple container, priority of items, etc.

This paper deals with an online version of the single container loading problem for density (volume utilization) maximization. Such problem is worthwhile for the situation where real-time processing is necessary. The online container loading problem has the following conditions which are different from the ordinary static or offline one, and they make the problem difficult to optimize.

(1) Sizes of the future items are not known in advance.
(2) Items must be packed in the container right after their arrival.
(3) Each item arrives one by one.
(4) Skip or pass an item is prohibited (if we cannot pack the current arrived item in the container, then terminate).
(5) Items are irrevocable (we cannot move afterwards the items in the container once packed).

There are similar conditions in the online bin packing problem where the number of bins is minimized [4]. The paper [7] deals with the multiple container case and proposed a heuristics approach. They proposed an empty maximal spaces based heuristics choosing a container among the containers that has the largest fill ratio. However, it is not trivial to apply their approach because we deal with the single container case here.

We propose *semi-online container loading problems* relaxing one of the conditions (1) and (2). More precisely, we consider the problem where one of the following conditions holds.

(1') Sizes of the future items are known in advance (let k be the number of the known future items),
(2') There is a buffer which temporary stores small number of input items (let b be the capacity of the buffer).

Thanks to these relaxed conditions, searching packing positions according to the future input item sizes or choosing an item from the buffer is possible which would be helpful to improve density.

Figure 1 is an illustration of the online and semi-online container loading problems where n is the number of input items. We define the semi-online problem $P(k, b)$, the precise definition is given in the next section, and $P(0, 0)$ becomes the (genuine) online problem. In particular, we deal with the following two problems in this paper.

[1] Infrastructure Systems and Development Center, Toshiba Infrastructure Systems & Solutions Corporation, 1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 212-8581, Japan
a) hiroo4.saito@toshiba.co.jp
b) yukio.asari@toshiba.co.jp

(1) The problem $P(n-1,0)$: every input size is known in advance and the buffer is not available, namely $k=n-1$ and $b=0$.
(2) The problem $P(0,b)$: the future input sizes are not known in advance and the buffer is available, though its capacity is limited, namely $k=0$ and $b \geq 1$.

We propose a Monte Carlo Tree Search (MCTS) based heuristics for $P(n-1,0)$ and an online version of deepest-bottom-left position based heuristics for $P(0,b)$, respectively. And we show that the relaxed conditions improve density by computational results.

The rest of this paper is organized as follow. In Section 2 we describe our problems in detail. In Section 3 we propose a Monte Carlo Tree Search and an online version of deepest-bottom-left position based heuristics. In Section 4 we show a computational experiments of our algorithms for data randomly generated from an actual logistics items. Finally, we give conclusions in Section 5.

2. Problem description

We define a semi-online container loading problem $P(k,b)$, which is an extension of the online one.

Let (w_i, l_i, h_i) be the size of an item $i = 1, 2, \ldots, n$, and (W, L, H) be the size of the single container. The shapes of item and container are cube. We put items in the container without overlap and within the container. And the objective is to maximize their density (volume utilization). We assume that each item is provided one by one from an item set whose output will be input of our system.

It is noteworthy that our system has buffer which is a storage for items and is a relatively small space to temporary store the input items. Our system also has the size data list which stores item size data. Given a nonnegative integer b for the capacity of the buffer and a nonnegative integer k for the length of the list of future input size data, we describe our problem in the form of procedure as follows.

Online container loading problem $P(k,b)$:

```
Step 1:Initialize the buffer $B := \emptyset$ and the size data $S := \emptyset$.
Step 2:If the item set is empty, then goto Step 6.
Step 3:Receive an input item $i$.
Step 4:Get sizes of the current and the following $k$ items ($n-i$ items when $i \geq n-k$) to update the size data $S := S \cup \{(w_i, l_i, h_i), \ldots, (w_{i+k}, l_{i+k}, h_{i+k})\}$.
Step 5:If the number of items in the buffer is not full ($|B| < b$), then put the item in the buffer $B := B \cup \{i\}$ and goto Step 2.
Step 6:Choose an item $j$ from the buffer and the current one ($j \in B \cup \{i\}$) and pack it in the container by an algorithm using the size data $S$. If such a solution does not exist, then exit.
Step 7:Update the buffer $B := B \cup \{i\} \setminus \{j\}$ and goto Step 2 for the next item.
```

Note that we cannot skip any item. More precisely, each item must be packed into a container and terminate unless we can, as in Step 6. We also remark that the order of items is fixed or limited due to the buffer constraints. Hence, a technique such as sorting items with respect to their sizes, which is a common heuristics in static container loading problem, is not always applicable. For example, we have to pack an item immediately when there is no buffer ($b=0$). We assume that the size of the buffer (not the capacity b but its volume) is large enough to store any b items.

The problem $P(k,b)$ contains several situations. For example, $P(n-1,n)$ and $P(0,0)$ are considered as the static and online container loading problems, respectively. We deal with the problem $P(n-1,0)$ and $P(0,b)$. When these relaxed problems are acceptable in actual application, we have an improve of density.

3. Algorithms

We propose algorithms to choose an item and pack it in the container in Step 6 of the problem $P(b,k)$ in the previous section.

Since our algorithms are based on the well-known deepest-bottom-left (DBL) strategy [9], [10], [12], which is an extension of two dimensional bottom-left strategy [3], we give a brief description of it. Figure 2 is our coordinate system where a corner of the container is located at the origin. We define the position of an item by its corner closest to the origin. Let positions of an item $p = (x, y, z)$ and $q = (x', y', z')$ and p is called more deep-bottom-left than q if $(z < z')$ or $(z = z', y < y')$ or $(z = z', y = y', x < x')$. We applied a DBL strategy based heuristics that locates input items one by one at its deepest-bottom-left position without overlapping the already packed items.

Given a newly arrived item, we enumerate the set of positions to put the item against the already packed items in the container. We use an algorithm to enumerate stable BL points in the two dimensional rectangle packing problem [8]. We use the two-dimensional algorithm layer by layer. The procedure first tries to put the item on the ground of the container projecting the already packed

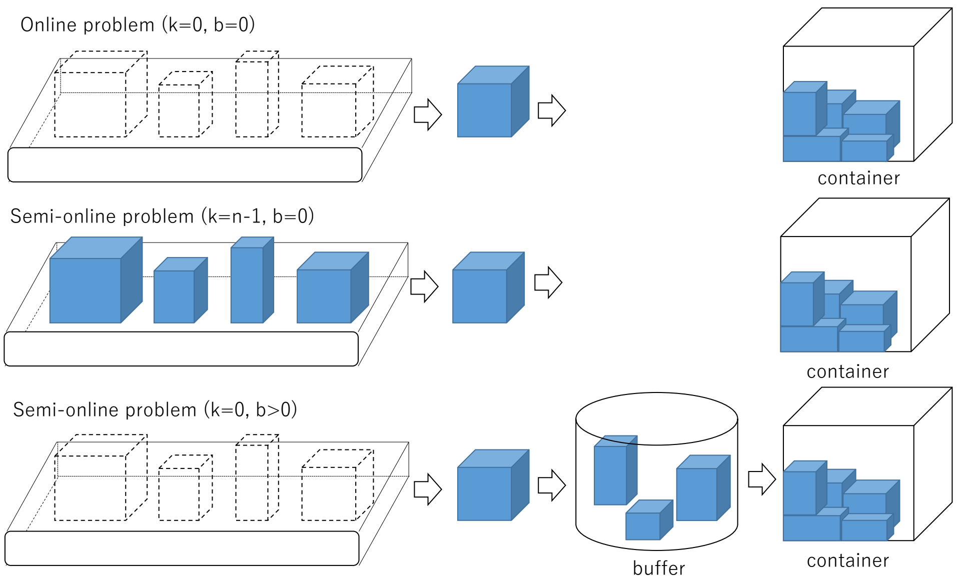
Online problem (k=0, b=0)
container
Semi-online problem (k=n-1, b=0)
container
Semi-online problem (k=0, b>0)
buffer
container

図 **1** Illustration of the online and semi-online container loading problems (sizes of the items with dotted lines are unknown)

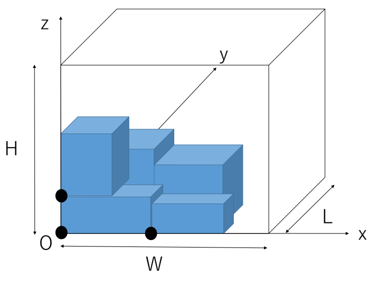

図 **2** A container and items in the coordinate system (a black dot is the position of an item)

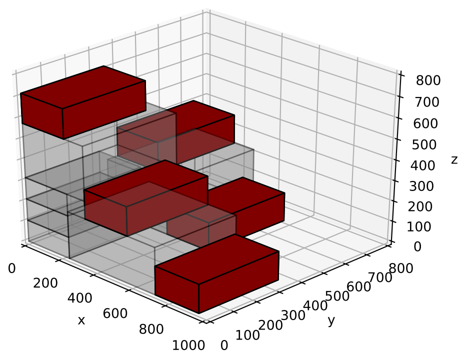

図 **3** Item positions by the enumeration algorithm

items on the ground to reduce a three-dimensional problem to a two-dimensional problem. Figure 3 is an example of our positions, where the five possible positions of the item is illustrated by the red cube against the already packed six items. We choose the deepest-bottom-left position among the enumerated positions in our algorithm.

We also adopt a simple stability check that stacking an item on the already packed items is possible when its overlap ratio is more than 0.5.

3.1 Monte Carlo Tree Search for $P(n-1,0)$

Since the problem $P(n-1,0)$ assumes that the future input item sizes are known in advance, it is possible to search position of each input item. Monte Carlo tree search (MCTS) is a search algorithm and there are many applications to games such as computer Go [6]. Since MCTS is based on random simulation called playout, it is easy to apply it to several problem. We use the UCT (the Upper Confidence Bound for Trees) algorithm [11] to our packing problem and node selection in a search tree is performed based on the UCB1 value [2]. This adaptation is straight-forward by the following correspondence: state is a packed items in the container, actions are positions of the newly arrived item against the already packed items, and reward is their density. We denote the number of iterations of the UCT for each item by T which controls the number of playout and is crucial for trade-off between density improvement and computational time.

Figure 4 is an example of a search tree of an instance of the computational results in Section 4. Note that this is a part of the tree because the total number of nodes is about 5,800. The root node is an empty container and the following children nodes are possible positions of an input item against its parent node (the already packed items).

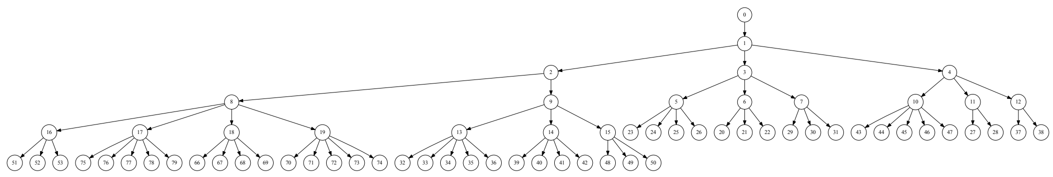

図 4 An example of a search tree

3.2 Online DBL heuristics for $P(0, b)$

Sorting items in advance is a well-known technique in the static container loading problem. Since the buffer is available in the semi-online problem $P(0, b)$, we give an online sorting procedure as follows.

DBL heuristics for Step 6 of $P(0, b)$:

Step 1:Sort the items in the buffer B. We denote the sorted items by $B = \{1, 2, \dots, b\}$.

Step 2:Initialize the index $i := 1$.

Step 3:If $i = b$, then exit.

Step 4:Find the deepest-bottom-left position for the item i in the container. If such position is not found, update $i := i + 1$ and goto Step 3.

Step 5:Output the item i with its deepest-bottom-left position and exit.

There are many keys for sorting items such as volume, length, width, height, random, etc. In this paper, we sort them with respect to their volume in descending order (large item comes first) based on preliminary computational experiments.

4. Computational results

The dataset is generated as follows. We chose eight cube items which are common in Japan [*1]. We set the container size $W = 1000$[mm], $L = 800$[mm], $H = 1500$[mm], and the frequency of each item in Table 1. We generated 100 instances each of which we randomly sampled $n = 100$ items according to the frequency.

We implemented our algorithms in C++. Computational experiments were conducted on a PC (Intel Xeon CPU E5-2680 v4 2.40GHz). We compared static version of deepest-bottom-left strategy (static BL), online version of deepest-bottom-left strategy without buffer (online BL), MCTS based search for the semi-online problem $P(n - 1, 0)$ (semi-online MCTS), and online version of deepest-bottom-left strategy with buffer (semi-online BL).

The performance metrics are density and computational time. The computational time for online problem is the worst one for each item. When an algorithm packs r-items one by one from a set of input items $i = 1, \dots, n$ and let $t_1, \dots, t_r$ be computational time for each item, the time is $\max\{t_1, \dots, t_r\}$. In static case, we take the total computation time $(t_1 + \cdots + t_r)$.

Item category	w[mm]	l[mm]	h[mm]	Frequency
Huge	345	445	340	0.05
Large	315	395	225	0.05
Medium	255	315	175	0.3
Small	175	255	145	0.3
Sake bottle single	135	135	445	0.1
Sake bottle double	135	265	445	0.05
Wine bottle single	110	110	315	0.1
Wine bottle double	110	210	315	0.05

表 1 Item sizes and their frequencies

Table 2 summarizes the results showing the average of density (density(m)), the standard deviation of density (density(sd)), and the average of computational time (time) for the 100 instances. We remark that the time for static case, (*) in Table 2, is the total of each item as described above.

First the density of online BL is worse than static BL. The average density degrades 78.0% to 50.9%. Since the static BL sorts the items before packing, sizes of the sequence of input items tends to be identical and the shape of packed items in the container becomes flat. This is effective for stacking items and we can pack items densely. The online case, however, such situation does not hold and the density degrades.

Using the future item information and the buffer is effective, respectively.

The average density of semi-online MCTS is improved to 56.0% (the UCT iteration is $T = 100$). We also remark that standard deviation of density is 6.6% and smaller than 11.4% of the online BL one. This shows that searching based on the future input is helpful to improve some cases where simple heuristics fails. The computation is very slow, however, it takes more than one second for each item. Though it is slow, it might be applicable when manipulation of robot arm takes few seconds, for example.

[*1] https://www.post.japanpost.jp/service/you_pack/package.html

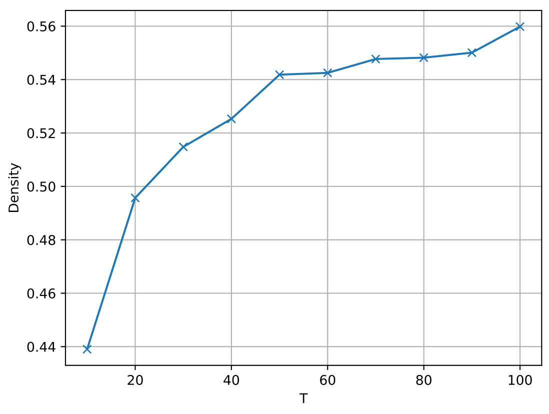

図 **5** UCT iteration T and density

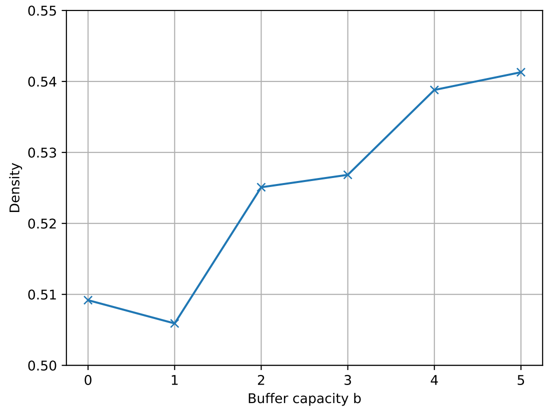

図 **6** Buffer capacity b and density

On the other side, the semi-online BL using the buffer whose capacity is $b = 5$ improves the density to 54.1% and the computation is fast. The demerit of this approach is that it requires extra physical space to store items in actual application.

Figure 5 is the UCT iteration T and density. The density increases as the T increases saturating around $T = 50$. When T is large, for example $T = 100$, the density is better but it takes computational time more than as in Table 2. It is slow because the number of playout amounts to more than 5,000 when $T = 100$. When T is less than $T = 30$, however, the density is worse than simple BL approach. Hence, sufficient number of simulation is necessary to improve the density but it requires computational cost.

Figure 6 is buffer capacity b and density. The density increases as the b increases. The computational time is 12.0 [ms] and much faster than the MCTS method.

Figure 7 and Figure 8 are packing results of online-BL and MCTS against an instance, respectively. In Figure 7, the shapes of upper faces of the packed items are not flat. Thus it is difficult to stack an item on them due to stability check (we check overlap ratio is greater than 0.5). In Figure 8, the search by MCTS avoids such situation.

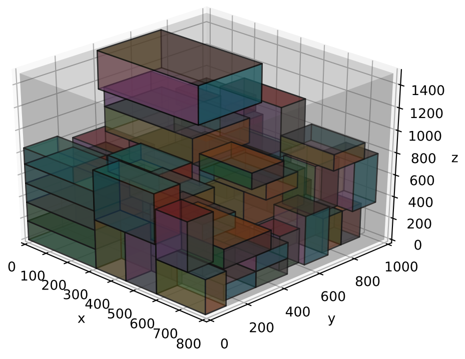

図 **7** The packing result by online BL

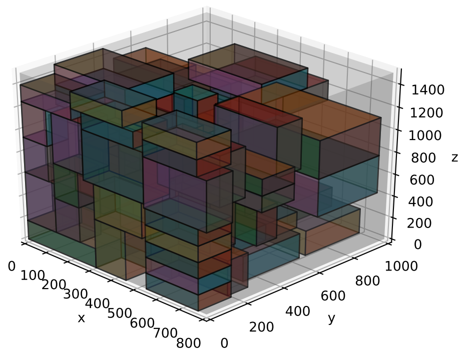

図 **8** The packing result by MCTS

5. Conclusions

We introduce semi-online container loading problems relaxing the online problems and proposed heuristics algorithms. The computational results show that such relaxing is effective to improve density against genuine online problem. These semi-online setting could be acceptable in some situations. The buffer which temporary stores the input items is an easy solution to improve density. It requires extra physical space, however, it might be difficult to apply in actual application. When the buffer is unavailable and the future item sizes are known in advance, the MCTS based search also improves the density. The computational time is very slow because a lot of simulation is necessary, thus the application might be limited to the case where manipulation of robot arm takes few seconds, for example.

	k	b	density(m)[%]	density(sd)[%]	time[ms]
static BL	$n-1$	n	78.0	2.2	48.3(*)
online BL	0	0	50.9	11.4	3.2
semi-online MCTS	$n-1$	0	56.0	6.6	1127.5
semi-online BL	0	5	54.1	9.4	12.0

表 **2** Summary of computational results

Acknowledgement

The authors thank Dr. Toshio Sato and Kentaro Yokoi for their valuable comments.

参考文献

[1] S.D. Allen, E.K. Burke, and G. Kendall: A hybrid placement strategy for the three-dimensional strip packing problem, European Journal of Operational Research, 209(3) (2011), pp. 219–227.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer: Finite-time Analysis of the Multiarmed Bandit Problem, Machine Learning 47 (2002), pp. 235–256.

[3] B.S. Baker, E.G. Coffman, and R.L. Rivest: Orthogonal packings in two dimensions, SIAM Journal on Computing, 9(4) (1980), pp. 846–855.

[4] S. Berndt, K. Jansen, and K.-M. Klein: Fully Dynamic Bin Packing Revisited, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015), 40 (2015), pp. 135–151.

[5] A. Bortfeldt and G. Wäscher: Constraints in container loading–a state-of-the-art review, European Journal of Operational Research, 229(1)(2013), pp. 1–20.

[6] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, et al.: A Survey of Monte Carlo Tree Search Methods, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, 4(1) (2012), pp. 1–43.

[7] C.T. Ha, T.T. Nguyen, L.T. Bui, and R. Wang: An Online Packing Heuristic for the Three-Dimensional Container Loading Problem in Dynamic Environments and the Physical Internet, Applications of Evolutionary Computation (2017), pp. 140–155.

[8] S. Imahori, Y. Chien, Y. Tanaka, and M. Yagiura: Enumerating bottom-left stable positions for rectangle placements with overlap, Journal of the Operations Research Society of Japan, 57(1) (2014), pp. 45–61.

[9] K. Karabulut, M.M. İnceoğlu: A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method, International Conference on Advances in Information Systems, ADVIS 2004, Lecture Notes in Computer Science, vol 3261. (2014), Springer, Berlin, Heidelberg.

[10] H. Kawashima, Y. Tanaka, S. Imahori, and M. Yagiura: An efficient implementation of a constructive algorithm for the three-dimensional packing problem, In Proceedings of the 9th Forum of Information Technology 2010, 1, pp. 31–38.

[11] L. Kocsis and C. Szepesvári: Bandit based Monte-Carlo Planning, ECML 2006, Lecture Notes in Computer Science, 4212, pp. 282—293.

[12] L. Wang, S. Guo, S. Chen, W. Zhu, and A. Lim: Two Natural Heuristics for 3D Packing with Practical Loading Constraints, In proceedings of PRICAI 2010, Lecture Notes in Computer Science, 6230, pp. 256–267.

[13] X. Zhao, J.A. Bennell, T. Bektaş, and K. Dowsland: A comparative review of 3D container loading algorithms, International Transactions in Operational Research, 23(1-2)(2016), pp. 287–320.