European Journal of Operational Research 252 (2016) 728–736

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A heuristic algorithm for container loading of pallets with infill boxes

Liu Sheng[a,b,*], Zhao Hongxia[a,b], Dong Xisong[a,b], Cheng Changjian[a,b]

[a] State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
[b] Qingdao Academy of Intelligent Industry, Qingdao 266000, China

ARTICLE INFO

Article history:
Received 26 August 2013
Accepted 16 January 2016
Available online 25 January 2016

Keywords:
Packing
Container loading
Heuristic algorithm
Tree search
Greedy algorithm

ABSTRACT

We consider the container loading problem that occurs at many furniture factories where product boxes are arranged on product pallets and the product pallets are arranged in a container for shipments. The volume of products in the container should be maximized, and the bottom of each pallet must be fully supported by the container floor or by the top of a single pallet to simplify the unloading process. To improve the filling rate of the container, the narrow spaces at the tops and sides of the pallets in the container should be filled with product boxes. However, it must be ensured that all of the infill product boxes can be entirely palletized into complete pallets after being shipped to the destination. To solve this problem, we propose a heuristic algorithm consisting of a tree search sub-algorithm and a greedy sub-algorithm. The tree search sub-algorithm is employed to arrange the pallets in the container. Then, the greedy sub-algorithm is applied to fill the narrow spaces with product boxes. The computational results on BR1–BR15 show that our algorithm is competitive.

1. Introduction

Cutting and packing (Dyckhoff & Finke, 1992; Wäscher, Haußner, & Schumann, 2007) are two classic combinatorial optimization problems. Cutting problems address the best possible utilization of materials, such as wood, steel and cloth, whereas packing problems address the best possible capacity use of packing space. The effective use of material and transport capacities is of great economic importance in production and distribution processes. It also contributes to the economical utilization of natural resources.

According to Wäscher et al. (2007), cutting and packing problems have an identical structure in common. They can be summarized as follows:

First, a set of large objects and a set of small items are given. The large objects and the small items are defined exhaustively in one, two, three or an even larger number of geometric dimensions. Select some or all of the small items, group them into one or more subsets and assign each of the resulting subsets to one of the large objects such that the geometric condition holds, *i.e.*

- all small items of the subset lie entirely within the large object, and
- the small items do not overlap,

and a given objective function is optimized.

Container loading problems are sub-problems of the cutting and packing problems. In Bortfeldt and Wäscher (2013), container loading problems are interpreted as geometric assignment problems, in which three-dimensional small items (called cargo) have to be assigned (packed into) to three-dimensional, rectangular (cubic) large objects (called containers) such that a given objective function is optimized and two basic geometric feasibility conditions hold, *i.e.*

- all small items lie entirely within the container, and
- the small items do not overlap.

In this paper, we consider a container loading problem for pallets with infill boxes (CLPIB), which is a special container loading problem case. Given an empty rectangular container and m rectangular product pallets, we determine a subset of pallets with maximal volume that can be placed in the container. Each product pallet contains a single type of rectangular product boxes. Some product pallets can be divided into product boxes for transportation. The product boxes can be placed in the narrow spaces between the pallets and the container after a container is filled with pallets. After the container is shipped to its destination, these product boxes should be exactly palletized onto their original

[*] Corresponding author at: State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China. Tel.: +86 15910634676.
E-mail address: liusheng7801@163.com, sheng.liu@ia.ac.cn (L. Sheng).

http://dx.doi.org/10.1016/j.ejor.2016.01.025
0377-2217/© 2016 Elsevier B.V. All rights reserved.

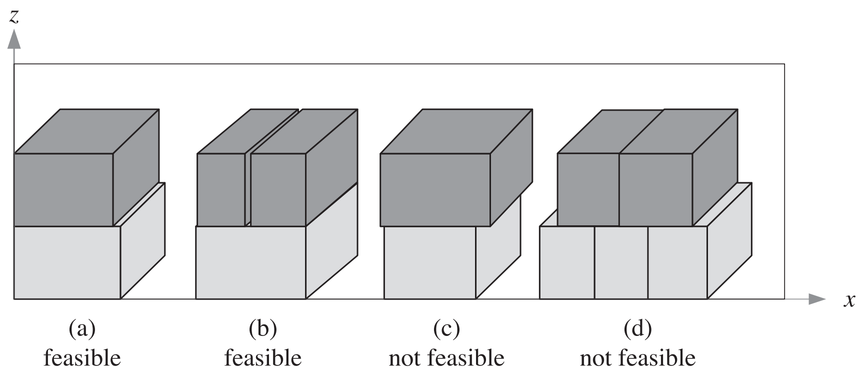

Fig. 1. Four pallet arrangements in a container.

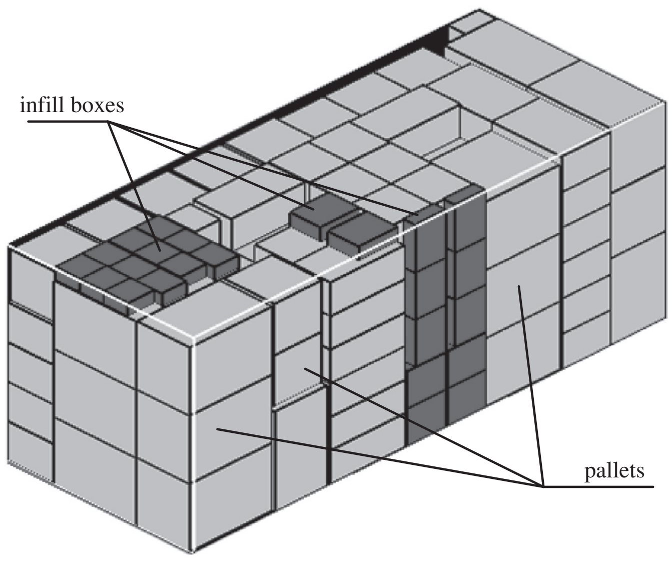

Fig. 2. An example solution for CLPIB.

pallets. The pallets must be placed with their bottoms parallel to the container, whereas the product boxes can be placed in all six orthogonal orientations. Additionally, the bottoms of each pallet must be fully supported by the container floor or by the top of a single pallet to simplify the unloading process and to ensure the stability of the pallets. The product boxes can be easily fastened on the pallets by adhesive tapes because they are relatively small and light. Thus they do not need to be fully supported.

Fig. 1 lists four pallet arrangements in a container. The arrangements in Fig. 1(a) and (b) are feasible, whereas the arrangements in Fig. 1(c) and (d) are not feasible.

An example solution for CLPIB is illustrated in Fig. 2.

The rest of this paper is divided as follows: Section 2 provides an overview of the literature, Section 3 presents the approach for CLPIB, Section 4 describes the computational experiments and presents the results, and Section 5 summarizes the paper.

2. Literature overview

Bortfeldt and Wäscher (2013) introduced a scheme to categorize the constraints of loading a container for the first time and found that the existing approaches have limited practical value because they do not pay sufficient attention to the constraints encountered in practice. The problem discussed in this paper is a knapsack container loading problem with several practical constraints (listed in the Section 1). There are no published approaches that address the container loading problem with these practical constraints. Because it is a knapsack container loading problem, we will briefly discuss some of the recent advances in three-dimensional container loading.

The three-dimensional container loading problem (3D-CLP) can be broadly characterized as type 3/B/O/-, according to the typology presented by Dyckhoff and Finke (1992), or type 3D-R-IIPP/SLOPP/SKP, according to the typology presented by Wäscher et al. (2007). A consignment of goods wrapped up in boxes is assumed to be loaded into a single container of known dimensions, and the boxes and containers are assumed to have a rectangular shape (Junqueira, Morabito, & Sato Yamashita, 2012).

3D-CLP is a typical NP-hard problem (Bischoff & Marriott, 1990) that cannot be solved optimally by an algorithm in polynomial time. When the number of box types increases, exact algorithms are usually confronted with a situation called “combinatorial explosion”. Thus, they can only solve problems with a weak heterogeneous box set. As a result, heuristic methods are usually the first selection for addressing the three-dimensional container loading problem. Heuristic methods may not obtain the best of all of the actual solutions to 3D-CLP, but they can usually produce sufficiently good solutions within acceptable times. Researchers have provided various heuristic methods.

Heuristic methods for 3D-CLP can be divided into two groups according to the method type.

(1) *Tree search methods:* Tree search or graph search methods were successfully utilized in 3D-CLP. Morabito and Arenales (1994) suggested an And/Or graph search method. Eley (2002) tried to fill the container with homogeneous blocks made up of identical items. Hifi (2002) presented a tree search method using hill-climbing strategies. Pisinger (2002) proposed an algorithm that first divides the whole container space into several vertical layers, then divides the layers into a number of horizontal or vertical strips and then generates the strips by solving the one-dimensional knapsack problem. Bortfeldt and Mack (2007) presented a heuristic algorithm that was derived from a branch-and-bound approach. Fanslau and Bortfeldt (2010) proposed an effective tree search algorithm based on the idea of a composite block. Zhang, Peng, and Leung (2012) designed a heuristic block-loading algorithm based on a multi-layer search. Liu, Tan, Xu, and Liu (2014) presented a heuristic wall-building algorithm. Araya and Riff (2014) proposed a beam search approach to the container loading problem.

(2) *Non-tree search methods:* Non-tree search methods include classic heuristic methods and intelligent heuristic methods. The former methods for solving 3D-CLP were presented by Bischoff and Ratcliff (1995), Bischoff, Janetz, and Ratcliff (1995), and Lim, Rodrigues, and Wang (2003), whereas the

latter methods have been the most used method types for 3D-CLP in recent years. Gehring and Bortfeldt (1997, 2002), Hemminki (1994), and Bortfeldt and Gehring (2001) utilized genetic algorithms (GAs). Sixt (1996) and Mack, Bortfeldt, and Gehring (2004) provided simulated annealing methods (SAs). Bortfeldt and Gehring (1998), Sixt (1996), and Bortfeldt, Gehring, and Mack (2003) suggested tabu search algorithms (TSs). Faroe, Pisinger, and Zachariasen (2003) and Mack et al. (2004) suggested local search methods. Moura and Oliveira (2005) and Parreno, Alvarez-Valdes, Oliveira, and Tamarit (2007) introduced a greedy randomized adaptive search procedure (GRASP).

In addition to classification according to method type, Pisinger (2002) grouped the methods into five classes according to the packing approaches: the wall building approach (suggested by Bortfeldt & Gehring, 2001; George & Robinson, 1980; Pisinger, 2002); the block building approach (representatives of this approach are the TS methods from Bortfeldt et al., 2003; the tree search methods from Eley, 2002; Fanslau & Bortfeldt, 2010; Zhang et al., 2012; Zhu & Lim, 2012; and the SA/TS hybrid methods from Mack et al., 2004); the horizontal layer building approach (realized by Bischoff et al., 1995; Terno, Scheithauer, Sommerweiß, & Rieme, 2000); the stack building approach (presented by Bischoff & Ratcliff, 1995; Gehring & Bortfeldt, 1997); and the guillotine cutting approach (mixed with the graph search method by Morabito and Arenales (1996)). Otherwise, Huang and He (2009) and He and Huang (2011) proposed heuristic algorithms based on the idea of caving degree. Lu and Cha (2014) proposed an algorithm on how to pack small items onto a large rectangular pallet. Zhu, Huang, and Lim (2012) proposed a prototype column generation strategy for the multiple container loading problem. Wei, Zhu, and Lim (2015) presented a goal-driven prototype column generation strategy for the multiple container loading cost minimization problem. Tian, Zhu, Lim, and Wei (2016) developed a two-phase algorithm to solve the multiple container loading problem with preference.

The great majority of methods mentioned above obey the orientation constraint and the support constraint as well. Several mentioned methods also include additional constraints from the packing context in the problem, *e.g.,* a weight constraint for the freight (Bortfeldt & Gehring, 2001; Terno et al., 2000; Lim, Ma, Qiu, & Zhu, 2013). The shipment priority constraint was considered by Ren, Tian, and Sawaragi (2011) and Wang, Lim, and Zhu (2013).

The remarkable paper by Egeblad, Garavelli, Lisi, and Pisinger (2010) is one case study of and an application related to practical container loading problems. It represents the type of publications that are concerned with the systematic integration of several types of constraints into solution approaches. The cargoes that will be loaded consist of a mixture of regular and irregular items that require satisfying specific stacking and orientation constraints.

3. Our method for CLPIB

In practical applications, the pallets must be placed with their bottoms parallel to the bottom of the container. To compare our method with existing algorithms, we assume that the pallets may have at most six admissible orthogonal orientations. It is easy to adjust the orientation constraints to fit practical applications.

As shown in Fig. 3, a container is placed in the first octant of a 3D coordinates system (3D-CS). Let

$$C = (L, W, H) \tag{1}$$

denote a container. Symbols L, W and H denote the container length, width and height, respectively.

Let

$$P = \{p_1, p_2, \ldots, p_m\} \tag{2}$$

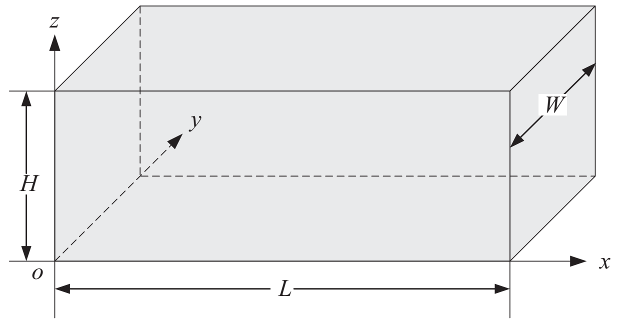

Fig. 3. Container in 3D coordinates system.

be the product pallet set that contains m pallets. p_i is the ith pallet in P that is defined as

$$p_i = (l_i, w_i, h_i, \alpha_i, \beta_i, \gamma_i, d_i, bl_i, bw_i, bh_i) \tag{3}$$

where l_i, $w_i$and h_i are the length, width and height of p_i, respectively. The implications of α_i, β_i, γ_i and d_i are listed:

$$\alpha_i = \begin{cases} 1, & \text{if } p_i \text{ can be placed with } l_i \text{ paralled to } H \\ 0, & \text{otherwise} \end{cases},$$

$$\beta_i = \begin{cases} 1, & \text{if } p_i \text{ can be placed with } w_i \text{ paralled to } H \\ 0, & \text{otherwise} \end{cases},$$

$$\gamma_i = \begin{cases} 1, & \text{if } p_i \text{ can be placed with } h_i \text{ paralled to } H \\ 0, & \text{otherwise} \end{cases}, \quad \text{and}$$

$$d_i = \begin{cases} 1, & \text{if } p_i \text{ can be divided into product boxes} \\ 0, & \text{otherwise} \end{cases}.$$

The symbols bl_i, bw_i and bh_i are the length, width and height of each product box in p_i, respectively.

As described in Section 1, CLPIB consists of two sequential sub-problems that regard how to load the product pallets into the container and how to fill the narrow spaces between the product pallets and the container with product boxes, respectively. The first sub-problem can be written as:

$$\max \left\{ Z_Q = \sum_{i=1}^{m} q_i l_i w_i h_i | q_i \in \{0, 1\} \ i = 1, 2, \ldots, m \right\}$$
$$q_i = \begin{cases} 1, & \text{if } p_i \text{ is placed as a whole in the container} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

whereas the second sub-problem can be expressed as:

$$\max \left\{ Z_R = \sum_{i=1}^{m} r_i l_i w_i h_i | r_i \in \{0, 1\} r_i \le 1 - q_i, \right.$$
$$\left. d_i = 1 \ i = 1, 2, \ldots, m \right\}$$
$$q_i = \begin{cases} 1, & \text{if } p_i \text{ is placed as product boxes in the container} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

In both Formulas (4) and (5), two basic geometric constraints must be satisfied:

- all small items lie entirely within the container, and
- the small items do not overlap.

A feasible solution for CLPIB is denoted as $(Q = \{q_1, q_2, \ldots, q_m\}, R = \{r_1, r_2, \ldots, r_m\})$.

We design a heuristic algorithm for CLPIB which is referred to as HCLPIB. HCLPIB consists of a tree search sub-algorithm that solves the sub-problem defined in Formula (4) and a greedy sub-algorithm that solve the sub-problem defined in Formula (5). The

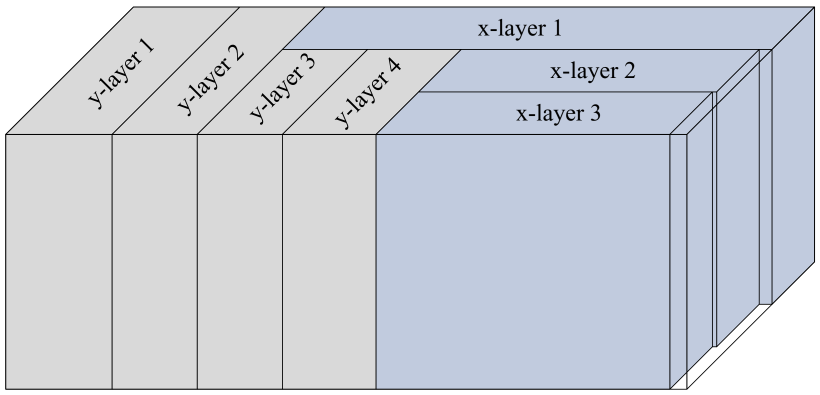

Fig. 4. An example solution for QTS.

tree search sub-algorithm is referred to as QTS (Quaternary Tree Search), whereas the greedy sub-algorithm is referred to as GIB (Greedy Algorithm for Infill Boxes). QTS and GIB will be described in detail in Sections 3.1 and 3.2, respectively.

3.1. QTS – the tree search sub-algorithm of HCLPIB

QTS places multiple pallets into the container by using the wall building strategy. Thus, the obtained container loading plan consists of a set of pallet layers, and each pallet layer consists of a set of pallet strips. The pallets in a strip are stacked along a line parallel to the *z*-axis in 3D-CS. The strips in a layer are arranged along a line that is parallel to the *x*-axis or *y* axis in 3D-CS. If the strips in a layer are along a line parallel to the *x*-axis, the layer is called the x-layer. Otherwise the layer is called the y-layer. The surfaces of each pallet must be parallel to one of the three planes: *xy*, *xz* and *yz* in 3D-CS. An example solution for QTS is shown in Fig. 4.

Let *ls* and *ws* be the length and width (generally $ls \geq ws$) of a pallet strip *s* (see Fig. 5(a)). The filling rate of *s* is defined as:

$$\mathrm{FRS}(s) = \sum_{i=1}^{m} l_i w_i h_i q_i / (ls * ws * H)$$

$$q_i = \begin{cases} 1, & \text{if } \ p_i \text{ is placed in } s \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, m \tag{6}$$

Let *ll* and *thickl* denote the length and width of layer *l*, respectively.

Let

$$R = (lr, wr, H) \tag{7}$$

denote the rectangular residual space of a container. The symbols *lr*, *wr* and *H* are its length, width and height, respectively. If a residual space is not completely rectangular, we consider the length, width and height of the maximum cuboid inside it as its length, width and height, respectively. When no layers are placed in the container, $R = (L, W, H)$. When an x-layer, which is shown in Fig. 5(b), is placed in an empty container, $R = (L, W - thickl, H)$. When a y-layer, which is shown in Fig. 5(c), is placed in an empty container, $R = (L - thickl, W, H)$.

If *l* is the x-layer, which is shown in Fig. 5(b), and *l* is placed in the residual space (lr, wr, H), the filling rate of *l* is defined as:

$$\mathrm{FRL}(l) = \sum_{i=1}^{m} l_i w_i h_i q_i / (lr * thickl * H)$$

$$q_i = \begin{cases} 1, & \text{if } \ p_i \text{ is placed in } l \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, m \tag{8}$$

If *l* is the y-layer, which is shown in Fig. 5(c), and *l* is placed in the residual space (lr, wr, H), the filling rate of *l* is defined as:

$$\mathrm{FRL}(l) = \sum_{i=1}^{m} l_i w_i h_i q_i / (wr * thickl * H)$$

$$q_i = \begin{cases} 1, & \text{if } \ p_i \text{ is placed in } l \\ 0, & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, m \tag{9}$$

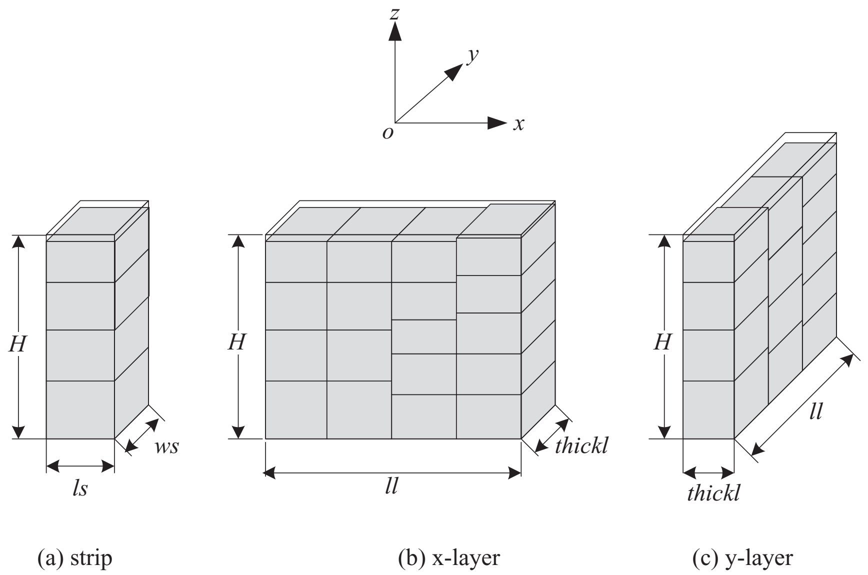

Fig. 5. Strip, x-layer and y-layer.

Let

$$S = \{s_1, s_2, \ldots, s_n\} \tag{10}$$

be a strip sequence, which contains n strips.

The length and width of a strip are determined before the strip is created. Length *ls* and width *ws* are always equal to two dimensions of the length, width and height of one pallet in *P*, respectively. To explain how to create strips one by one using *P*, we introduce several definitions.

$$H\alpha_i(ls,\ ws) = \begin{cases} l_i, & \text{if } \ (max\{w_i, h_i\} \le ls,\ min\{w_i, h_i\} \le ws, \\ & \quad \text{and } \alpha_i = 1) \\ +\infty, & \text{otherwise} \end{cases} \tag{11.1}$$

Eq. (11.1) means the height of the envelope cuboid of p_i in a strip with length *ls* and width *ws* if the length of p_i is parallel to the *z*-axis.

$$H\beta_i(ls,\ ws) = \begin{cases} w_i, & \text{if } \ (max\{l_i, h_i\} \le ls,\ min\{l_i, h_i\} \le ws, \\ & \quad \text{and } \beta_i = 1) \\ +\infty, & \text{otherwise} \end{cases} \tag{11.2}$$

Eq. (11.2) means the height of the envelope cuboid of p_i in a strip with the length *ls* and width *ws* if the width of p_i is parallel to the *z*-axis.

$$H\gamma_i(ls,\ ws) = \begin{cases} h_i, & \text{if } \ (max\{l_i, w_i\} \le ls,\ min\{l_i, w_i\} \le ws, \\ & \text{and } \gamma_i = 1) \\ +\infty, & \text{otherwise} \end{cases} \tag{11.3}$$

Eq. (11.3) means the height of the envelope cuboid of p_i in a strip with the length *ls* and width *ws* if the height of p_i is parallel to the *z*-axis. The value $+\infty$ in Formulas (11.1)–(11.3) means that the corresponding orientations are prohibited or that the pallet exceeds the boundary of the envelope cuboid of the strip.

Therefore, it is obvious that

$$H_i(ls, ws) = \min\{H\alpha_i(ls, ws), H\beta_i(ls, ws), H\gamma_i(ls, ws)\} \tag{12}$$

is the height of the envelope cuboid of p_i in a strip with length *ls* and width *ws*.

Let

$$\mathrm{KS}^{\mathrm{strip}}(H, ls, ws) = \max\left\{\sum_{i=1}^{m} l_i w_i h_i q_i \;\middle|\; \begin{array}{l} Q = \{q_1, q_2, \ldots, q_m\}\ is\ a\ feasible\ solution, \\ q_i \in \{0, 1\}, \sum_{i=1}^{m} q_i * H_i(ls,\ ws) \le H,\ i = 1, 2, \ldots, m \end{array}\right\} \tag{13}$$

denote a one-dimensional knapsack model to generate a strip with length *ls* and width *ws*, where q_i means that p_i is placed in the strip ($q_i = 1$) or is not placed in the strip ($q_i = 0$).

When we group a pallet set into a strip sequence *S*, we utilize two of the three sizes (length, width and height) of each pallet as the length and width to create a strip. Thus, we obtain a candidate strip sequence *S_CAN*. Then, we select a strip from *S_CAN* and insert it into *S*. If we select the first (or the last or the middle) strip with a filling rate higher than a given value (called strip filling rate threshold, *sfrt* for short, usually in [0.9, 1]) from *S_CAN*, we can usually obtain a better solution than by selecting the strip with the highest filling rate. The three strip selecting types (*sst* for short) are defined as “FIRST”, “LAST” and “MID”. Therefore, we define the function:

$$\mathrm{SelectStrip}^{sst}_{\ge sfrt}(S\_CAN) \tag{14}$$

If *sst* = “FIRST”, Formula (14) returns the first strip whose filling rate is not less than *sfrt* in *S_CAN*. If *sst* = “LAST”, Formula (14) returns the last strip whose filling rate is not less than *sfrt* in *S_CAN*. If *sst* = “MID”, Formula (14) returns the middlemost strip whose filling rate is not less than *sfrt* in *S_CAN*. If no strip exists in *S_CAN* whose filling rate is not less than *sfrt*, Formula (14) returns the strip with the highest filling rate in *S_CAN*.

If we create a layer from a strip sequence *S*, the thickness of a layer is always equal to the length or width of one strip in *S*. To explain how to create a layer using the strips in *S*, several definitions are presented.

$$L(s_i,\ thickl) = \left\{\begin{array}{ll} ls_i, & \text{if } \ (ls_i > thickl \text{ and } ws_i \le thickl) \\ ws_i, & \text{if } \ (ls_i \le thickl \text{ and } ws_i \le thickl) \\ +\infty & \text{otherwise} \end{array}\right\} \tag{15}$$

means the length of the envelope cuboid of the strip s_i in a layer with thickness *thickl*. The strip must be placed such that it does not exceed the boundary of the layer. The value $+\infty$ means that s_i cannot be placed in a layer with thickness *thickl*.

Let

$$\mathrm{KS}^{\mathrm{layer}}(H,\ ll,\ thickl) = \max\left\{\sum_{i=1}^{n} V_i \;\middle|\; \begin{array}{l} Q = \{q_1, q_2, \ldots, q_n\}\ is\ a\ feasible\ solution,\ q_i \in \{0, 1\}, \\ \sum_{i=1}^{n} q_i * L(s_i, thickl) \le ll,\ i = 1, 2, \ldots, n \end{array}\right\} \tag{16}$$

denote a one-dimensional knapsack model to generate a layer with length *ll* and thickness *thickl*. Herein, V_i is the total volume of the pallets in the strip s_i and q_i indicates whether s_i is included in the layer (0 not included; 1 included).

The mathematical model

$$\mathrm{LAYER}_x(lr, S) = \{\mathrm{KS}^{\mathrm{layer}}(H, lr, ls_i), \mathrm{KS}^{\mathrm{layer}}(H, lr, ws_i) | i = 1, 2, \ldots, n\} \tag{17}$$

creates a set of candidate x-layers whose lengths equal to the length of the current residual space. *lr* is the length of the current residual space.

The mathematical model

$$\mathrm{LAYER}_y(wr, S) = \left\{\mathrm{KS}^{\mathrm{layer}}(H, wr, ls_i), \mathrm{KS}^{\mathrm{layer}}(H, wr, ws_i) | i = 1, 2, \ldots, n\right\} \tag{18}$$

creates a set of candidate y-layers whose lengths equal to the width of the current residual space. *wr* is the width of the current residual space.

The mathematical model

$$L2 = \mathrm{SORT}(L1) = \{l2_1, l2_2, l2_3, \ldots\} \tag{19}$$

sorts the layers in the layer set *L1* and obtains a layer sequence *L2* such that $\mathrm{FRL}(l2_1) \ge \mathrm{FRL}(l2_2) \ge \mathrm{FRL}(l2_3) \ge \cdots$.

QTS loads pallets into a container with two different priority styles. For the first style, QTS loads undividable and dividable pallets with the same priority. For the second style, QTS loads undividable pallets prior to the dividable pallets. We call the first style “SAME_PRIORITY” and the second style “DIFFERENT_PRIORITY”.

The main procedure for QTS is described in Procedure 1. *L*, *W*, *H*, *P*, and *prior_style* represent the length, width, and height of the container, the pallet set, and the priority style, respectively. We find that we can keep a good balance between the solution quality and the computation time if we use *SFRT* = {0.9,0.904,0.908,0.912,…,1} as the *sfrt* value set. We consider all three *sst* values, which are “FIRST”, “LAST” and “MID” in QTS. The invoked procedure FillCuboidSpace is described in Procedure 2.

Procedure 1

```
QTS ($L, W, H, P$, $prior\_style$)
    // $L\_temp$ and $L\_best$ are 2 layer sets, $V_L$ and $UV_L$ denote the volume of the
    // pallets and the volume of the undividable pallets in the layer set $L$, respectively
    for each $sfrt \in \{0.9, 0.904, 0.908, 0.912, \cdots, 1\}$
        for each $sst \in$ {"FIRST", "LAST", "MID"}
            $L\_emp :=$ FillCuboidSpace ($L, W, H, B, sst, sfrt, prior\_style$)
            if ($prior\_style$ = "SAME_PRIORITY")
                if($V_{L\_temp} > V_{L\_best}$)
                    $L\_best := L\_temp$
            else if ($prior\_style$ = "DIFFERENT_PRIORITY")
                if ($UV_{L\_temp} > UV_{L\_best}$ or ($UV_{L\_temp} = UV_{L\_best}$ and $V_{L\_temp} > V_{L\_best}$))
                    $L\_best := L\_temp$
    return $L\_best$
```

Procedure 2

```
FillCuboidSpace ($lr, wr, H, P, sst, sfrt, prior\_style$)
    if (no pallets in $P$ can be placed in the residual space $lr \times wr \times H$)
        return $\emptyset$
    // $L1, L2, L3$ and $L4$ are 4 layer sets
    // $XL = \{xl_1, xl_2, \cdots\}$ and $YL = \{yl_1, yl_2, \cdots\}$ are 2 layer sequences
    $XL :=$ CreateCandidateLayers ($lr, wr, H, P, sst, sfrt$, "X_LAYER")
    $YL :=$ CreateCandidateLayers ($wr, lr, H, P, sst, sfrt$, "Y_LAYER")
    $L1 := xl_1 +$ FillCuboidSpace ($lr, wr -$ thickness of $xl_1, H, P -$ pallets in $xl_1$,
        $sst, sfrt, prior\_style$)
    $L2 := yl_1 +$ FillCuboidSpace ($lr -$ thickness of $yl_1, wr, H, P -$ pallets in $yl_1$,
        $sst, sfrt, prior\_style$)
    if (FRL ($xl_2$) $\geq$ FRL ($yl_1$))
        $L3 := xl_2 +$ FillCuboidSpace ($lr, wr -$ thickness of $xl_2, H, P -$ pallets in $xl_2$,
        $sst, sfrt, prior\_style$)
    if (FRL ($yl_2$) $\geq$ FRL ($xl_1$))
        $L4 := yl_2 +$ FillCuboidSpace ($lr -$ thickness of $yl_2, wr, H, P -$ pallets in $yl_2$,
        $sst, sfrt, prior\_style$)
    if ($prior\_style$ = "SAME_PRIORITY")
        return the one in $\{L1, L2, L3, L4\}$ with the highest pallet volume
    else if ($prior\_style$ = "DIFFERENT_PRIORITY")
        return the one in $\{L1, L2, L3, L4\}$ with the highest undividable pallet volume
            or return the one with the highest pallet volume in a subset of $\{L1, L2, L3, L4\}$
            where the layers have the same highest undividable pallet volume
```

Procedure 3

```
CreateCandidateLayers ($ll, max\_thickl, H, P, sst, sfrt, layer\_type$)
    // $S\_C$ and $S\_D$ are 2 strip sequences, $s$ is a strip
    // $L\_SET$ is a layer set, $L\_SEQ$ is a layer sequence
    while ($P \neq \emptyset$)
        $S\_C :=$ CreateCandidateStrips ($ll, thickl, H, P$)
        $s :=$ SelectStrip$^{sst}_{\geq sfrt}$ ($S\_C$)
        $S\_D := S\_D + s$
        remove the pallets in $s$ from $P$
    if ($layer\_type$ = "X_LAYER")    $L\_SET :=$ LAYER$_x$ ($ll, S\_D$)
    elseif ($layer\_type$ = "Y_LAYER")    $L\_SET :=$ LAYER$_y$ ($ll, S\_D$)
    $L\_SEQ :=$ SORT ($L\_SET$)
    return $L\_SEQ$
```

As shown in Procedure 2, FillCuboidSpace is invoked recursively to create and place new layers into the residual space of the container. The symbols *lr*, *wr* and *H* are the length, width and height of the residual space. The symbols *P*, *sst*, *sfrt*, and *prior_style* are the pallet set, strip selecting type, strip filling rate threshold, and the priority style, respectively. If the residual space ($lr \times wr \times H$) cannot accommodate any pallet in *P*, Procedure 2 returns an empty layer set. Otherwise, an x-layer sequence and a y-layer sequence are created using CreateCandidateLayers (described in Procedure 3), respectively. The first x-layer and the first y-layer will be placed in the container. If the filling rate of the second x-layer is no less than the filling rate of the first y-layer, the second x-layer is considered. If the filling rate of the second y-layer is no less than the filling rate of the first x-layer, the second y-layer is considered. In the end, Procedure 2 returns the layer set with the maximum pallet volume or with the maximum undividable pallet volume.

Procedure 4

```
CreateCandidateStrips($lr, wr, H, P$)
    //$S$ is a strip sequence, $s$ is a strip
    for each pallet $p_i$ in $P$
        if($p_i$ can be placed in the space $lr \times wr \times H$ with $h_i \parallel H$)
            $s :=$ KS$^{strip}$($H, l_i, w_i$); $S := S + s$
        if($p_i$ can be placed in the space $lr \times wr \times H$ with $l_i \parallel H$)
            $s :=$ KS$^{strip}$($H, w_i, h_i$); $S := S + s$
        if($p_i$ can be placed in the space $lr \times wr \times H$ with $w_i \parallel H$)
            $s :=$ KS$^{strip}$($H, l_i, h_i$); $S := S + s$
    return $S$
```

Procedure 5

```
GIB($P, C\_SPACE$)
    while($P$ contains at least 1 pallet that can be divided and placed in $C\_SPACE$)
        select the pallet $p$ that can be divided and placed in $C\_SPACE$ with largest volume
        divide $p$ and palce the obtained product boxes into $C\_SPACE$
        remove used cuboid spaces from $C\_SPACE$
        remove $p$ from $P$
    return null
```

CreateCandidateLayers (Procedure 3) creates a sequence of candidate layers that can be placed in the corresponding residual space. The symbols *ll*, *max_thickl* and *H* are the length, the upper bound of the thickness and the height of the layers. The symbols *P*, *sst* and *sfrt* are the pallet set, strip selecting type and strip filling rate threshold, respectively. The symbol *layer_type* indicates that x-layers or y-layers will be created. First, a strip sequence *S_D* is created by circularly invoking CreateCandidateStrips (Procedure 4) and SelectStrip$^{sst}_{\geq sfrt}$() (Formula (14)). With *ll* and *S_D* as the parameters, a layer set *L_SET* is created by invoking LAYER$_x$() (Formula (17)) or LAYER$_y$() (Formula (18)). By arranging the layers in *L_SET*, a layer sequence *L_SEQ* is obtained. *L_SEQ* is the solution for CreateCandidateLayers (Procedure 3).

Procedure 4 describes how to create a sequence of candidate strips from a pallet set *P*. The symbols *lr*, *wr*, and *H* are the length, width and height of the residual space where the candidate strips will be placed. *P* is the pallet set. With the dimensions of the three different surfaces of each pallet in *P* as the lengths and widths, we create three strips by invoking KS^{strip}() (Formula (13)). All of these strips form a strip sequence *S*. *S* is the solution for CreateCandidateStrips (Procedure 4).

3.2. GIB – the greedy sub-algorithm of HCLPIB

After a container is filled with pallets, narrow spaces often exist at the tops and sides of the pallets in the container. To improve the filling rate, some of the remaining pallets are divided into product boxes and are placed in these narrow spaces. After the container is shipped to its destination, these product boxes will be palletized onto their original pallets.

To fill the narrow spaces with product boxes, we divide all of the narrow spaces in the container into multiple cuboid spaces. As shown in Fig. 6, there is only one cuboid space on the top of each good strip and one cuboid space between the right side (or back side) of each good layer and the right side (or back side) of the container. Additionally, the back right corner contains one cuboid space.

Procedure 5 describes how GIB fills the gaps with product boxes in a loaded container. *P* is a set of dividable pallets. *C_SPACE* is a set of cuboid residual spaces (to denote the narrow spaces in the container). GIB tries to divide each pallet and insert the obtained product boxes into the residual spaces. The algorithm for packing identical small items into a large empty box (characterized as the Identical Item Packing Problem, IIPP, by Wäscher et al., 2007 and implemented by George, 1992) is employed to insert the

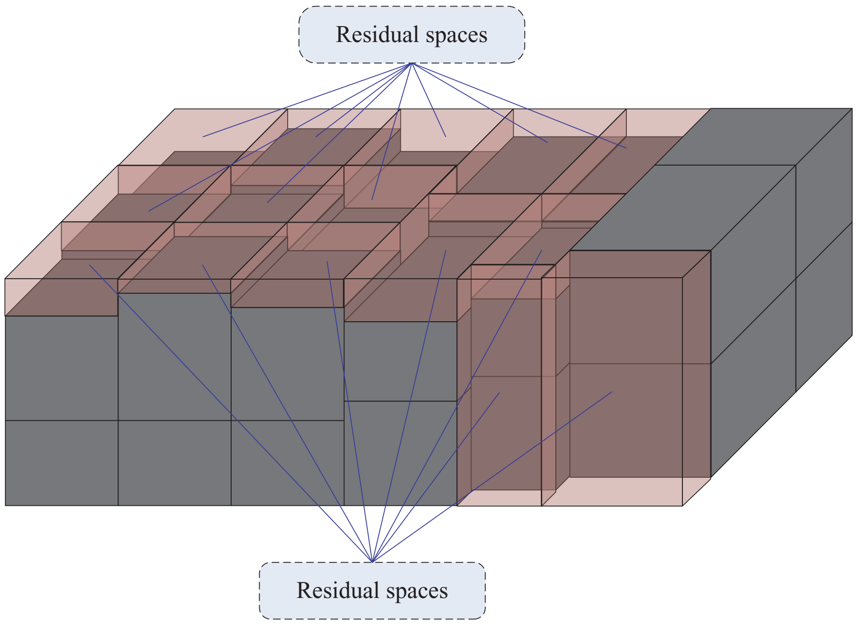

Fig. 6. Residual spaces in a loaded container.

product boxes into an individual residual space. Typically, more than one pallet can be divided and completely inserted into the residual spaces. Only the pallet with the highest volume will be divided to fill the residual spaces. We repeat the process until there are no pallets that can be divided and placed in the residual spaces.

4. Computational experiments and results

HCLPIB was implemented in C# and run on an Intel Q9400 @2.66 gigahertz with Microsoft Windows XP Professional. The compiling environment was Microsoft Visual Studio 2005.

First, we test QTS using the cases in BR1–BR15. Then, we test HCLPIB using the cases that are generated from the cases in BR1–BR15.

4.1. Computational experiments and results of QTS

Many algorithms for solving 3D-CLP have been published in recent years. H_BR (Bischoff & Ratcliff, 1995), GA_GB (Gehring & Bortfeldt, 1997), PT_SA (Bortfeldt et al., 2003), CLTRS (Fanslau & Bortfeldt, 2010), ID-GLTS (Zhu & Lim, 2012), HBMLS (Zhang et al., 2012) and HBTS (Liu et al., 2014) meet the orientation constraint and the full support constraint, whereas GRASP (Moura & Oliveira, 2005), MSA (Parreno et al., 2007), A2 (Huang & He, 2009), VNS (Parreno, Alvarez-Valdes, Oliveira, & Tamarit, 2010) and FDA (He & Huang, 2011) obey the orientation constraint only. To our knowledge, CBGAT by Gehring and Bortfeldt (1997) and HBTS by Liu et al. (2014) may satisfy the support constraint and the orientation constraint in this paper. We used the same data cases as the ones utilized by CBGAT and HBTS. To compare QTS with CBGAT and HBTS, we obeyed the orientation constraints defined for the boxes in BR1–BR15. Thus, there are at most six admissible orientations for one box.

Table 1 reports the computational results of CBGAT, HBTS and QTS for BR1–BR15 (CBGAT was only tested with BR1–BR7). QTS outperforms the other two algorithms for all of the test data. The computation time of QTS increases when the number of box types increases. The computation time of QTS for each case is acceptable, with the longest one requiring less than 30 minutes.

Table 1
The computational results of CBGAT, HBTS and QTS for BR1–BR15.

Case	Number of	Filling rate (percent)			Time (seconds)
	box types	CBGAT	HBTS	QTS	QTS
BR1	3	85.80	90.57	**90.99**	72
BR2	5	87.26	91.46	**91.92**	24
BR3	8	88.10	92.39	**92.84**	63
BR4	10	88.04	92.33	**92.79**	89
BR5	12	87.86	92.42	**92.85**	121
BR6	15	87.85	92.35	**92.86**	153
BR7	20	87.68	92.11	**92.69**	224
BR8	30	–	91.93	**92.46**	361
BR9	40	–	91.61	**92.13**	459
BR10	50	–	91.39	**91.98**	642
BR11	60	–	91.13	**91.74**	674
BR12	70	–	90.96	**91.39**	940
BR13	80	–	90.59	**91.73**	966
BR14	90	–	90.25	**90.39**	1127
BR15	100	–	89.79	**90.13**	1561
Mean		87.51	91.42	**91.90**	498

We briefly give the results of several famous algorithms. The mean filling rates for BR1–BR15 by CLTRS, ID-GLTS and HBMLS are 91.89 percent, 92.40 percent and 92.81 percent, respectively, when the orientation constraint and the full support constraint are met. QTS with an average filling rate of 91.90 percent for BR1–BR15 is slightly weaker than ID-GLTS and HBMLS on the filling rate. This is partly because of the need to satisfy the extra constraint (as shown in Fig. 1, the bottom of each pallet must be fully supported by the container floor or by the top of a single pallet).

4.2. Computational experiments and results of HCLPIB

Due to the lack of universally acknowledged test data for *HCLPIB*, we generated 1500 cases from the 1500 cases of BR1–BR15 by considering that some box types can be divided into small rectangular boxes if their serial numbers are even numbers. The length, width and height of each small box are 1/2, 1/2, and 1/3 of the length, width and height of the corresponding box type, respectively. Accordingly, we call the obtained case groups BR1p, BR2p,..., BR15p. The undivided boxes in BR1p–Br15p obey

Table 2
The computational results of HCLPIB for BR1p–BR15p.

Case	Number of	Number of	SAME_PRIORITY		DIFFERENT_PRIORITY	
	box types	undividable box types	Filling rate (percent)	Time (seconds)	Filling rate (percent)	Time (seconds)
BR1p	3	1	92.24	73	91.41	75
BR2p	5	2	93.09	25	92.31	26
BR3p	8	4	94.20	66	93.29	67
BR4p	10	5	93.95	94	93.04	101
BR5p	12	6	93.79	129	92.90	130
BR6p	15	7	93.95	169	93.09	173
BR7p	20	10	93.76	241	92.93	244
BR8p	30	15	93.53	391	92.69	392
BR9p	40	20	93.01	479	92.12	485
BR10p	50	25	93.27	667	92.36	669
BR11p	60	30	92.87	691	92.02	695
BR12p	70	35	92.20	972	91.32	977
BR13p	80	40	91.77	989	91.06	994
BR14p	90	45	91.53	1203	90.95	1211
BR15p	100	50	91.32	1619	90.71	1630
Mean			92.97	520	92.15	525

the same orientation constraints as the ones in BR1–BR15. However, if one box is divided, the obtained small rectangular boxes can be placed on six admissible orientations.

Table 2 reports the computational results of HCLPIB for BR1p–BR15p. The mean filling rates for BR1p–BR15p by HCLPIB are 92.97 percent when the undividable boxes have the same priority as the dividable boxes. The mean filling rates for BR1p–BR15p by HCLPIB are 92.15 percent when the undividable boxes are loaded prior to the dividable boxes. It is obvious that the same loading priority for both undividable boxes and dividable boxes may provide better results than the different loading priorities.

5. Conclusions

The container loading of pallets with infill boxes occurs at many typical furniture factories. There are many pallets of different sizes in these factories. We would like to load as many of these pallets as possible into a container. Usually, a certain number of pallets cannot be completely loaded into a container. Some pallets should be divided into product boxes to fill the narrow spaces at the sides and tops of the pallets placed in the container. We should guarantee that all of these infill boxes can be entirely palletized into complete pallets. We utilize a heuristic algorithm that consists of a tree search sub-algorithm and a greedy sub-algorithm to solve this problem. The tree search sub-algorithm is employed to arrange the pallets in the container. Then, the greedy sub-algorithm is applied to fill the narrow spaces with product boxes. The computational results on BR1–BR15 show that our algorithm is competitive under the proposed conditions. We also conclude that the same loading priority for both undividable boxes and dividable boxes may provide better results than the different loading priorities for them.

Acknowledgments

The authors would like to thank the editors and the anonymous reviewers. This work was supported by the National Nature Science Foundation of China (Grant nos. 61104054, 71232006, 61233001 and 61203166).

References

Araya, I., & Riff, M. C. (2014). A beam search approach to the container loading problem. *Computers & Operations Research, 43*(4), 100–107.

Bischoff, E. E., & Ratcliff, M. S. W. (1995). Issues in the development of approaches to container loading. *Omega, 23*(4), 377–390.

Bischoff, E. E., Janetz, F., & Ratcliff, M. S. W. (1995). Loading pallets with non-identical items. *European Journal of Operational Research, 84*(3), 681–692.

Bischoff, E. E., & Marriott, M. (1990). Comparative evaluation of heuristics for container loading. *European Journal of Operational Research, 44*(2), 267–276.

Bortfeldt, A., & Gehring, H. (1998). A tabu search algorithm for weakly heterogeneous container loading problems. *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.

Bortfeldt, A., & Mack, D. (2007). A heuristic for the three dimensional strip packing problem. *European Journal of Operational Research, 183*(3), 1267–1279.

Bortfeldt, A., & Wäscher, G. (2013). Constraints in container loading – A state-of-the-art review. *European Journal of Operational Research, 229*(1), 1–20.

Dyckhoff, H., & Finke, U. (1992). *Cutting and packing in production and distribution*. Heidelberg: Physica-Verlag.

Egeblad, J., Garavelli, C., Lisi, S., & Pisinger, D. (2010). Heuristics for container loading of furniture. *European Journal of Operational Research, 200*(3), 881–892.

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.

Faroe, O., Pisinger, D., & Zachariasen, M. (2003). Guided local search for three-dimensional bin-packing problem. *INFORMS Journal on Computing, 15*(3), 267–283.

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. (1992). A method for solving container packing for a single size of box. *Journal of the Operational Research Society, 43*(4), 307–312.

George, J. A., & Robinson, D. F. (1980). A heuristic for packing boxes into a container. *Computers and Operations Research, 7*(3), 147–156.

He, K., & Huang, W. (2011). An efficient placement heuristic for three-dimensional rectangular packing. *Computers & Operations Research, 38*(1), 227–233.

Hemminki, J. (1994). Container loading with variable strategies in each layer. In *Presented at ESI-X, July 2–15*. Jouy-En-Josas, France: EURO Summer Institute.

Hifi, M. (2002). Approximate algorithms for the container loading problem. *International Transactions in Operational Research, 9*(6), 747–774.

Huang, W., & He, K. (2009). A caving degree approach for the single container loading problem. *European Journal of Operational Research, 196*(1), 93–101.

Junqueira, L., Morabito, R., & Sato Yamashita, D. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research, 39*(1), 74–85.

Lim, A., Ma, H., Qiu, C., & Zhu, W. (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.

Liu, S., Tan, W., Xu, Z., & Liu, X. (2014). A tree search algorithm for the container loading problem. *Computers & Industrial Engineering, 11*(5), 20–30.

Lu, Y., & Cha, J. (2014). A fast algorithm for identifying minimum size instances of the equivalence classes of the pallet loading problem. *European Journal of Operational Research, 237*(3), 794–801.

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.

Morabito, R., & Arenales, M. (1996). Staged and constrained two-dimensional guillotine cutting problems: An AND/OR-graph approach. *European Journal of Operational Research, 94*(3,8), 548–560.

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.

Parreno, F., Alvarez-Valdes, R., Oliveira, J. F., & Tamarit, J. M. (2007). A maximal space algorithm for the container loading problem. *INFORMS Journal on Computing, 20*(3), 412–422.

Parreno, F., Alvarez-Valdes, R., Oliveira, J. F., & Tamarit, J. M. (2010). Neighborhood structures for the container loading problem: AVNS implementation. *Journal of Heuristics, 16*(1), 1–22.

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research, 141*(2), 143–153.

Ren, J., Tian, Y., & Sawaragi, T. (2011). A tree search method for the container loading problem with shipment priority. *European Journal of Operational Research, 214*(3), 526–535.

Sixt, M. (1996). *Dreidimensionale Packprobleme. Losungsverfahren basierend auf den Meta-Heuristiken Simulated Annealing und Tabu-Suche*. Frankfurt am Main: Europaischer Verlag der Wissenschaften.

Terno, J., Scheithauer, G., Sommerweiß, U., & Rieme, J. (2000). An efficient approach for the multi-pallet loading problem. *European Journal of Operational Research, 123*(2), 372–381.

Tian, T., Zhu, W., Lim, A., & Wei, L. (2016). The multiple container loading problem with preference. *European Journal of Operational Research, 248*(1), 84–94.

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. (2012). A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers & Operations Research, 39*, 2267–2276.

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., Huang, W., & Lim, A. (2012). A prototype column generation strategy for the multiple container loading problem. *European Journal of Operational Research, 223*(1), 27–39.