




# A two-phase constructive algorithm for the single container mix-loading problem

Tian Tian<sup>1</sup> · Wenbin Zhu<sup>2</sup> · Ying Zhu<sup>3</sup> · Qiang Liu<sup>3</sup> · Lijun Wei<sup>3</sup> 

Received: 11 July 2022 / Accepted: 8 August 2023 / Published online: 1 September 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

## Abstract

Manufacturers usually store their products in palletized storage units (PSUs). PSUs are convenient for storage but sometimes not cost-effective for transportation because they may result in large empty spaces of waste in containers. To improve the utilization of its containers, a manufacturer is willing to remove products from PSUs (a process called *depalletizing*) and load the individual products, together with other PSUs, into a container. Once a PSU is depalletized, its products must be loaded into the container. No PSU can be depalletized if the total volume of complete PSUs loaded in the container is not maximized. We introduce this problem as the single container mix-loading problem (SCMLP). Then, we develop a two-phase constructive algorithm for the SCMLP that uses a stochastic beam-search-based method developed for loading items into a given set of spaces as the sub-routine. In the first phase, the stochastic beam-search-based method is called upon to load PSUs into the container. In the second phase, a proper set of PSUs is selected, and the stochastic beam-search-based method is used to load all products of the selected PSUs into the remaining spaces in the container. The performance of our algorithm is demonstrated by experiments conducted on a set of instances generated from the historical data of the manufacturer. Besides, we also used the well-known 1500 single container loading problem instances to test the performance of

---

✉ Lijun Wei  
villagerwei@gdut.edu.cn

Tian Tian  
tiantian@dufe.edu.cn

Wenbin Zhu  
i@zhuwb.com

Qiang Liu  
liuqiang@gdut.edu.cn

<sup>1</sup> School of Management Science and Engineering, Key Laboratory of Liaoning Province for Data Analytics and Decision-Making Optimization, Dongbei University of Finance and Economics, Dalian, Liaoning 116025, China

<sup>2</sup> School of Business Administration, South China University of Technology, Guangzhou, Guangdong 510641, China

<sup>3</sup> Guangdong Provincial Key Laboratory of Computer Integrated Manufacturing System, State Key Laboratory of Precision Electronic Manufacturing Technology and Equipment, Guangdong University of Technology, Guangzhou, Guangdong 510006, China

our stochastic beam-search-based method, and the results showed that our approach is highly competitive with state-of-the-art methods.

**Keywords** Container loading · Packing · Beam search · Heuristic

## 1 Introduction

Our team collaborates with an international audio equipment manufacturer to solve problems that had emerged in its logistics process. The Manufacturers usually load their products onto pallets for storage. These loaded pallets are called Palletized Storage Units (PSUs) in the industry. To operate efficiently, a PSU usually consists of identical products. When delivering products, manufacturers load PSUs (instead of individual products) into containers (or trucks). Loading PSUs is convenient; however, large spaces on the tops or sides of the loaded PSUs could be wasted in each container. To improve the utilization of a container, the audio equipment manufacturer is willing to remove products from PSUs (a process called *depalletizing*) and load the products, together with other PSUs, into the container.

When a container delivered by the manufacturer arrives at its destination, all individual products loaded in it are rearranged into PSUs (a process called *repalletizing*). This is because moving PSUs from the port (or other unloading areas) to the customer's warehouse is much more efficient and safe than moving individual products, and customers prefer to store PSUs rather than individual products in their warehouses. Therefore, once a PSU is depalletized, its products must be loaded into the container.

Both depalletizing and repalletizing increase the complexity of the manufacturer's operations, thus increasing its operational costs. Only when the transportation cost saved by improving the volume utilization of each container is much higher than the operational cost induced by depalletizing and repalletizing would the manufacturer prefer to depalletize PSUs and load products, together with PSUs, into containers. As a result, the partner manufacturer constrains that no PSUs will be depalletized if the total volume of complete PSUs loaded in the container is not maximized.

To summarize, the audio equipment manufacturer would like to depalletize PSUs and load the individual products, together with other PSUs, into a container such that the volume utilization of the container is maximized. Once a PSU is depalletized, its products must be loaded into the container. No PSU should be depalletized if the total volume of complete PSUs loaded in the container is not maximized. We name the resulting optimization problem the Single Container Mix-Loading Problem (SCMLP) and formally define it in Sect. 3.

In Sect. 4, we develop a two-phase constructive algorithm for the SCMLP. The algorithm uses a stochastic beam-search-based method developed for loading items into a given set of spaces as a sub-routine. Beam search is a variant of the branch-and-bound technique. Unlike the traditional beam search, which expands the most promising nodes at each level of a search tree, we add the stochastic process to select the node randomly based on the solution quality of the nodes. In other words, the promising nodes will be given a high probability. In the first phase of our constructive algorithm, the stochastic beam-search-based method is called upon to load PSUs into the container. In the second phase, a proper set of PSUs is selected, considering the remaining volume of the container, and the stochastic beam-search-based method is used to load all products of the selected PSUs into the remaining spaces of the container.

We generate 70 test instances based on historical data provided by the audio equipment manufacturer, and we conduct extensive computational experiments to demonstrate the effectiveness of our approach, as described in Sect. 5.

Commonly, the manufacturer's products are packaged in cuboid cartons. So, we use cartons to represent products in the following sections. To the best of our knowledge, there is no literature concerned with the single container mix-loading problem. However, the SCMLP can be considered an extension of the Single Container Loading Problem (SCLP). When no PSU can be depalletized, the single container mix-loading problem is reduced to the single container loading problem (SCLP). The 1500 well-studied SCLP instances by Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) are selected for the experiments. The results show that our approach is highly competitive with state-of-the-art methods, and the stochastic strategy improves the results compared to the traditional beam search method. The detailed results are given in Sect. 5.

## 2 Literature review

The efficient loading of three-dimensional items into three-dimensional containers is a fundamental problem in the freight transportation and logistics industries. A tremendous amount of literature in operations research has been devoted to various container loading problems (Lim et al., 2013; Wang et al., 2013), from single-container versions to multiple-container ones (Wei et al., 2015; Kurpel et al., 2020).

In the **Single Container Loading Problem (SCLP)**, given a fairly large set of small cuboid items and one cuboid container, we have to pack some or all of the items feasibly into the given container, such that the total value of the packed items is maximized. The assortment of small items could be identical, weakly heterogeneous, or strongly heterogeneous. The SCLP is an output maximization problem, according to the typology developed by Wäscher et al. (2007).

If the volume of an item is viewed as its value, the objective of the single container loading problem becomes maximizing the total volume of the packed items or, equivalently, minimizing the unused space in the container (George & Robinson, 1980). Single container loading problems can be further classified by the number of items of each type that can be packed into the container. If the number is limited with a lower and/or an upper bound, it is called a constrained problem; otherwise, it is an unconstrained problem.

The SCLP is a strongly  $\mathcal{NP}$ -hard combinatorial optimization problem because the one-dimensional bin packing problem, which is an  $\mathcal{NP}$ -hard optimization problem, is a special case (Martello et al., 2000). Therefore, it is not easy to solve the SCLP to optimality. The literature on exact algorithms for SCLP is relatively scarce, and we list some relevant studies as follows.

Martello et al. (2000) developed a branch-and-bound algorithm for SCLP, where only the corner points are considered as the candidate place. Later, Den Boef et al. (2005) pointed out that only considering the corner points is insufficient and proposed a new approach based on constraint programming. Hifi (2004) proposed two exact algorithms for a three-dimensional cutting problem, equivalent to the single container loading problem in the packing context. The first algorithm uses the dynamic programming technique, and it is extended from an approach designed for the two-dimensional version (Gilmore & Gomory, 1966). The second algorithm also adapts a method developed for the two-dimensional version (Hifi & Zisisimopoulos, 1996), which is based on a graph search procedure with a depth-first search

strategy. Most recently, Fekete et al. (2007) developed a two-level tree search algorithm for solving high-dimensional packing problems, including various container loading problems, to optimality. They first invented a data structure for feasible packing based on graph-theoretic characterizations of interval graphs and then combined the data structure with other heuristics and the lower bounds they computed in earlier research works (Fekete & Schepers, 1997a, b, 2004a, b). They conducted computational experiments for instances with up to 80 small items, and more than 70% of these instances were solved optimally. Silva et al. (2019) provided an up-to-date comparative study concerning the most significant exact methods associated with SCLP.

Lots of heuristics have been developed to handle instances with a larger size; these heuristics can be classified into four groups (Pisinger, 2002; Zhao et al., 2016): wall building approaches (George & Robinson, 1980), stack building approaches (Gehring & Bortfeldt, 1997), cuboid arrangement (or block building) approaches (Eley, 2002), and guillotine cutting approaches (Morabito & Arenales, 1994). This classification can be extended by adding horizontal-layer building approaches (Bischoff & Ratcliff, 1995).

As the wall-building approaches, the container is filled with walls (or vertical layers) of small items across one of its horizontal dimensions. George and Robinson (1980) proposed the first wall-building method. Several variants have been presented since then. Bischoff and Marriott (1990) compared 14 different heuristics based on the framework by George and Robinson (1980). Bischoff and Ratcliff (1995) pointed out that wall-building approaches might produce unstable packing, which is dangerous in transportation. They introduced a horizontal-layer building approach, in which the container is loaded from the floor upwards using layers of up to two types of small items. This approach extends the heuristic method proposed by Bischoff et al. (1995).

After the development of the wall building and horizontal-layer building approaches, stack-building approaches were proposed, in which the container is packed with stacks (or towers/columns) of small items. Gehring and Bortfeldt (1997) developed a two-step stack-building approach. In the first step, small items are stacked into towers, in which each item is fully supported from below by the surface of another item or by the container floor. Then, a genetic algorithm is called upon to arrange the towers into the container, thus solving a two-dimensional packing problem.

All the above three types of building approaches simulate the process of workers packing items into a container manually.

The container in the cuboid arrangement (or block building) approach is filled by cuboid arrangements (or blocks) made up of items (Bortfeldt & Gehring, 1998). Eley (2002) built the block using identical items, developed a greedy search method to arrange the blocks, and then improved the solution found by the greedy method using a tree search approach. Bortfeldt et al. (2003) and Mack et al. (2004) followed this work by using blocks composed of identical items in their heuristic methods. Later, Fanslau and Bortfeldt (2010) improved the performance of cuboid arrangement approaches by introducing blocks with small inner gaps. Zhu and Lim (2012) and Zhang et al. (2012) proved the efficiency of using both blocks of identical items and blocks of different items in heuristic methods. Araya and Riff (2014) improved the results further by replacing the greedy look-ahead heuristic with the beam search. Recently, Araya et al. (2017) introduced a new heuristic function for selecting boxes and got the best results on the 1600 well-known benchmark instances. Araya et al. (2020) extended the beam search approach to the bi-objective container loading problem, where the second objective is maximizing the loaded boxes' total profit.

Morabito and Arenales (1994) proposed the guillotine-cutting approach. In their approach, the container is partitioned by guillotine cuts into small parts in which only one item can be

packed. The result of partitioning the container is called a “guillotine partition pattern”, and each guillotine partition pattern can be represented in an AND/OR graph. The best guillotine partition pattern (corresponding to the best solution to the SCLP) can be found by determining the most valuable complete path in the AND/OR graph.

In a departure from all the above approaches, Ngoi et al. (1994) developed an algorithm that does not make use of any item arrangement (e.g., walls, stacks, etc.) and does not constrain the loading sequence (e.g., from back to front, from floor to ceiling, etc.). Their algorithm is based on a spatial representation technique proposed by Ngoi and Whybrew (1993), in which the position and dimensions of each packed item and each empty space are stored in a three-dimensional matrix. The algorithm iteratively selects an item and places it in empty space until specific stopping criteria are satisfied. Similarly, Huang and He (2009); He and Huang (2011) proposed caving degree approaches for the single container loading problem, which do not use any special arrangement of items and do not constrain the loading sequence. They developed a placement rule based on the caving degree, which always favors corners or even caves of the container when packing items, such that the items are packed as compactly as possible.

Lim et al. (2003) also developed an approach that does not involve any special arrangement of items but does stipulate that items should be packed from the floor up. In their Multi-Faced Buildup method, all container walls can be used as a base (or floor) to place items. Additionally, the authors combined the Multi-Faced Buildup technique with a look-ahead strategy to improve the solution quality.

Among all the methods mentioned above, some are classic heuristic methods, while others are advanced. The methods proposed by Bischoff and Ratcliff (1995); Bischoff et al. (1995); Lim et al. (2003) belong to the classic heuristic methods. The advanced heuristic methods include tree search methods (Eley, 2002; Fanslau & Bortfeldt, 2010; Pisinger, 2002; Terno et al., 2000; Zhang et al., 2012), graph search methods (Morabito & Arenales, 1994), the Simulated Annealing algorithms (Mack et al., 2004), the Tabu Search algorithms (Bortfeldt et al., 2003), the Genetic Algorithms (Bortfeldt & Gehring, 2001; Gehring & Bortfeldt, 1997, 2002), and the Greedy Randomized Adaptive Search Procedures (Moura & Oliveira, 2005; Parreño et al., 2008). Commonly, classic heuristic methods are used to construct initial solution(s) for local search methods, such as the Simulated Annealing algorithms, the Tabu Search algorithms, and population-based heuristics (e.g., the Genetic Algorithms). For a comparative review of 3D container loading algorithms, the reader is referred to Zhao et al. (2016).

The SCMLP we proposed is closely related to the Container Loading Problem for Pallets with Infill Boxes (CLPIB) proposed by Sheng et al. (2016). In the CLPIB, the bottoms of each pallet must be fully supported by the container floor or by the top of a single pallet. While in the SCMLP, each pallet could be fully supported by the container floor or any combination of pallets and cartons. Therefore, the space utilization of a container in the SCMLP is higher, and the solution space of SCMLP is larger than those in the CLPIB.

### 3 Problem definition of SCMLP

We are given a container denoted by  $C$ , whose dimensions are  $L$ ,  $W$ , and  $H$  and whose volume is  $V = LWH$ , and a set of PSUs denoted by  $\mathcal{P}$ . Each PSU  $p \in \mathcal{P}$  has the dimensions  $L_p$ ,  $W_p$ , and  $H_p$  and the volume  $V_p = L_p W_p H_p$ . We use an indicator  $d_p$  to denote whether PSU  $p \in \mathcal{P}$  can be depalletized. If  $d_p$  equals 1, PSU  $p$  is depalletizable; otherwise,  $p$  is not

depalletizable. For example, the cartons in some PSUs are fragile, and a wooden box may be used during palletizing to protect the cartons from being crushed. The PSUs, in this case, are not depalletizable. Each PSU comprises  $N_p$  identical cartons, and each carton in the PSU  $p$  has the dimensions  $l_p, w_p,$  and  $h_p$  and the volume  $v_p = l_p w_p h_p$ . The set of all cartons in the PSU  $p$  is denoted by  $\mathcal{I}_p$ . We use  $V_{I_p} = N_p v_p$  to denote the volume of all cartons in  $p$ . It should be pointed out that the volume of a PSU may be larger than the total volume of the cartons contained within it ( $V_p \geq V_{I_p}$ ) because gaps between cartons may exist in each PSU.

We define a **depalletizable PSU set**, denoted by  $\mathcal{D}$ , as a set of PSUs ( $\mathcal{D} \subseteq \mathcal{P}$ ) whose elements are all depalletizable ( $d_p = 1, \forall p \in \mathcal{D}$ ), and we use the set  $\mathcal{DP}$  to store all of the depalletizable PSU sets. Suppose there are  $n$  depalletizable PSUs in the set  $\mathcal{P}$ . In this case, the cardinality of the set  $\mathcal{DP}$  is  $2^n$ . Note that the empty set belongs to the set  $\mathcal{DP}$ .

As described in Sect. 1, in the single container mix-loading problem, we need to select two **disjoint** subsets of PSUs:

1.  $\mathcal{S}_1 \subseteq \mathcal{P}$ :  $\mathcal{S}_1$  is a set of complete PSUs.
2.  $\mathcal{S}_2 \in \mathcal{DP}$ :  $\mathcal{S}_2$  is a depalletizable PSU set.

Next, we need to load all of the PSUs in  $\mathcal{S}_1$  and all of the cartons in the set  $\mathcal{I} = \cup_{p \in \mathcal{S}_2} \mathcal{I}_p$  into the container  $C$ , such that the total volume of the complete PSUs in the set  $\mathcal{S}_1$  is maximized and the volume utilization of the container  $C$  is maximized. The volume utilization of the container  $C$  is defined as:

$$\frac{\sum_{p \in \mathcal{S}_1} V_p + \sum_{i \in \mathcal{I}} v_i}{V} \tag{1}$$

We define that a set of items, either PSUs or cartons, can be loaded into the container if the following geometric constraints are satisfied:

- **Containment:** Each item must be placed completely within the container.
- **Orthogonal placement:** Each item must be placed with its edges parallel to those of the container.
- **No overlap:** Any two items inside the container must be interior-disjointed.
- **Full support:** The bottom of any item in the container must be fully supported by either the container floor or the tops of other items.
- **Orientation restriction:** Every item must be placed in one of its allowed orientations. We use three flags ( $f^l, f^w,$  and  $f^h$ ) to indicate whether an item can be placed with its length, width, and height, respectively, aligned with the vertical axis of the container. For example, if  $f^h = 1$ , the item can be placed with its height aligned with the vertical axis of the container.
  - For any pallet  $p \in \mathcal{P}$ , we have  $f_p^l = 0, f_p^w = 0,$  and  $f_p^h = 1$ .
  - For any carton  $i \in \mathcal{I}_p, \forall p \in \mathcal{P}$ , we have  $f_i^l = 1, f_i^w = 1,$  and  $f_i^h = 1$ .

### 4 A two-phase constructive method

A two-phase constructive algorithm is developed for the SCMLP. It uses a stochastic beam-search-based method developed for loading items into a given set of spaces as its sub-routine. Beam search is a branch-and-bound technique variant that expands the most promising nodes at each level of a search tree. In the first phase of our constructive algorithm, the stochastic beam-search-based method is called upon to load PSUs into the container. In the second

phase, a proper set of PSUs is selected, considering the remaining volume of the container, and the stochastic beam-search-based method is used to load all products of the selected PSUs into the remaining spaces of the container.

Our constructive algorithm is a block-building approach, according to the classification proposed by Pisinger (2002). In block-building approaches, blocks of items, rather than individual items, are loaded into residual spaces.

## 4.1 A stochastic beam-search based method

Beam search is a tree search method whose search tree is built with a breadth-first strategy. However, in contrast to the breadth-first search, at each level of the search tree, only a predetermined number of nodes are selected by an evaluation function and expanded. This predetermined number is referred to as the beam width. If the beam width is infinite, all nodes will be selected and expanded at each level of the search tree; thus, the beam search is equivalent to the breadth-first search.

Each intermediate node of the search tree represents a partial solution to the problem, and each leaf node represents a complete solution. Beam search does not guarantee termination with an optimal solution because the nodes leading to the optimal solutions may be pruned.

We present a stochastic beam-search-based method that iteratively calls upon a beam search in the following sections. This method solves the problems of loading three-dimensional items into a given set of three-dimensional residual spaces in a container, such that the total volume of loaded items is maximized. Here, the items refer to the PSUs and cartons in the SCMLP. It is a block-building approach in which blocks of identical items, rather than individual items, are loaded into the residual spaces.

### 4.1.1 Node representation of beam search

Four elements represent each node of the search tree of the beam search we developed:

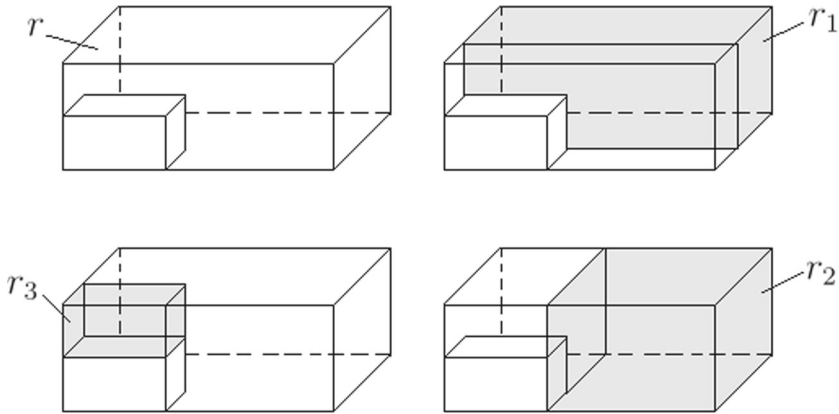
- A set of remaining items, denoted by  $I$ .
- A set of blocks, denoted by  $B$ .
- A set of three-dimensional residual spaces in the container  $C$ , denoted by  $R$ .
- A set of loaded items, denoted by  $LI$ .

A block  $b \in B$  is the minimum bounding cuboid of an arrangement of items from the set  $I$ . The items in block  $b$  are identical, and they are arranged in such a way as to satisfy the orthogonal placement constraint, the no-overlap constraint, the full-support constraint, and the orientation restriction described in Sect. 3. We present the method for generating blocks below.

A residual space  $r \in R$  is a fully-supported maximal space that does not overlap with any placed blocks in the container  $C$ . The *maximal space* concept was first introduced by Lim et al. (2003) and has been used in several other works (He & Huang, 2011; Parreño et al., 2008; Zhu & Lim, 2012). Figure 1 shows three new residual spaces generated when a block is placed in the residual space  $r$ . We describe the generation approach for fully-supported maximal spaces below.

In the root node of the search tree,  $I$  contains all available items,  $B$  contains the blocks generated using the items in the set  $I$ ,  $R$  contains all available spaces, and  $LI$  includes the already packed items. Each intermediate node of the search tree represents a partial solution to the problem, in which sets  $I$ ,  $B$ ,  $R$ , and  $LI$  are all non-empty. A leaf node represents a





**Fig. 1** Three new residual spaces ( $r_1, r_2, r_3$ ) generated when a block is placed in the residual space  $r$

complete solution to the problem, in which either  $I$  is an empty set meaning all the items are packed, or no item in  $I$  can be loaded into any space in  $R$ .

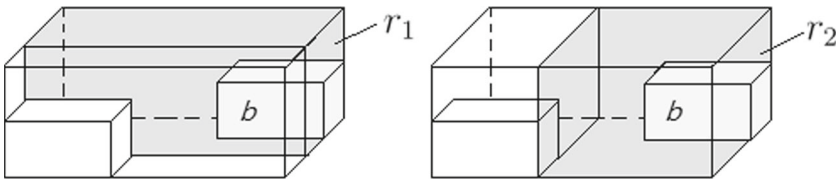
**Block generation:** In existing block-building approaches for container loading problems, two types of blocks are generally used: namely, the simple block and the general block. A simple block consists of identical items, and all items in a simple block are placed with the same orientation. Therefore, each item in a simple block is fully supported from below by either the top of another item or the base of the block. A general block may either be composed of two or more types of items or be composed of identical items with different orientations. Special operations are involved in generating general blocks to satisfy the full-support constraint.

Both simple blocks and general blocks are proven to be effective when loading items into spaces (Bortfeldt et al., 2003; Eley, 2002; Fanslau & Bortfeldt, 2010; Mack et al., 2004; Zhang et al., 2012; Zhu & Lim, 2012). Zhu et al. (2012) observed that using both simple blocks and general blocks are better for solving problems with strongly heterogeneous items while using only simple blocks is better for solving problems with identical or weakly heterogeneous items. We use simple blocks in our stochastic beam-search-based method because the full-support constraint is easily satisfied, and the PSUs and cartons in the SCMLP are weakly heterogeneous.

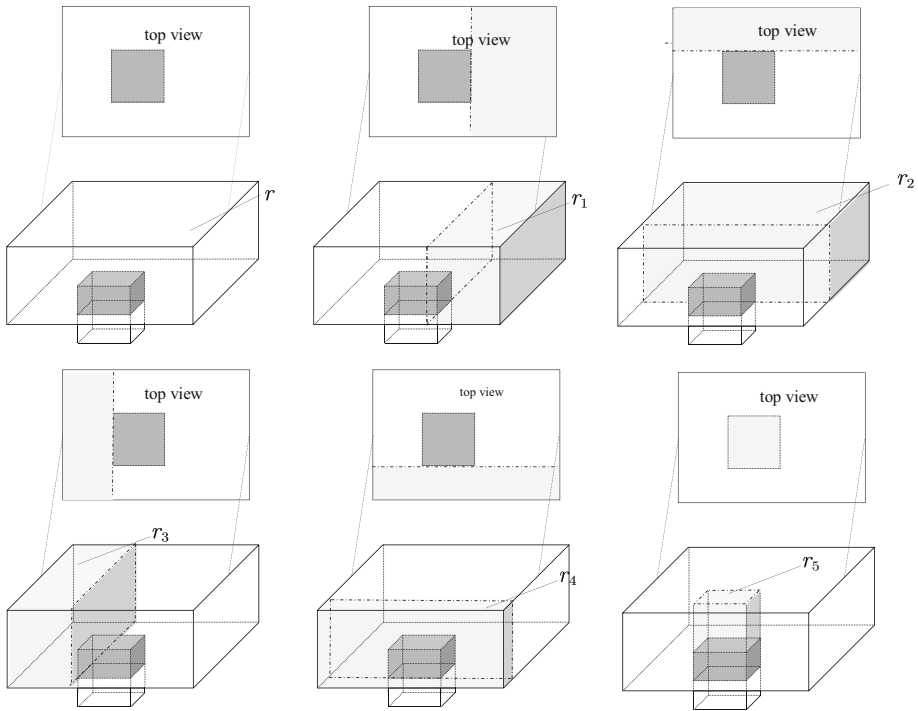
A simple block of item  $i \in I$  can be generated by replicating the item  $i$  for  $n_l$ ,  $n_w$ , and  $n_h$  times along the length, width, and height, respectively, of the block. Therefore, all simple blocks composed of item  $i$  can be generated by enumerating all possible values of  $n_l$ ,  $n_w$ , and  $n_h$ . Note that a single item  $i$  is also a simple block. The algorithm to generate simple blocks will be called upon only once at the beginning of the stochastic beam-search-based method, and all the simple blocks form the set  $B$  of the root node of the search tree. Simple blocks with the same dimensions and containing the same set of items are considered to be duplicates, so only one of these is stored, and the others are deleted. Simple blocks too large to be loaded into any residual space in the root node are also deleted.

**Residual space generation:** The residual spaces in the container  $C$  are fully-supported maximal spaces in our beam search. When no block is placed in a container, the whole space within the container is a fully-supported maximal space. Once a block is placed at a corner of the residual space  $r$ , three fully-supported maximal spaces are generated by (1) performing a guillotine cut parallel to the length of  $r$  (see  $r_1$  in Fig. 1); (2) performing a guillotine cut





**Fig. 2** A block  $b$  intrudes into the residual space  $r_2$  when it is placed in the residual space  $r_1$



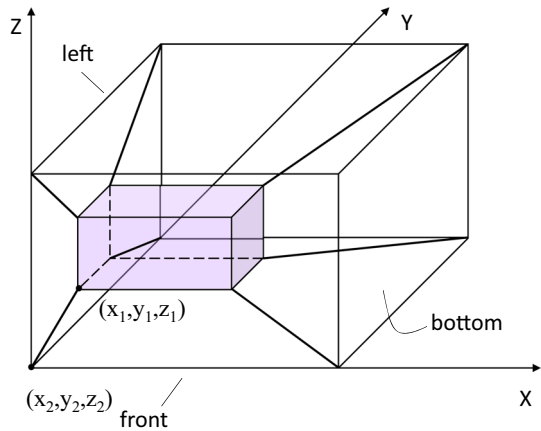
**Fig. 3** Five new residual spaces ( $r_1$ ,  $r_2$ ,  $r_3$ ,  $r_4$ , and  $r_5$ ) generated when a block  $b$  intrudes into the residual space  $r$

parallel to the width of  $r$  (see  $r_2$  in Fig. 1); and (3) cutting a space above the block, such that the base of the space is the top of the block (see  $r_3$  in Fig. 1). Note that the spaces  $r_1$  and  $r_2$  in Fig. 1 overlap.

Due to overlapping between fully-supported maximal spaces, it may intrude into other residual spaces when a block is placed at a corner of residual space. Figure 2 shows an example in which a block  $b$  is placed in the residual space  $r_1$  and intrudes into the residual space  $r_2$ .

Once a block  $b$  is placed in a residual space, the intruded residual spaces must be updated. We first perform, at most, two guillotine cuts parallel to the length of  $r$ , which will result in two fully-supported maximal spaces (see  $r_2$  and  $r_4$  in Fig. 3). Next, we perform, at most, two guillotine cuts parallel to the width of  $r$ , resulting in two fully-supported maximal spaces (see  $r_1$  and  $r_3$  in Fig. 3). Finally, we cut a space above the block, such that the base of the space is the top of the block (see  $r_5$  in Fig. 3). No space below the block will be generated

**Fig. 4** The relationship between the corners of the residual space  $r$  and the corners of the container  $C$



because of the full-support constraint. Therefore, we will get, at most, five fully-supported maximal spaces when updating an intruded residual space. Figure 3 demonstrates the five new residual spaces generated when a block  $b$  intrudes into the residual space  $r$ .

#### 4.1.2 Child node generation of beam search

Consider an intermediate node whose elements are  $I$ ,  $B$ ,  $R$ , and  $LI$ . We call this a parent node. One child of the parent node is generated by placing a block  $b \in B$  into a residual space  $r \in R$ .

##### Space selection:

Each residual space is a cuboid and has eight corners. Each corner of the residual space  $r$  is associated with a corner of the container  $C$  (see Fig. 4). For example, the front-left-bottom corner of  $r$  is related to the front-left-bottom corner of  $C$ . The Manhattan distance between a corner of the residual space  $r$  and the corresponding corner of the container  $C$  is defined as  $|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$ , where  $(x_1, y_1, z_1)$  is the coordinate of the corner of  $r$  and  $(x_2, y_2, z_2)$  is the coordinate of the corner of  $C$  (see Fig. 4). The corner of  $r$  with the smallest Manhattan distance from its corresponding corner of  $C$  is called the *anchor corner*, and the smallest Manhattan distance is called the *anchor distance*.

When generating a child of an intermediate node, the beam search selects the usable residual space of  $R$  with the smallest anchor distance. Ties are broken by larger volume of the residual space. A usable residual space is a residual space that is large enough to accommodate at least one block of  $B$ .

##### Block selection:

Given the selected residual space  $r \in R$ , we use function  $f(b, r) = V - (V_{waste} + V_{loss})$  to evaluate each block  $b \in B$ . The block with the largest value of  $f(b, r)$  is selected. This function was introduced by Zhu and Lim (2012).  $V$  is the total volume of all the items in block  $b$ .  $V_{waste}$  is the difference between the volume of block  $b$  and  $V$ .  $V_{loss}$  is the lower bound of the unusable volume of the residual space  $r$ . The unusable volume of the residual space  $r$  is the difference between the volume of  $r$  and the maximum total volume of items in  $I$  that can be loaded into  $r$ .

If the block  $b$  cannot be loaded into the residual space  $r$ , the function  $f(b, r)$  returns negative infinity; otherwise,  $V_{loss}$  is computed as follows. We first calculate the remaining

length (width, height) of the residual space  $r$  that can be used after placing the block  $b$  and denote this length (width, height) by  $l(w, h)$ . Next, we find the dimensions of each item  $i \in I$  that can be placed along the length (width, height) of the residual space  $r$  and store all these dimensions in the set  $L_r(W_r, H_r)$ . Then, we solve the (one-dimensional) knapsack problem, in which the capacity of the knapsack is  $l(w, h)$  and the set of items to be packed in the knapsack is  $L_r(W_r, H_r)$ , using the standard Dynamic Programming (DP) algorithm proposed by Martello and Toth (1990) which runs in pseudo-polynomial time. Suppose the optimal solution value returned by the dynamic programming algorithm is  $l_{max}(w_{max}, h_{max})$ . In this case,  $V_{loss}$  is equal to  $V(r) - (l(b) + l_{max}) \times (w(b) + w_{max}) \times (h(b) + h_{max})$ , where  $V(r)$  is the volume of the residual space  $r$  and  $l(b)$ ,  $w(b)$ , and  $h(b)$  are the length, width, and height, respectively, of the block  $b$ .

Once the block  $b$  is placed at the anchor corner of the selected residual space  $r$ , all four elements of the parent node ( $I$ ,  $B$ ,  $R$ , and  $LI$ ) are updated. The updated elements are inherited by the child node generated by placing  $b$  in  $r$ .

**Updating the Item Set  $I$ , the Block Set  $B$ , and the Loaded-Item Set  $LI$ :** All items in  $b$  will be deleted from  $I$  and inserted into  $LI$ . Some blocks in  $B$  are also deleted because there are not enough items in  $I$  to form these blocks.

**Updating the Residual-Space Set  $R$ :** Once the block  $b$  is placed in the residual space  $r$ , new residuals will be generated according to the method described in Sect. 4.1.1. The residual space  $r$  will be deleted from the set  $R$ , and all newly generated residual spaces will be inserted into  $R$ . Due to the overlapping nature of the fully-supported maximal spaces, some spaces may be completely contained by other residual spaces. Therefore, a process is called upon to delete these completely contained residual spaces once new residual spaces are inserted into  $R$ . Some spaces are too small to accommodate any block in the updated block set  $B$ . These small residual spaces are marked as unusable but not deleted from  $R$ .

### 4.1.3 Node evaluation of beam search

In beam search, at each level of the search tree, only a number (beam width) of child nodes are selected for further expansion based on an evaluation approach. In our beam search, we use a greedy strategy to construct a complete solution from a given child node and use the total volume of loaded items in the complete solution to evaluate the child node.

The greedy strategy works as follows. Given a child node ( $I, B, R, LI$ ) representing a partial solution to the SCMLP, a complete solution ( $I', B', R', LI'$ ) is constructed by iteratively calling upon the four operations:

- Select the usable residual space  $r \in R$  with the minimum anchor distance.
- Select the block  $b \in B$  with the maximum value of the function  $f(b, r)$ .
- Place the selected block  $b$  in the selected residual space  $r$ .
- Update the item set, the block set, the residual-space set, and the loaded-item set into  $I'$ ,  $B'$ ,  $R'$ , and  $LI'$ .

This continues until no item remains in  $I'$ , no usable residual space remains in  $R'$ , or no item can be loaded into any usable residual space. Then, the evaluation function of the given child node is the volume utilization (the percentage of loaded items' volume to that of the container) in the corresponding complete solution, which is denoted by  $g(LI')$ .

#### 4.1.4 The stochastic beam-search based method

Our stochastic beam-search-based method is described in Algorithm 1, which iteratively calls upon the stochastic beam search presented in Algorithm 2.

---

#### Algorithm 1 The Stochastic Beam-Search Based Method

---

```

SBSM ( $I, R, LI$ )
  // Input:  $I$ : a set of items
  //        $R$ : a set of residual spaces
  //        $LI$ : a set of loaded items
1:  $bestSol = NULL$ 
2:  $w = 1$ 
3: while time limit is not exceeded do
4:    $I' = I, R' = R, LI' = LI$ 
5:    $B =$  generate all simple blocks composed of items in  $I$ 
6:   StochasticBeamSearch ( $I', B, R', LI', bestSol$ )
7:    $w = \lceil \sqrt{2}w \rceil$ 
8: end while
9: return  $bestSol$ 

```

---

The input of the stochastic beam-search-based method includes a set of three-dimensional items  $I$  and a set of three-dimensional residual spaces  $R$ .  $bestSol$  is used to store the best complete solution found by the method, and it consists of four elements,  $I_{best}$ ,  $B_{best}$ ,  $R_{best}$ , and  $LI_{best}$ , which represent the set of remaining items, the set of blocks composed of items in  $I_{best}$ , the set of remaining residual spaces, and the set of loaded items, respectively. The beam width of our beam search is represented by the parameter  $w$ .  $w$  also determines the search effort because it limits the number of child nodes that can be generated and evaluated to be, at most,  $w^2$ , as presented in Algorithm 2.  $w$  is increased to  $\lceil \sqrt{2}w \rceil$  after each execution of Algorithm 2, such that the search effort is doubled.

The stochastic beam search iteratively called upon in our beam-search-based method is described in Algorithm 2.

The root node is initialized with the four elements  $I$ ,  $B$ ,  $R$ , and  $LI$ . A set  $N$  is used to store all of the nodes at the current level of the search tree. Each iteration in Algorithm 2 executes the following operations: 1)  $w$  child nodes of each node  $n_i \in N$  are generated using process GenerateChildNodes shown as Algorithm 3, and all  $w^2$  child nodes are stored in set  $N'$ ; 2) only  $w$  nodes in set  $N'$  are selected to compose the next level of the search tree using process Greedy shown as Algorithm 4. Note that  $w^2$  child nodes will be generated for the root node, such that each level of the search tree has the same number of children (Araya & Riff, 2014).  $bestSol$  stores the best complete solution found by Algorithm 4.

The method for generating  $w$  child nodes of a given node  $(I_i, B_i, R_i, LI_i)$  is presented in Algorithm 3.

A residual space  $r \in R_i$  with the minimum anchor distance is selected, with details described in Sect. 4.1.2. Blocks in  $B_i$  are ranked in descending order based on the value of  $f(b, r)$  (see Sect. 4.1.2), and the first  $w$  blocks in the rank are selected. If the number of available blocks in  $B_i$  is smaller than  $w$ , all blocks in  $B_i$  are selected. Then, at most  $w$  child nodes of  $(I_i, B_i, R_i, LI_i)$  are generated by placing each selected block at the anchor corner of the selected residual space  $r$ .

Algorithm 4 describes the method for selecting  $w$  nodes from the set of child nodes of the current level of the search tree ( $N'$ ). For each child node  $n_i = (I_i, B_i, R_i, LI_i) \in N'$ , we iteratively call upon Algorithm 3 with  $w = 1$  until all items in  $I_i$  are loaded or no

**Algorithm 2** The Stochastic Beam-Search

---

```

StochasticBeamSearch ( $I, B, R, LI, bestSol$ )
  //Input:  $I$ : a set of items
  //       $B$ : a set of blocks, each of which consists of items in  $I$ 
  //       $R$ : a set of residual spaces
  //       $LI$ : a set of loaded items
  //       $bestSol$ : the best found solution
1:  $n_0 =$  root node ( $I, B, R, LI$ )
2:  $N = \{n_0\}$ 
3: while  $N \neq \emptyset$  do
4:   the set of child nodes  $N' = \emptyset$ 
5:   for  $n_i = (I_i, B_i, R_i, LI_i) \in N$  do
6:     if  $n_i = n_0$  then
7:        $children =$  GenerateChildNodes( $I, B, R, LI, w^2$ )
8:     else
9:        $children =$  GenerateChildNodes( $I_i, B_i, R_i, LI_i, w$ )
10:    end if
11:     $N' = N' \cup children$ 
12:  end for
13:   $bestChildren =$  Greedy ( $w, N', bestSol$ )
14:   $N = bestChildren$ 
15: end while

```

---

**Algorithm 3** Generate  $w$  Child Nodes of Node ( $I_i, B_i, R_i, LI_i$ )

---

```

GenerateChildNodes ( $I_i, B_i, R_i, LI_i, w$ )
  //Input:  $I_i$ : a set of items
  //       $B_i$ : a set of blocks, each of which consists of items in set  $I_i$ 
  //       $R_i$ : a set of residual spaces
  //       $LI_i$ : a set of loaded items
  //       $w$ : the number of child nodes to be generated
1:  $children = \emptyset$ 
2: select the residual space  $r \in R_i$  with the minimum anchor distance
3: if  $w > |B_i|$  then
4:    $B = B_i$ 
5: else
6:    $B =$  the  $w$  blocks in  $B_i$  that maximize  $f(b, r)$ 
7: end if
8: for  $b \in B$  do
9:   place  $b$  at the anchor corner of  $r$ 
10:   $I'_i =$  update  $I_i, B'_i =$  update  $B_i, R'_i =$  update  $R_i, LI'_i =$  update  $LI_i$ 
11:  put the  $child = (I'_i, B'_i, R'_i, LI'_i)$  into  $children$ 
12: end for
13: return  $children$ 

```

---

item in  $I_i$  can be loaded into any residual space in  $R_i$ . The resultant node is a leaf node  $leaf_i = (I'_i, B'_i, R'_i, LI'_i)$  representing a complete solution, and it is evaluated using the function  $g(LI'_i)$  (see Sect. 4.1.3). Then, we rank all the nodes  $n_i \in N'$  into descending order based on the value of  $g(LI'_i)$ . Unlike the traditional beam search method, we randomly select the first  $w/2$  nodes in the rank and select the left  $w/2$  nodes. The probability of selecting node  $n_i$  is calculated by  $e^{-10dv}$ , where  $dv$  is the volume utilization difference between  $LI'_i$  and  $LI'_i$ . In other words, the node whose complete solution has a larger volume utilization will be selected with a higher probability. Since each leaf node corresponds to a complete solution, we update the best solution  $bestSol$  with the first node in the ranking, if necessary. Note that if the cardinality of  $N'$  is smaller than  $w$ , all nodes of  $N'$  are selected.

**Algorithm 4** Select  $w$  Nodes Using a Greedy Strategy

---

```

Greedy ( $w, N', bestSol$ )
  // Input:  $w$ : the number of nodes to be selected
  //  $N'$ : a set of nodes
  //  $bestSol$ : the best solution found so far
1:  $leafPool = \emptyset, bestChildren = \emptyset$ 
2: if  $w > |N'|$  then
3:   return ( $N', bestSol$ )
4: end if
5: for  $n_i = (I_i, B_i, R_i, LI_i) \in N'$  do
6:    $I = I_i, B = B_i, R = R_i, LI = LI_i$ 
7:   while  $I \neq \emptyset$  and  $R \neq \emptyset$  and some item in  $I$  can be placed in a
      residual space  $r \in R$  do
8:      $leaf_i = (I'_i, B'_i, R'_i, LI'_i) = \text{GenerateChildrenNodes}(I, B, R, LI, 1)$ 
9:      $I = I'_i, B = B'_i, R = R'_i, LI = LI'_i$ 
10:   end while
11:   put  $leaf_i$  in  $leafPool$ 
12: end for
13: Sort nodes in  $N'$  by decreasing  $g(LI')$  value of its leaf node
14: for each node  $n_i \in N'$  do
15:    $dv = g(LI'_i) - g(LI_i)$ 
16:   if size of  $bestChildren$  is smaller than  $w/2$  or
       $\text{Random}(0, 1) \leq e^{-10dv}$  then
17:     put  $n_i$  in  $bestChildren$ 
18:   end if
19:   if size of  $bestChildren$  is equal to  $w$  then
20:     break the for loop
21:   end if
22: end for
23: if  $bestSol = NULL$  or the maximum  $g(LI')$  of  $leafPool > g(LI_{best})$  then
24:    $bestSol = leaf$  in  $leafPool$  with maximum  $g(LI')$ 
25: end if
26: return  $bestChildren$ 

```

---

**4.2 The two-phase constructive algorithm**

We develop a two-phase constructive algorithm for the single container mix-loading problem described in Algorithm 5.

The inputs of the algorithm are the given set of PSUs ( $\mathcal{P}$ ) and the given container ( $C$ ). In the first phase of our constructive algorithm, the stochastic beam-search-based method (Algorithm 1) is called upon to load PSUs in  $\mathcal{P}$  into the given container  $C$ , such that the total volume of loaded complete PSUs is maximized. At the end of the first phase, if all of the given PSUs are loaded into the container or if there is no residual space in the container, the algorithm returns the solution found in the first phase ( $solP1$ ) and terminates; otherwise, the algorithm continues with the second phase. Note that all the unusable residual spaces generated during the beam search are not deleted (see Sect. 4.1.2), so no residual space in the container means no usable or unusable residual space exists in the container.

In the second phase of the algorithm, we first store all the depalletizable PSUs that are not loaded into the container in the set  $D$  and store all the residual spaces in the container in the set  $R$ ; meanwhile, we mark all the unusable residual spaces in  $R$  as usable. Suppose the total volume of all the residual spaces in  $R$  is  $V_R$ . If  $V_R$  is smaller than the smallest carton volume of depalletizable PSU in  $I$ , no PSU in  $I$  can be loaded into any residual space in

**Algorithm 5** The Two-Phase Constructive Method

---

```

TwoPhase ( $\mathcal{P}, C$ )
  //Input:  $\mathcal{P}$ : a set of PSUs
  //       $C$ : the container
1:  $sol = NULL$ 
2:  $I =$  all of the PSUs in  $\mathcal{P}$ 
3:  $R = C$ 
4:  $solP1 = (I_{solP1}, B_{solP1}, R_{solP1}, LI_{solP1}) = SBSM(I, R, \emptyset)$ 
5: if  $I_{solP1} = \emptyset$  or  $R_{solP1} = \emptyset$  then
6:   return  $solP1$ 
7: else
8:    $D =$  all of the depalletizable PSUs in  $I_{solP1}$ 
9:    $R = R_{solP1}, LI = LI_{solP1}$ 
10:   $V_R =$  the total volume of all of the residual spaces in  $R$ 
11:  if  $V_R <$  the smallest  $V_{I_p}, \forall$  PSUs  $p \in D$  then
12:    return  $solP1$ 
13:  else
14:     $lb = 0, ub = 1, factor = 0$ 
15:    while  $ub - lb > 0.01$  do
16:       $I^* = D, R^* = R, LI^* = LI$ 
17:       $factor = \frac{ub+lb}{2}$ 
18:       $I' = SelectCartons(V_R, I^*, factor)$ 
19:       $tempSol = (I_t, B_t, R_t, LI_t) = SBSM(I', R^*, LI^*)$ 
20:      if  $I_t = \emptyset$  then
21:         $lb = factor$ 
22:         $solP2 = (I_{solP2}, B_{solP2}, R_{solP2}, LI_{solP2}) = tempSol$ 
23:      else
24:         $ub = factor$ 
25:      end if
26:    end while
27:  end if
28: end if
29: return  $solP2$ 

```

---

$R$ ; therefore, the algorithm returns the solution  $solP1$  and terminates; otherwise, the second phase continues with a binary search.

The binary search determines the maximum volume of PSUs whose cartons can be loaded into the residual spaces in  $R$ . In each iteration of the binary search, first, Algorithm 6 is called upon to solve the integer programming problem **SC** and depalletize all the PSUs, which are selected by solving **SC**, into cartons.

**Algorithm 6** Select PSUs in  $I^*$  and Depalletize Them into Cartons

---

```

SelectCartons ( $V_R, I^*, factor$ )
  // Input:  $V_R$ : the total volume of a set of residual spaces
  //       $I^*$ : a set of PSUs
  //       $factor$ : a parameter
1:  $I_{cartons} = \emptyset$ 
2:  $I =$  solve the problem SC( $V_R, I^*, factor$ ) using CPLEX
3:  $I_{cartons} =$  all of the cartons depalletized from PSUs in  $I$ 
4: return  $I_{cartons}$ 

```

---

$$\mathbf{SC}(V_R, D, factor) : \text{Maximize } \sum_{p \in D} V_{I_p} x_p \quad (2)$$



$$\text{Subject to } \sum_{p \in D} V_{I_p} x_p \leq factor \times V_R \quad (3)$$

$$x_p \in \{0, 1\}, \forall p \in D \quad (4)$$

Next, the stochastic beam-search-based method is used again to load cartons selected by Algorithm 6 into the residual spaces in  $R$ . If all the cartons can be loaded into the residual spaces in  $R$ , the current best solution to the single container mix-loading problem ( $tempSol$ ) is generated, and we increase  $lb$  of the binary search in the expectation of finding a better solution; otherwise, we decrease  $ub$  of the binary search to increase the chance that the newly depalletized cartons can be totally loaded into the residual spaces in  $R$ .

## 5 Computational experiments

Our two-phase constructive algorithm was implemented as a sequential algorithm in Java (JDK 7 updated 21, 64-bit edition), and no multi-threading was explicitly used. All experiments described in this section were conducted on a computer with an Intel® Xeon(R) Gold 6146 CPU clocked at 3.20 gigahertz with 16 gigabytes of RAM, running a Windows Server 2016 (64-bit) operating system. The commercial integer linear programming solver used was the IBM ILog CPLEX Optimization Studio 12.2 (64-bit) with its default settings.

Since there is no standard benchmark data for the single container mix-loading problem, we generated 70 SCMLP instances based on historical data provided by the audio equipment manufacturer to test the effectiveness of our approach. The details are presented in Sect. 5.1. The stochastic beam-search-based method described in Sect. 4.1 can solve the single container loading problem. Therefore, we test stochastic beam-search-based method on well-known SCLP benchmark instances and report its performance in Sect. 5.2. We demonstrate the performance of the two-phase constructive algorithm for all SCMLP instances in Sect. 5.3.

### 5.1 Test instances generation

We collected 12-month purchase order data from the audio equipment manufacturer in Hong Kong. Most orders consist of  $N$  types of PSUs, with  $N$  coming from the set  $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 11\}$ . Each PSU contains identical products packaged in cartons. Information about each PSU is provided, including the dimensions of the PSU, the dimensions of the cartons in the PSU, and the number of cartons in the PSU. The 40-foot standard container, with dimensions of 12045 mm by 2309 mm by 2379 mm, is used for each order.

We generated 70 SCMLP instances. Each instance is generated as follows. We first set a target value  $V$  for the total volume of PSUs in the instance. The target volume is randomly generated so that it is larger than the volume of the 40-foot standard container. We then uniformly randomly select  $N$  types of PSUs from the orders provided by the manufacturer and initialize their quantities to 1. Next, we uniformly randomly select one type from the  $N$  types of PSUs and increase its quantity by 1. We repeatedly increase the quantity of each type of PSU until the total volume of all PSUs in the instance exceeds the target volume  $V$ . The manufacturer did not provide information regarding which PSUs are depalletizable; therefore, we randomly set some PSU types in the instance to be depalletizable. Furthermore, we suppose that each PSU in the instance can only be rotated along its height, but that any carton contained in the PSU is fully rotatable.

**Table 1** Computational environments

Algo	CPU	RAM	Time limit
BSG-CLP	T420, with 2 quad-processors Intel Xeon, 2.20GHz	8 GB	150 s, 30 s
BSG-VCS	Intel Xeon Quad-Core (x2) server, 2.20 GHz	8 GB	150 s, 30 s
CLA-SS(W)	Intel® Core(TM)2 Duo CPU E8600 (3.33 GHz)	8 GB	–
SBSM	Intel® Xeon(R) Gold 6146 CPU (3.20 GHz)	16 GB	150 s, 30 s

## 5.2 Analysis of the stochastic beam-search based method

The stochastic beam-search-based method (SBSM) (Algorithm 1) can be used to solve the Single Container Loading Problem (SCLP) with the full-support constraint. The input of the stochastic beam-search-based method is the set of items and the container given by the SCLP. Several algorithms are designed to solve the SCLP with the full-support constraint in the literature. We compare our approach with the following state-of-the-art ones. The computational environments of each approach are shown in Table 1.

- **BSG-CLP**: a beam-search-based approach developed by Araya and Riff (2014).
- **CLA-SS(W)**: a container loading algorithm with static stability developed by Ramos et al. (2016).
- **BSG-VCS**: a beam-search-based approach using a new heuristic function for selecting boxes developed by Araya et al. (2017).

These authors tested their algorithms on the well-known benchmark data sets BR1–BR15. BR1–BR7 were generated by Bischoff and Ratcliff (1995). BR8–BR15 were proposed by Davies and Bischoff (1999). There are 100 instances in each of the 15 sets. These data sets can be divided into two groups based on the heterogeneity of items: each instance in BR1–BR7 consists of weakly heterogeneous items, and each instance in BR8–BR15 consists of strongly heterogeneous items.

We ran our method on BR1–BR15 and compared its performance with the abovementioned methods. The time limit is set to 150 s. The comparison is summarized in Table 2, where SBSM is our stochastic beam-search-based method. To test the effect of our stochastic strategy, we implemented another version of our approach by removing the stochastic strategy. In other words, similar to BSG-CLP and BSG-VCS, we select the most promising  $w$  nodes at each level of the tree search process. The modified approach is denoted as BSM. The first column presents the name of the test data set. Columns 150 s and 30 s represent the volume utilization obtained by the corresponding algorithm averaged over 100 instances of the corresponding data set by 150 and 30 s, respectively. The last three rows report the average volume utilization of instances over different sets.

Table 2 shows that our stochastic beam-search-based method outperforms the existing algorithms for the single container loading instances with heterogeneous items. Besides, the results of SBSM are slightly better than those of BSM. Since these instances are well studied in the literature, the improvement of SBSM compared to BSM demonstrates the effect of our stochastic strategy, although the improvement is slight. Besides, the SBSM can get different solutions than BSM by setting different random seeds, which can diversify the search space without losing solution quality.

**Table 2** Comparison between SBSM and the best existing algorithms for the SCLP with the full-support constraint

Set	BSG-CLP		BSG-VCS		CLA-SS(W) 146 s	SBSM		BSM	
	150s	30s	150s	30s		150s	30s	150s	30s
BR1	94.50	94.35	<b>94.74</b>	94.58	93.86	94.65	94.56	94.65	94.56
BR2	95.03	94.84	<b>95.38</b>	95.20	94.55	95.25	95.03	95.25	95.02
BR3	95.17	94.97	<b>95.65</b>	95.38	94.75	95.52	95.20	95.48	95.20
BR4	94.97	94.75	<b>95.48</b>	95.22	94.63	95.34	94.94	95.29	94.93
BR5	94.80	94.57	<b>95.33</b>	95.08	94.38	95.13	94.72	95.09	94.71
BR6	94.65	94.39	<b>95.23</b>	94.89	94.24	94.97	94.62	94.95	94.60
BR7	94.09	93.77	<b>94.66</b>	94.34	93.82	94.49	94.06	94.45	94.07
BR8	93.15	92.71	93.73	93.29	93.16	<b>93.77</b>	93.33	93.76	93.34
BR9	92.53	92.03	<b>93.21</b>	92.67	92.62	93.18	92.74	93.17	92.75
BR10	92.04	91.56	<b>92.65</b>	92.11	92.09	92.64	92.24	92.63	92.24
BR11	91.40	90.89	91.96	91.44	91.56	<b>92.04</b>	91.57	92.01	91.54
BR12	90.92	90.31	91.35	90.74	91.28	<b>91.62</b>	91.18	91.62	91.19
BR13	90.51	89.96	90.92	90.26	90.93	<b>91.08</b>	90.59	91.08	90.54
BR14	89.93	89.34	90.40	89.63	90.38	<b>90.57</b>	90.02	90.55	89.97
BR15	89.33	88.77	89.83	88.95	<b>90.08</b>	89.89	89.43	89.89	89.46
Avg(1-7)	94.74	94.52	<b>95.21</b>	94.96	94.32	95.05	94.73	95.02	94.73
Avg(8-15)	91.23	90.70	91.76	91.14	91.51	<b>91.85</b>	91.39	91.84	91.38
Avg(1-15)	92.87	92.48	<b>93.37</b>	92.92	92.82	93.34	92.95	93.32	92.94

### 5.3 Computational results

We set the stochastic beam search’s time limit to 500s and performed the two-phase constructive algorithm on the 70 SCMLP instances. Table 3 summarizes the final results. Each row in the table reports the instance, the number of PSU types involved in the instance, the volume utilization of the solution found in the first phase of the algorithm ( $Util_1$ ), the volume utilization of the final solution found by the algorithm ( $Util_2$ ), and the number of PSUs that are depalletized in the final solution. The last row is the average result of 70 instances. Note that only complete PSUs are loaded into the container in the solution found in the first phase of the constructive algorithm. Table 3 shows that the volume utilization is increased by approximately 3% by packing cartons into the container.

### 6 Conclusion

We investigated the Single Container Mix-Loading Problem, inspired by the requirements of an international audio equipment manufacturer in Hong Kong. The manufacturer stores its products in Palletized Storage Units (PSUs). When delivering products, loading PSUs, rather than individual products, into containers (or trucks) is convenient; however, large spaces in each container could be wasted. To improve the utilization of a container, the manufacturer is willing to depalletize PSUs and load the individual products, together with other PSUs, into a container. Once a PSU is depalletized, all of its products must be loaded into the container,

**Table 3** Results of performing the two-phase constructive algorithm on the 70 test instances

Instance	PSU Type #	$Util_1$ (%)	$Util_2$ (%)	Depalletized PSUs #
1	1	83.57	85.99	2
2	1	45.49	45.49	0
3	1	77.42	77.42	0
4	1	77.63	80.81	4
5	1	69.45	71.84	2
6	1	61.34	63.88	2
7	1	61.11	63.44	2
8	1	56.97	59.31	2
9	2	83.57	85.99	2
10	2	77.70	80.52	3
11	2	74.83	77.65	2
12	2	83.57	83.57	0
13	2	83.57	86.40	2
14	2	83.20	86.02	2
15	2	80.01	83.20	4
16	2	87.01	90.22	3
17	2	66.69	66.69	0
18	2	70.38	70.38	0
19	2	77.18	77.18	0
20	2	64.65	67.47	2
21	2	76.70	79.47	2
22	2	71.08	72.70	1
23	2	77.78	80.80	2
24	3	83.74	86.56	2
25	3	77.80	77.80	0
26	3	83.57	86.34	2
27	3	80.28	83.46	4
28	3	75.46	78.36	3
29	3	70.52	70.52	0
30	3	74.47	77.40	2
31	3	66.83	69.65	2
32	3	83.77	86.49	2
33	3	74.84	77.67	2
34	3	83.62	83.62	0
35	3	69.45	72.27	2
36	4	83.88	86.77	2
37	4	82.22	85.40	3
38	4	79.00	81.99	3
39	4	78.79	81.80	2
40	4	83.67	86.75	3
41	4	79.22	82.38	3

**Table 3** continued

Instance	PSU Type #	$Util_1$ (%)	$Util_2$ (%)	Depalletized PSUs #
42	4	71.52	74.22	2
43	4	83.04	86.16	2
44	4	81.60	84.50	3
45	4	83.69	86.81	2
46	4	80.38	83.53	2
47	4	82.45	85.63	4
48	4	81.62	84.44	3
49	4	80.92	83.81	2
50	6	80.17	83.38	3
51	6	80.79	83.77	3
52	6	82.83	86.02	3
53	6	78.98	81.68	2
54	6	78.03	81.05	2
55	6	81.32	84.35	2
56	6	76.06	79.19	2
57	6	83.98	87.19	3
58	6	80.77	80.77	0
59	6	81.38	84.51	2
60	6	76.21	79.40	3
61	6	83.12	85.91	2
62	6	80.66	80.66	0
63	7	62.43	62.43	0
64	8	77.96	81.16	3
65	8	71.53	74.42	3
66	8	55.88	55.88	0
67	9	81.83	85.02	2
68	11	66.95	70.16	3
69	11	81.84	84.82	3
70	11	82.16	85.37	4
avg	4.01	76.74	79.11	2.01

and no PSU can be depalletized if the total volume of complete PSUs loaded in the container is not maximized.

We developed a two-phase constructive algorithm for the SCMLP. The algorithm uses a stochastic beam-search-based method as its sub-routine, which iteratively calls upon a beam search. In the first phase of our constructive algorithm, the stochastic beam-search-based method is called upon to load complete PSUs into the container. In the second phase, a proper set of PSUs is selected considering the remaining volume of the container, and the stochastic beam-search-based method is used to load all products depalletized from the selected PSUs into the remaining spaces of the container.

We generated 70 test instances based on historical data provided by the audio equipment manufacturer, and we reported the solutions to the test instances found by the two-phase constructive algorithm for further reference.

**Acknowledgements** This paper is supported by the Science Fund for Distinguished Young Scholars of Guangdong Province (No. 2022B1515020076), the Natural Science Foundation of China (No. 72271062), the Key Program of National Natural Science Foundation of China (Grant No. 71831003), the Fundamental Scientific Research Project of Department of Education of Liaoning (Grant No. LJKMZ20221579), and Dalian Science and Technology Talent Innovation Support Plan (2022RG17).

## Declarations

**Ethical approval** This article contains no studies with human participants or animals performed by any authors.

**Conflict of interest** The authors disclose no financial or conflict of interest.

## References

- Araya, I., Guerrero, K., & Nuñez, E. (2017). Vcs: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research*, *82*, 27–35.
- Araya, I., Moyano, M., & Sanchez, C. (2020). A beam search algorithm for the biobjective container loading problem. *European Journal of Operational Research*, *286*(2), 417–431.
- Araya, I., & Riff, M. C. (2014). A beam search approach to the container loading problem. *Computers & Operations Research*, *43*, 100–107.
- Bischoff, E. E., Janetz, F., & Ratcliff, M. (1995). Loading pallets with non-identical items. *European journal of operational research*, *84*(3), 681–692.
- Bischoff, E. E., & Marriott, M. D. (1990). A comparative evaluation of heuristics for container loading. *European Journal of Operational Research*, *44*(2), 267–276.
- Bischoff, E. E., & Ratcliff, M. (1995). Issues in the development of approaches to container loading. *Omega*, *23*(4), 377–390.
- Bortfeldt, A., & Gehring, H. (1998). Ein Tabu Search-Verfahren für Containerbeladeprobleme mit schwach heterogenem Kistenvorrat. *OR Spectrum*, *20*(4), 237–250.
- Bortfeldt, A., & Gehring, H. (2001). A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, *131*(1), 143–161.
- Bortfeldt, A., Gehring, H., & Mack, D. (2003). A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, *29*(5), 641–662.
- Davies, A. P., & Bischoff, E. E. (1999). Weight distribution considerations in container loading. *European Journal of Operational Research*, *114*(3), 509–527.
- Den Boef, E., Korst, J., Martello, S., et al. (2005). Erratum to “the three-dimensional bin packing problem”: Robot-packable and orthogonal variants of packing problems. *Operations Research*, *53*(4), 735–736.
- Eley, M. (2002). Solving container loading problems by block arrangement. *European Journal of Operational Research*, *141*(2), 393–409.
- Fanslau, T., & Bortfeldt, A. (2010). A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, *22*(2), 222–235.
- Fekete, S. P., & Schepers, J. (1997). On more-dimensional packing I: Modeling. Technical report, Mathematisches Institut, Universität zu Kiel.
- Fekete, S. P., & Schepers, J. (1997). On more-dimensional packing II: Bounds. Technical report, Mathematisches Institut, Universität zu Kiel.
- Fekete, S. P., & Schepers, J. (2004). A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, *29*(2), 353–368.
- Fekete, S. P., & Schepers, J. (2004). A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, *60*(2), 311–329.
- Fekete, S. P., Schepers, J., & Van der Veen, J. C. (2007). An exact algorithm for higher-dimensional orthogonal packing. *Operations Research*, *55*(3), 569–587.
- Gehring, H., & Bortfeldt, A. (1997). A genetic algorithm for solving the container loading problem. *International transactions in operational research*, *4*(5–6), 401–418.

- Gehring, H., & Bortfeldt, A. (2002). A parallel genetic algorithm for solving the container loading problem. *International Transactions in Operational Research*, 9(4), 497–511.
- George, J. A., & Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3), 147–156.
- Gilmore, P., & Gomory, R. E. (1966). The theory and computation of knapsack functions. *Operations Research*, 14(6), 1045–1074.
- He, K., & Huang, W. (2011). An efficient placement heuristic for three-dimensional rectangular packing. *Computers & Operations Research*, 38(1), 227–233.
- Hifi, M. (2004). Exact algorithms for unconstrained three-dimensional cutting problems: A comparative study. *Computers & Operations Research*, 31(5), 657–674.
- Hifi, M., & Zissimopoulos, V. (1996). A recursive exact algorithm for weighted two-dimensional cutting. *European Journal of Operational Research*, 91(3), 553–564.
- Huang, W., & He, K. (2009). A caving degree approach for the single container loading problem. *European Journal of Operational Research*, 196(1), 93–101.
- Kurpel, D. V., Scarpin, C. T., Junior, J. E. P., et al. (2020). The exact solutions of several types of container loading problems. *European Journal of Operational Research*, 284(1), 87–107.
- Lim, A., Ma, H., Qiu, C., et al. (2013). The single container loading problem with axle weight constraints. *International Journal of Production Economics*, 144(1), 358–369.
- Lim, A., Rodrigues, B., & Wang, Y. (2003). A multi-faced buildup algorithm for three-dimensional packing problems. *Omega*, 31(6), 471–481.
- Mack, D., Bortfeldt, A., & Gehring, H. (2004). A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, 11(5), 511–533.
- Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research*, 48(2), 256–267.
- Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. Wiley.
- Morabito, R., & Arenales, M. (1994). An and/or-graph approach to the container loading problem. *International Transactions in Operational Research*, 1(1), 59–73.
- Moura, A., & Oliveira, J. F. (2005). A grasp approach to the container-loading problem. *IEEE Intelligent Systems*, 20(4), 50–57.
- Ngoi, B., Tay, M., & Chua, E. (1994). Applying spatial representation techniques to the container packing problem. *The International Journal of Production Research*, 32(1), 111–123.
- Ngoi, B. K. A., & Whybrew, K. (1993). A fast spatial representation method (applied to fixture design). *The International Journal of Advanced Manufacturing Technology*, 8(2), 71–77.
- Parreño, F., Alvarez-Valdés, R., Tamarit, J. M., et al. (2008). A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, 20(3), 412–422.
- Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2), 382–392.
- Ramos, A. G., Oliveira, J. F., Gonçalves, J. F., et al. (2016). A container loading algorithm with static mechanical equilibrium stability constraints. *Transportation Research Part B: Methodological*, 91, 565–581.
- Sheng, L., Hongxia, Z., Xisong, D., et al. (2016). A heuristic algorithm for container loading of pallets with infill boxes. *European Journal of Operational Research*, 252(3), 728–736.
- Silva, E. F., Toffolo, T. A. M., & Wauters, T. (2019). Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. *Computers & Operations Research*, 109, 12–27.
- Terno, J., Scheithauer, G., Sommerweiß, U., et al. (2000). An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research*, 123(2), 372–381.
- Wang, N., Lim, A., & Zhu, W. (2013). A multi-round partial beam search approach for the single container loading problem with shipment priority. *International Journal of Production Economics*, 145(2), 531–540.
- Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3), 1109–1130.
- Wei, L., Zhu, W., & Lim, A. (2015). A goal-driven prototype column generation strategy for the multiple container loading cost minimization problem. *European Journal of Operational Research*, 241(1), 39–49.
- Zhang, D., Peng, Y., & Leung, S. C. (2012). A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & Operations Research*, 39(10), 2267–2276.
- Zhao, X., Bennell, J. A., Bektaş, T., et al. (2016). A comparative review of 3d container loading algorithms. *International Transactions in Operational Research*, 23(1–2), 287–320.
- Zhu, W., & Lim, A. (2012). A new iterative-doubling greedy-lookahead algorithm for the single container loading problem. *European Journal of Operational Research*, 222(3), 408–417.



Zhu, W., Oon, W. C., Lim, A., et al. (2012). The six elements to block-building approaches for the single container loading problem. *Applied Intelligence*, 37(3), 431–445.

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.