Computers & Industrial Engineering 75 (2014) 20–30

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A tree search algorithm for the container loading problem ☆

Liu Sheng [a,*], Tan Wei [b], Xu Zhiyuan [c], Liu Xiwei [a]

[a] *State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, 100190 Beijing, China*
[b] *IBM T. J. Watson Research Center, NY, United States*
[c] *Transport Planning and Research Institute, Ministry, Beijing, China*

ARTICLE INFO

Article history:
Received 2 December 2013
Received in revised form 5 May 2014
Accepted 31 May 2014
Available online 18 June 2014

Keywords:
Packing
Container loading
Heuristic algorithm
Tree search

ABSTRACT

This paper presents a binary tree search algorithm for the three dimensional container loading problem (3D-CLP). The 3D-CLP is about how to load a subset of a given set of rectangular boxes into a rectangular container, such that the packing volume is maximized. In this algorithm, all the boxes are grouped into strips and layers while three constraints, i.e., full support constraint, orientation constraint and guillotine cutting constraint are satisfied. A binary tree is created where each tree node denotes a container loading plan. For a non-root each node, the layer set of its left (or right) child is obtained by inserting a directed layer into its layer set. A directed layer is parallel (or perpendicular) to the left side of the container. Each leaf node denotes a complete container loading plan. The solution is the layer set whose total volume of the boxes is the greatest among all tree nodes. The proposed algorithm achieves good results for the well-known 3D-CLP instances suggested by Bischoff and Ratcliff with reasonable computing time.

1. Introduction

Cutting and packing problems (Dyckhoff & Finke, 1992) are classic problems focusing on the optimal use of resources. Cutting problems concern the best utilization of materials such as wood, steel and cloth, while packing problems concern the best capacity use of a given 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 and to relieving the traffic congestion.

This paper focuses on one of the packing problems, i.e., the three-dimensional container loading problem (3D-CLP). According to a recent typology proposed by Wäscher, Haußner, and Schumann (2007), this problem is in the broader category of the single knapsack problems (SKP) which is also called orthogonal packing problems (OPP). The problem is defined as follows: A container which is a large cuboid and a set of boxes which are small cuboids are given; usually the total volume of the boxes exceeds the container's volume. We assume that the profit of a box is proportional to its volume. A feasible arrangement of a sub-set of the given boxes is to be identified in such a way that: (1) the packed volume of the container is maximized, and (2) where applicable, additional constraints are met. A box arrangement is considered to be feasible if:

— Each box is placed completely in the container, with no two boxes overlapping in space; and
— The sides of each box are parallel to the container's boundary surfaces.

A box type is defined by the three side dimensions of a box and its placement constraints. A box type may contain one or more boxes of the same dimensions and placement constraints. A box set is characterized as *homogeneous* if it only contains one box type. A box set is considered as a *weakly heterogeneous* one if it only contains a few box types, and each box type includes a relatively large number of boxes. On the other hand, a box set is regarded as *strongly heterogeneous* if it includes many box types, and each box type only contains a few boxes.

In many practical applications, 3D-CLP is often subject to a large variety of constraints. A comprehensive and detailed survey about the constraints in container loading problems can be found in Zhang, Peng, and Leung (2012) and Zhang, Peng, and Zhang (2012). Three well-known constraints that are discussed e.g., by Bortfeldt and Wäscher (2013), Fanslau and Bortfeldt (2010), Zhang, Peng, and Leung (2012) and Zhang, Peng, and Zhang (2012):

(C1) *Orientation constraint*. There are up to 6 box orientations possible, but only 3 vertical box orientations. For certain box

☆ This manuscript was processed by Area Editor Qiuhong Zhao.
* Corresponding author. Tel.: +86 15910634676.
E-mail address: liusheng7801@163.com (S. Liu).

http://dx.doi.org/10.1016/j.cie.2014.05.024
0360-8352/© 2014 Elsevier Ltd. All rights reserved.

types, up to 2 of the maximal 3 possible vertical orientations are prohibited.
(C2) *Support constraint*. The bottom of each box which is not placed on the floor of the container must be supported completely (i.e., 100%) by other boxes underneath.
(C3) *Guillotine cutting constraint*. All boxes in a packing plan can be reproduced by a series of guillotine cuts. The cutting area of a guillotine cut lies parallel to a boundary surface of the container, and the cut piece is always completely separated in two smaller parts.

The constraints (C1) and (C2) appear frequently in practical packing situations (Bischoff & Ratcliff, 1995). Both the constraints (C1) and (C3) are relevant in 3D cutting because automated cutting machines sometimes can only perform guillotine cuts, whereas an orientation constraint is presented frequently if the items to be cut are decorated or corrugated. Sometimes when a forklift is employed to load (unload) some cargoes into (from) a container, the constraints (C3) should be fulfilled so as to improve its efficiency.

This paper presents a binary tree search method for 3D-CLP where the constraints (C1), (C2) and (C3) are all fulfilled. The rest of the paper is organized as follows. In Section 2, we provide a literature review of 3D-CLP. In Section 3, we present the basic concepts for 3D-CLP and for the proposed method. In Section 4, we analyze the results of the proposed method on BR1–BR15, and compare the proposed method with other methods. Finally in Section 5, we summarize the paper and present some perspectives for future research.

2. Literature review

The three-dimensional container loading problem is a typical NP-hard problem (Bischoff & Marriott, 1990), and therefore it cannot be solved by polynomial-time optimal algorithms. Exact algorithms for container loading (Hadjiconstantinou & Christofides, 1995; Martello & Vigo, 1998; Fekete, Schepers, & van der Veen, 2007) are efficient for small and medium instances. However they are usually confronted with the situation called combinatorial space explosion when the number of box types increases. As a result, heuristic methods prove to be a more realistic alternative of dealing with the three-dimensional container loading problem. Heuristic methods may get a suboptimal solution but they can produce good enough solutions in a reasonable timeframe. Many researchers provided various heuristic methods for solving the 3D-CLP.

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

(1) *Tree search methods*. Tree search or graph search methods were successfully utilized in the 3D-CLP. Several tree search methods were provided e.g., by Terno, Scheithauer, Sommerweiß, and Rieme (2000), Hifi (2002), Eley (2002) and Pisinger (2002). An And/Or graph search method is suggested by Morabito and Arenales (1994). A caving degree based flake arrangement approach for the container loading problem is presented by He and Huang (2010).
(2) *Non tree search methods*. Non tree search methods include classic heuristic methods and intelligent heuristic methods. The former for solving the 3D-CLP were presented by Bischoff, Janetz, and Ratcliff (1995), Bischoff and Ratcliff (1995), and Lim, Rodrigues, and Wang (2003), while the latter are the most used method types for the 3D-CLP in recent years. Genetic algorithms (GAs) were utilized by Hemminki (1994), Gehring and Bortfeldt (1997), Gehring and Bortfeldt (2002), and Bortfeldt and Gehring (2001). Simulated annealing methods (SAs) were provided by Sixt (1996) and Mack, Bortfeldt, and Gehring (2004). Tabu search algorithms (TSs) were suggested by Sixt (1996), Bortfeldt and Gehring (1998), Bortfeldt, Gehring, and Mack (2003). Local search methods were suggested by Faroe, Pisinger, and Zachariasen (2003) and Mack et al. (2004). Moura and Oliveira (2005) as well as Parreno, Alvarez-Valdes, Oliveira, and Tamarit (2007) introduce a greedy randomized adaptive search procedure (GRASP). Egeblad and Pisinger (2009) suggest a new heuristic algorithm based on sequence triple.

According to the packing approaches, Pisinger (2002) grouped these methods into five classes which are named wall building approach (suggested e.g., by Bortfeldt & Gehring, 2001; George & Robinson, 1980; Pisinger, 2002), block building approach (representatives of the approach are the TS method from Bortfeldt et al. (2003), Eley (2002), Fanslau and Bortfeldt (2010), Zhang, Peng, and Leung (2012), Zhang, Peng, and Zhang (2012, and the SA/TS hybrid method from Mack et al. (2004), horizontal layer building approach (realized e.g., by Bischoff et al., 1995; Terno et al., 2000), stack building approach (presented e.g., by Bischoff & Ratcliff, 1995; Gehring & Bortfeldt, 1997) and guillotine cutting approach(mixed with graph search method by Morabito & Arenales, 1994). Otherwise, heuristic algorithms based on the idea of caving degree were proposed by Huang and He (2009), He and Huang (2010), 2011). As far as we know, a tree search algorithm based on block building approach by Zhang, Peng, and Leung (2012) achieved the best solutions on the classic data set from Bischoff and Ratcliff (1995) and Davies and Bischoff (1998).

The great majority of the methods mentioned above obey the orientation constraint (C1) and the support constraint (C2) as well. Some methods also include further constraints from the packing context in the problem, e.g., a weight constraint for the freight (seen in Terno et al., 2000 and Bortfeldt & Gehring, 2001). A heuristic algorithm for container loading of furniture by Egeblad, Garavelli, Lisi, and Pisinger (2010) is remarkable where a large variety of irregular items are considered and many practical constraints are satisfied.

3. The binary tree search algorithm

In this paper we refer to our binary tree search method as HBTS (Heuristic Binary Tree Search Algorithm). HBTS and the algorithm by Pisinger (2002) are both tree search algorithms based on wall building. The diagrams of the container loading plans obtained by HBTS and the algorithm by Pisinger (2002) are shown in Fig. 1. The number on each layer indicates its loading order in the corresponding plan. It is obvious that HBTS differs from the algorithm by Pisinger (2002), in terms of that, each layer can be separated from the layer set according to its loading order by a guillotine cutting.

3.1. Basic concepts

First some basic concepts are presented and terminological agreements are made.

As shown in Fig. 2, a container is placed in the first octant of a 3D coordinates system (3D-CS). The terms such as "left", "top" are illustrated. Let

$$C = (len, wid, hei) \tag{1}$$

denote a container. Symbols *len*, *wid* and *hei* denote the container length, width and height, respectively.

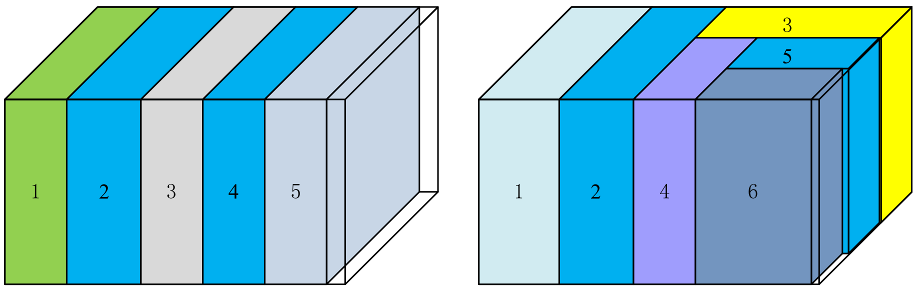

Fig. 1. The diagrams of two container loading plans.

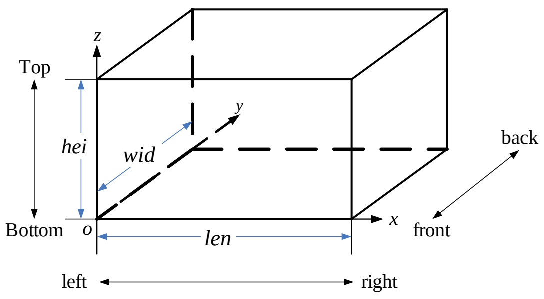

Fig. 2. Container in 3D coordinates system.

Let

$$B = \{b_1, b_2, \ldots, b_m\} \tag{2.1}$$

be the box set which contains m box types. b_i is the ith box type in the set that is defined as

$$b_i = (l_i, w_i, h_i, d_i, \alpha_i, \beta_i, \gamma_i) \tag{2.2}$$

where the symbols l_i, w_i and h_i are the length, width and height of b_i, respectively. The symbol d_i is the number of the boxes of b_i. Fig. 3 displays the dimensions and the lying orientation of b_i. The values of the symbols α_i, β_i and γ_i and their corresponding implications are listed:

$$\alpha_i = \begin{cases} 1 & \text{(if the orientations shown in Fig. 3(b) are allowed)} \\ 0 & \text{(if the orientations shown in Fig. 3(b) are prohibited)} \end{cases},$$

$$\beta_i = \begin{cases} 1 & \text{(if the orientations shown in Fig. 3(c) are allowed)} \\ 0 & \text{(if the orientations shown in Fig. 3(c) are prohibited)} \end{cases} \text{and},$$

$$\gamma_i = \begin{cases} 1 & \text{(if the orientations shown in Fig. 3(d) are allowed)} \\ 0 & \text{(if the orientations shown in Fig. 3(d) are prohibited)} \end{cases}.$$

Therefore the solution of container loading can be obtained by solving the following integer programming (IP) problem:

$$\max\left\{ v_P = \sum_{i=1}^{m} l_i w_i h_i q_i \mid Q \text{ is a feasible solution}; q_i \text{ is an integer}, 0 \leqslant q_i \leqslant d_i, i = 1,2,\ldots,m \right\},$$

where q_i is the frequency of type-i boxes ($i = 1,2, \ldots, m$) in the container and is therefore the decision variable. $Q = \{q_1, q_2, \ldots, q_m\}$ is the set of q_i ($i = 1,2, \ldots, m$).

A cuboid (box, envelope cuboid of a strip and layer, empty space in the container) is referred to as *oriented* if it may no longer be turned around the 3D-CS. An oriented cuboid lies "somewhere" in the first octant of the 3D-CS and parallel to its axes. We assume that the corner nearest to the origin of the 3D-CS is the reference corner of an oriented cuboid. The *mx*, *my*, and *mz* denote the three dimensions of an oriented cuboid in the coordinate directions, respectively.

A residual space is an oriented cuboid space in the container given by its three side dimensions and its reference corner. The *mz* of each residual space in the presented method always equals the *mz* of the container. An example for a residual space with two placed layers is shown in Fig. 4.

A residual space is denoted as

$$R = (lr, wr, hei) \tag{3}$$

where *lr*, *wr* and *hei* are its *mx*, *my* and *mz*, respectively. If a residual space is not completely rectangular, we consider the *mx*, *my* and *mz* of the maximum cuboid inside it as its *mx*, *my* and *mz*, respectively.

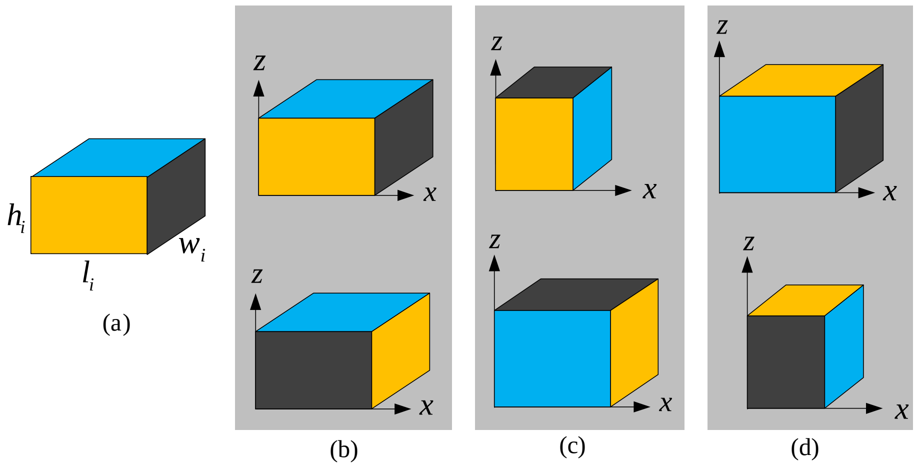

Fig. 3. An box and its lying orientations in 3D coordinates system.

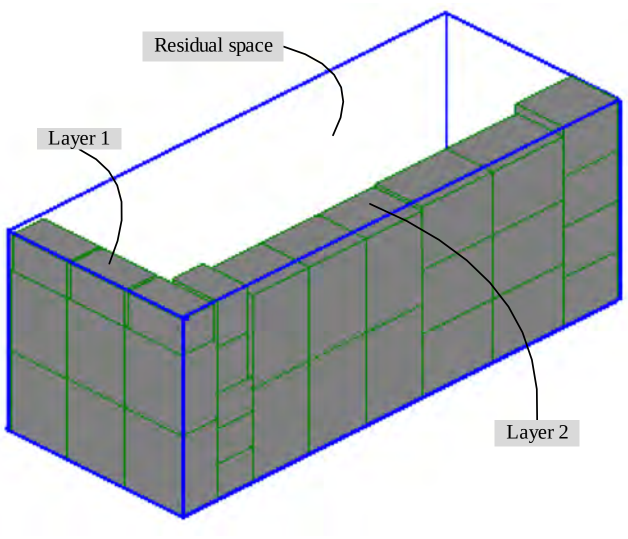

Fig. 4. Residual space with two placed layers.

As shown in Fig. 5, the boxes in each strip (called *s*) are arranged along a straight line which is parallel to the *z* axis in the 3D-CS. The surfaces of each box must be parallel to one of the three planes: *xy*, *xz* and *yz* in Fig. 5. The boxes should also be arranged such that *s* is no longer in height than the container and the volume of the envelope cuboid (see Fig. 5(a)) of *s* is minimized. Let *ls* and *ws* be the length and width (Generally the length is not shorter than the width) of the envelope cuboid. The filling rate of *s* is defined as:

$$frs = \sum_{i=1}^{m} l_i w_i h_i p_i / (ls^* ws^* hei) \tag{4}$$

where p_i is the frequency of b_i in *s*.

Let

$$S = \{s_1, s_2, \cdots, s_n\} \tag{5}$$

be a strip set which contains *n* strips.

As shown in Fig 5(b), the strips in each layer (called *l*) are arranged along a straight line parallel to either *x* or *y* axis in the 3D-CS. The surfaces of the envelope cuboid of each strip should be parallel to *xy* plane, *xz* plane or *yz* plane. The strips in *l* should also be arranged such that the volume of the envelope cuboid (see Fig. 5(b)) is minimized. The envelope cuboid must be no longer in length than *mx* (or *my*) of the corresponding residual space if *l* is placed parallel to *x* (or *y*) axis. Let *ll* and *thickl* be the length and width of the envelope cuboid of the layer, respectively. *ll* is appointed to equal *mx* (or *my*) if *l* is placed parallel to *x* (or *y*) axis. The filling rate of *l* is defined as:

$$frl_l = \begin{cases} \sum_{j=1}^{n}\sum_{i=1}^{m} l_i w_i h_i p_{ij} / (mx^* thickl^* hei) & (\text{if } l \text{ lies parallel to } x \text{ axis}) \\ \sum_{j=1}^{n}\sum_{i=1}^{m} l_i w_i h_i p_{ij} / (my^* thickl^* hei) & (\text{if } l \text{ lies parallel to } y \text{ axis}) \end{cases} \tag{6}$$

where p_{ij} is the frequency of b_i in s_j (i = 1,2, ... ,m; j = 1,2, ... ,n).

As illustrated in Fig. 5, all the strips in a layer can be separated from the layer by a set of parallel guillotine cuttings. Similarly, all the boxes in a strip also can be separated from the strip by a set of parallel guillotine cuttings.

The computational time of HBTS mainly consists of the time to solve knapsack problems. Therefore we try to use the branch-and-bound algorithm to reduce the number of times the knapsack problems are solved. We should record the parameters with that each strip is created. Thus we can create a strip by copying a similar record instead of by solving a knapsack problem. Let

$$KnapRecStrip = (h\_c, l\_c, w\_c, INPUT, OUTPUT) \tag{7}$$

be the strip creating record where *h_c*, *l_c*, *w_c*, *INPUT*, and *OUTPUT* denote the height, the length, the width of the envelope cuboid of the strip, the candidate box set, and the subset that is chosen to form the strip, respectively. According to the Branch & Bound theory (Lawler & Wood, 1966; Narendra & Fukunaga, 1977), we can confirm that *OUTPUT* must be upper bound of the knapsack problem whose input (denoted as *INPUT1*) is a subset of *INPUT*. If *OUTPUT* is a subset of *INPUT1*, *OUTPUT* must be the output of solving the knapsack problem whose input is *INPUT1*.

After a strip is created by solving a knapsack problem, we apply Formula (7) to recording the inputs and outputs of the knapsack problem. Through this recording, when we try to create a strip, we can create it by referring to these records instead of solving a knapsack problem. As a result, the total time to compute a container loading plan will be reduced significantly.

3.2. The overall algorithm

The binary tree for searching a good three dimensional container loading plan is shown in Fig. 6. Each tree node is the top view of a container loading plan in the 3D-CS. Each leaf node must be a complete container loading plan. A container loading plan is

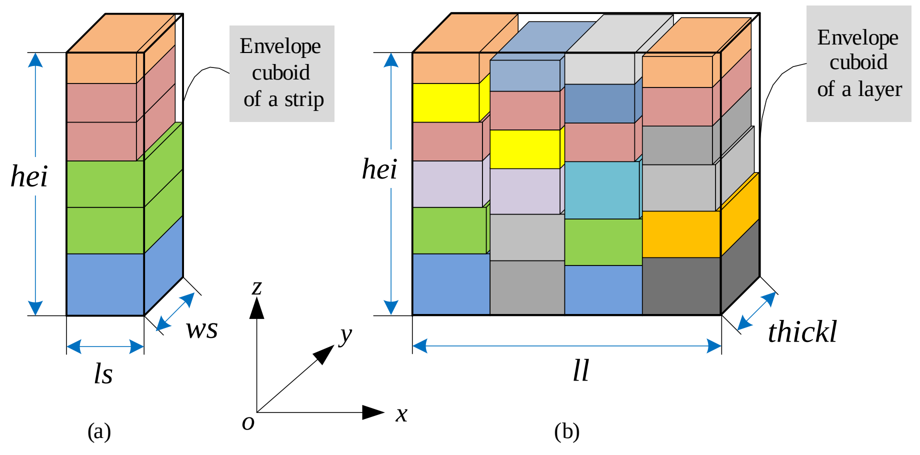

Fig. 5. Strip, layer and their envelope cuboids.

considered as *complete* if the residual space of the container cannot accommodate any remaining boxes.

The length and width of a strip are determined before the strip is created. The length *ls* and width *ws* are always equal to two dimensions of the length, width and height of one box in the box set *B*, respectively. We create strips one by one from *B* by a greedy strategy. To explain this strategy, firstly we introduce some definitions.

$$H_\alpha(b_i, ls, ws) = \begin{cases} h_i & \text{if}(\max\{l_i, w_i\} \leqslant ls,\ \min\{l_i, w_i\} \leqslant ws,\ \alpha_i = 1) \\ +\infty & \text{otherwise} \end{cases} \tag{8.1}$$

(8.1) means the height of the envelope cuboid of b_i in a strip with the length *ls* and width *ws* if the height of b_i are parallel to *z* axis.

$$H_\beta(b_i, ls, ws) = \begin{cases} l_i & \text{if}(\max\{w_i, h_i\} \leqslant ls,\ \min\{w_i, h_i\} \leqslant ws,\ \beta_i = 1) \\ +\infty & \text{otherwise} \end{cases} \tag{8.2}$$

(8.2) means the height of the envelope cuboid of b_i in the strip if the length of b_i are parallel to *z* axis.

$$H_\gamma(b_i, ls, ws) = \begin{cases} w_i & \text{if}(\max\{l_i, h_i\} \leqslant ls,\ \min\{l_i, h_i\} \leqslant ws,\ \gamma_i = 1) \\ +\infty & \text{otherwise} \end{cases} \tag{8.3}$$

(8.3) means the height of the envelope cuboid of b_i in the strip if the width of b_i are parallel to *z* axis. The value $+\infty$ in Formula (8.1), (8.2), (8.3) means the corresponding orientations are prohibited or the box exceeds the boundary of the envelope cuboid of the strip.

Therefore it is obvious that

$$H(b_i, ls, ws) = \min\{H_\alpha(b_i, ls, ws), H_\beta(b_i, ls, ws), H_\gamma(b_i, ls, ws)\} \tag{9}$$

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

Let

$$KS^{strip}(hei, ls, ws) = \max\left\{ \sum_{i=1}^{m} \frac{l_i w_i h_i q_i}{hei^* ls^* ws} \middle| \begin{array}{l} Q = \{q_1, q_2, \ldots, q_m\} \\ \quad \text{is a feasible solution,} \\ q_i \leqslant d_i,\ \sum_{i=1}^{m} q_i^* H(b_i, ls, ws) \leqslant hei, \\ \quad i = 1, 2, \ldots, m \end{array} \right\} \tag{10}$$

denote a one dimensional knapsack model to generate a strip with the length *ls* and width *ws* where q_i is the frequency of b_i in the strip.

When we group a box set into a strip set, we utilize two of the three sizes (length, width and height) of each box as the length and width to create a strip. Then we select a strip from the obtained strips and insert it into the strip set. If we select the strip with filling rate closest to a given value (usually in [0.9, 1]), we can usually get a better container loading plan than selecting the strip with the highest filling rate. Therefore we define a function

$$\underset{attp}{\text{closest}}\{M\} \tag{11}$$

that returns the number that is closest to *attp* in the numeric set *M*. Herein *attp* is called *attraction point*. Usually *attp* is in [0.9, 1].

The mathematical model

$$strip_\alpha(hei, lr, wr, attp) = \underset{attp}{\text{closest}}\left\{ KS^{strip}(hei, l_i, w_i) \middle| \begin{array}{l} \max\{l_i, w_i\} \leqslant \max\{lr, wr\}, \\ \min\{l_i, w_i\} \leqslant \min\{lr, wr\}, \\ \alpha_i = 1,\ \ i = 1, 2, \ldots, m \end{array} \right\} \tag{12.1}$$

is to get a strip whose length and width equal the length and width of one of the boxes in *B*.

Similarly, the model

$$strip_\beta(hei, lr, wr, attp) = \underset{attp}{\text{closest}}\left\{ KS^{strip}(hei, w_i, h_i) \middle| \begin{array}{l} \max\{w_i, h_i\} \leqslant \max\{lr, wr\}, \\ \min\{w_i, h_i\} \leqslant \min\{lr, wr\}, \\ \beta_i = 1,\ \ i = 1, 2, \ldots, m \end{array} \right\} \tag{12.2}$$

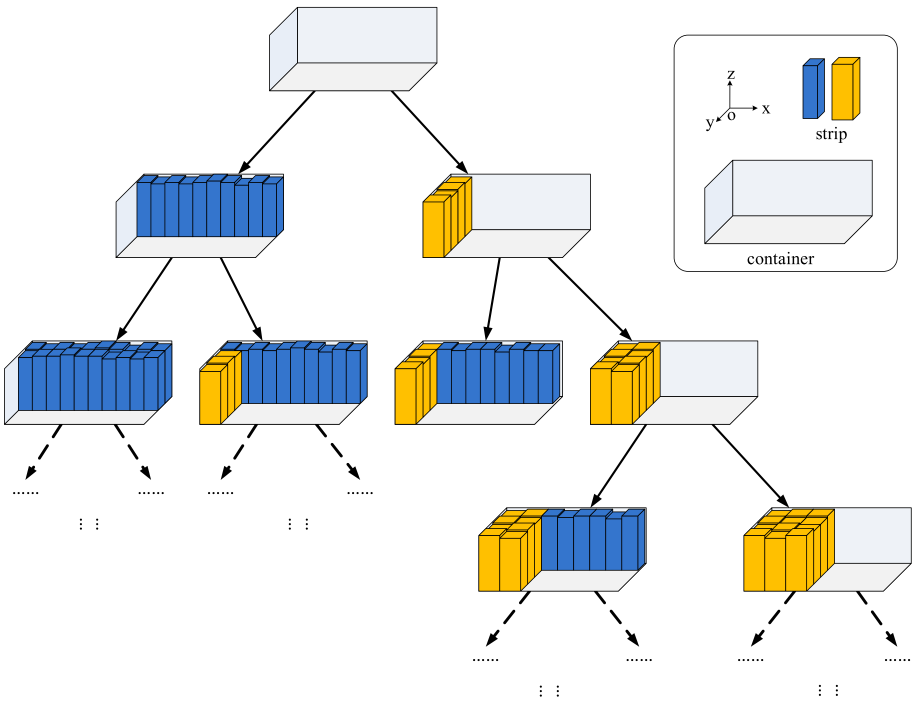

Fig. 6. The binary tree for searching a good container loading plan.

is to get a strip whose length and width equal the width and height of one of the boxes in B. And

$$strip_{\gamma}(hei, lr, wr, attp) = \underset{attp}{\text{closest}}\left\{ KS^{strip}(hei, l_i, h_i) \middle| \begin{array}{l} \max\{l_i, h_i\} \leqslant \max\{lr, wr\}, \\ \min\{l_i, h_i\} \leqslant \min\{lr, wr\}, \\ \gamma_i = 1, \quad i = 1, 2, \ldots, m \end{array} \right\} \tag{12.3}$$

is to get a strip whose length and width equal the length and height of one of the boxes in B.

In Formulas (12.1), (12.2), (12.3), *attp* is an *attraction point* in [0.9, 1]. *lr*, *wr* and *hei* are the length and width of the current residual space and the height of the container, respectively.

Then we define

$$strip(hei, lr, wr, attp) = \underset{attp}{\text{closest}}\left\{ \begin{array}{l} strip_{\alpha}(hei, lr, wr, attp), \\ strip_{\beta}(hei, lr, wr, attp), \\ strip_{\gamma}(hei, lr, wr, attp) \end{array} \right\} \tag{13}$$

as the mathematical model to generate a strip according to the given box set and residual space. We repeatedly solve the mathematical model in Formula (13) until all the boxes are converted into strips. Then we get a strip set $S = \{s_1, s_2, \ldots, s_n\}$.

If we create a layer from a strip set S, the thickness of a layer is always equal to the length or width of one strip in S. We also create a layer from S by a greedy strategy. Firstly some definitions are presented.

$$L(s_i, thickl) = \begin{cases} ls_i & \text{if}(ls_i > thickl \ \& \ ws_i \leqslant thickl) \\ ws_i & \text{if}(ls_i \leqslant thickl \ \& \ ws_i \leqslant thickl) \\ +\infty & \text{otherwise} \end{cases} \tag{14}$$

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

Let

$$KS^{layer}(hei, ll, thickl) = \max\left\{ \frac{\sum_{i=1}^{n} V_i}{hei^{*} ll^{*} thickl} \middle| \begin{array}{l} P = \{p_1, p_2, \ldots, p_n\} \text{ is a feasible solution}, \\ p_i \in \{0,1\}, \sum_{i=1}^{n} p_i^{*} L(s_i, thickl) \leqslant ll, \\ i = 1, 2, \ldots, n \end{array} \right\} \tag{15}$$

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

The mathematical model

$$layer_x(lr) = \max\left\{ \begin{array}{l} KS^{layer}(lr, ls_i) | i = 1, 2, \ldots, n \\ \cup \\ KS^{layer}(lr, ws_i) | i = 1, 2, \ldots, n \end{array} \right\} \tag{16}$$

is to get a layer whose length equals the length of the current residual space. *lr* is the length of the current residual space.

The mathematical model

$$layer_y(wr) = \max\left\{ \begin{array}{l} KS^{layer}(wr, ls_i) | i = 1, 2, \ldots, n \\ \cup \\ KS^{layer}(wr, ws_i) | i = 1, 2, \ldots, n \end{array} \right\} \tag{17}$$

is to get a layer whose length equals the width of the current residual space. *wr* is the width of the current residual space.

A container loading plan may be denoted as a layer set. In the set, some layers are parallel to the left side of the container while others are perpendicular to it.

In the presented binary tree, each tree node denotes a unique layer set. The root node denotes an empty layer set that includes no layers. For each non-root node, we can get the layer set of its left child by solving Formula (16). Similarly, we can get the layer set of its right child by solving Formula (17). Obviously, the solution is the layer set in which the total volume of the boxes is greater than the others among the tree.

The overall algorithm of HBTS is described in Algorithm 1. In Algorithm 1, C, B and *attp* represent the container, the box set and the attraction point (defined in Formula (11)), respectively. The layer set L^{left} denotes the best container loading plan in the left child tree while L^{right} denotes the best container loading plan in the right child tree. L^{left} is obtained by solving the algorithm CreateLeftChildTree (described in Algorithm 2) while L^{right} is obtained by solving the algorithm CreateRightChildTree (described in Algorithm 3). Algorithm 1 returns L^{left} if the total volume of the boxes in L^{left} is higher than the one in L^{right}. Otherwise, it returns L^{right}.

Algorithm 1

```
HeuristicBinary TreeSearch (C, B, attp)
  $L^{left}$: =CreateLeftChildTree (B, len, wid, hei, attp)
  $L^{right}$: =CreateRightChildTree (B, len, wid, hei, attp)
  if(the total volume of the boxes in $L^{left}$ $\geqslant$ the total volume of
     the boxes in $L^{right}$) return $L^{left}$
  else return $L^{right}$
```

As shown in Algorithm 2, CreateLeftChildTree is invoked recursively to create the left child tree of a tree node in the binary tree (shown in Fig. 7). If the residual space ($lr \otimes wr \otimes hei$) cannot accommodate any box left in the box set B, Algorithm 2 returns an empty layer set. Otherwise, B is grouped into a strip set S. Then a layer l is obtained by solving Formula (16). Notice that l is parallel to the left side of the container. And then the boxes that are included in l are removed from B. CreateLeftChildTree is invoked again to create the left child tree of the current tree node while CreateRightChildTree (described in

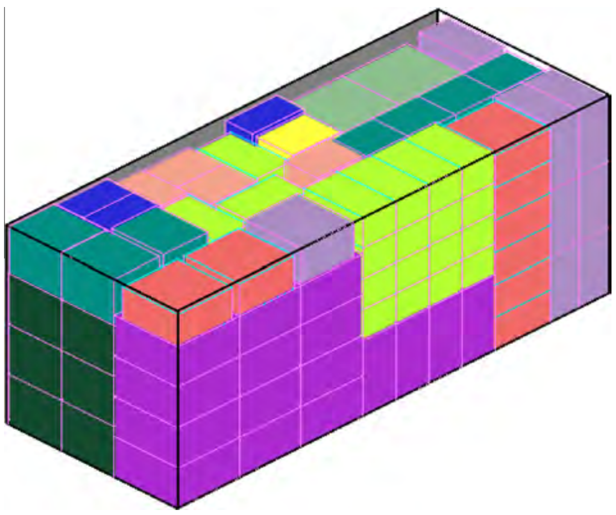
Fig. 7. Example solution for weakly heterogeneous instance.

Algorithm 3) is invoked to create the right child tree. In the end, Algorithm 2 returns the layer set where the total volume of the boxes is the highest in the left child tree of the current tree node.

Algorithm 2

```
CreateLeftChildTree (B, lr, wr, hei, attp)
  if(B contains boxes that can be accommodated by the
      residual space ($lr \otimes wr \otimes hei$))
    S: =CreateStripSet (B, lr, wr, hei, attp) //create a strip set
        from the box set B
    l: =solve Formula (16) with the strip set S and the layer
        length lr
    remove the boxes which are included in l from B
    $L^{left}$: =l + CreateLeftChildTree(B, lr, wr − l.thickl, hei, attp)
    $L^{right}$: =l + CreateRightChildTree(B,lr,wr − l.thickl,hei,attp)
      if(the total volume of the boxes in $L^{left} \geqslant$ the total
  volume of the boxes in $L^{right}$) return $L^{left}$
    else return $L^{right}$
  else return $\varnothing$
```

CreateRightChildTree (see Algorithm 3) is invoked recursively to create the right child tree of a tree node in the binary tree (shown in Fig. 7). If the residual space ($lr \otimes wr \otimes hei$) cannot accommodate any box in the box set *B*, Algorithm 3 returns an empty layer set. Otherwise, *B* is firstly grouped into a strip set *S*. Then a layer *l* is obtained by solving Formula (17). Notice that *l* is perpendicular to the left side of the container. And then the boxes that are included in *l* are removed from *B*. CreateRightChildTree is invoked again to create the right child tree of the current tree node while CreateLeftChildTree (described in Algorithm 2) is invoked to create the left child tree. At last Algorithm 3 returns the layer set where the total volume of the boxes is the highest in the right child tree of the current tree node.

Algorithm 3

```
CreateRightChildTree (B, lr, wr, hei, attp)
  if(B contains boxes that can be accommodated by the
      residual space ($lr \otimes wr \otimes hei$))
    S: =CreateStripSet (B, lr, wr, hei, attp) //create a strip set
        from the box set B
    l: =solve Formula (17) with the strip set S and the layer
        length wr
    remove the boxes which are included in l from B
    $L^{left}$: =l + CreateLeftChildTree(B, lr − l.thickl, wr, hei, attp)
    $L^{right}$: =l + CreateRightChildTree(B, lr − l.thickl, wr, hei, attp)
    if(the total volume of the boxes in $L^{left} \geqslant$ the total volume
        of the boxes in $L^{left}$) return $L^{right}$
    else return $L^{right}$
  else return $\varnothing$
```

Algorithm 4 describes how to create a strip set from a box set *B*. Firstly an empty strip set *S* is created. Then we judge whether *B* contains boxes that can be accommodated by the residual space ($lr \otimes wr \otimes hei$). If *B* contains such boxes, a strip *s* will be created from *B* by referring to a similar record or solving Formula (13). Then the boxes which are included in *s* will be removed from *B*. After that, *s* will be inserted into *S*. We repeat creating strips until *B* contains no boxes that can be accommodated by the residual space.

Algorithm 4

```
CreateStripSet (B, lr, wr, hei, attp)
  S: =$\varnothing$
  while(B contains boxes that can be accommodated by the
      residual space ($lr \otimes wr \otimes hei$))
    B_1: =GetSimilarKnapsackRecord (hei, lr, wr, B)
    if(B_1 = $\otimes$)
      s: =solve Formula (13) with B
      store hei, lr, wr, B and the set of boxes in s as a strip
      creating record in the memory
    else
      s: =create a strip with B_1
    remove the boxes which are included in s from B
    S: =S + s
  return S
```

Algorithm 5 seeks a similar strip creating record from the strip creation history. The similar record describes input and output of solving one knapsack problem which are defined as *INPUT* and *OUTPUT* in Formula (7) and (8), respectively. We will create a strip from the box set *IN*. If the height, length and width of a strip that was created before are equal to *hei*, *l* and *w*, respectively, and *IN* is a subset of the corresponding input *INPUT*, we get the upper bound—the corresponding output *OUTPUT*. If *OUTPUT* is a subset of *IN*, we get a feasible solution—*OUTPUT*. It is obvious *OUTPUT* is the best solution.

Algorithm 5

```
GetSimilarKnapsackRecord (hei, l, w, IN)
  for each record(formatted as KnapRecStrip) r in the memory
    if $\left( \begin{array}{l} r.h\_c = hei \text{ and } r.l\_c = l \text{ and } r.w\_c = w \text{ and} \\ r.INPUT \supseteq IN \text{ and } r.OUTPUT \subseteq IN \end{array} \right)$
    return r.OUTPUT
  return $\varnothing$
```

4. Computational experiments and results

The proposed HBTS algorithm is implemented in C#, and run on a server with Intel Core2 Duo Q8300@2.5 GHz and Microsoft Windows XP Professional. The compiling environment is Microsoft Visual Studio 2005. By computational experiments, we find that we can get a good balance between the solution quality and the computation time if we use $ATTP = \{attp_i = 0.9 + 0.003i, i = 1, 2, \ldots, 33\}$ as the *attraction point* set in the proposed algorithm. Algorithm 1 is invoked 33 times with $attp_i$ (i = 1, 2, ..., 33) as the corresponding parameter. For each instance, the solution is the best result among the 33 tested groups. Solving the knapsack problems in Algorithm 4 consumes most of the computational times. Thus a strip creating record set is generated to store the input and output data of the knapsack functions. The frequency of solving the knapsack problems is greatly reduced by creating strip according to the historic data. When the computational time of Algorithm 1 exceeds 600 s, Algorithm 1 will terminate and return the best found solution.

The test data that comes from Bischoff and Ratcliff (1995) and Davies and Bischoff (1998) include 16 cases from BR0 to BR15. Each case includes 100 instances. These instances can be downloaded from OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/

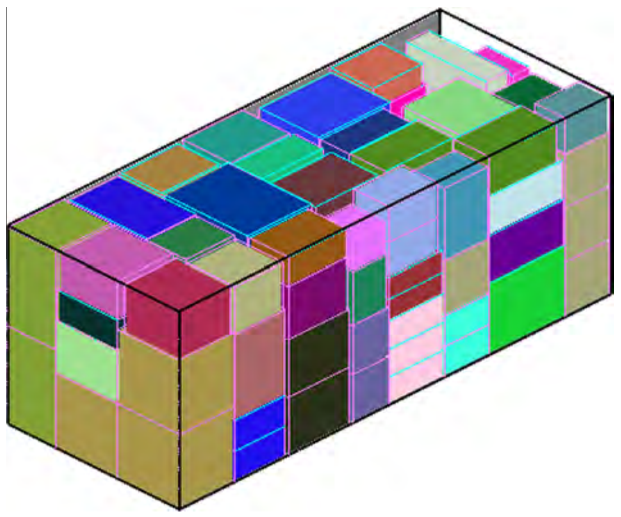

Fig. 8. Example solution for strongly heterogeneous instance.

Table 1
Platforms for testing HBTS and other algorithms.

Algorithm	Platform
H_BR (Bischoff & Ratcliff, 1995)	–
GA_GB (Gehring & Bortfeldt, 1997)	Pentium 130
PTSA (Bortfeldt et al., 2003)	Pentium 2 GHz
GRASP (Moura & Oliveira, 2005)	–
MSA (Parreno et al., 2007)	Pentium Mobile 1500 MHz, 512 MB Ram, C++
HSA (Zhang et al., 2009)	Core 2 Duo 2.0 GHz, C++
A2 (Huang & He, 2009)	1.7 GHz, Windows, Java
VNS (Parreno et al., 2010)	Pentium Mobile 1500 MHz, 512 MB Ram, C++
CLTRS (Fanslau & Bortfeldt, 2010)	Set A: 2.6 GHz; set B: 800 MHz
FDA (He & Huang, 2011)	Xeon 2.33 GHz, Java, J2SE V1.5.0_14
MLHS (Zhang, Peng, & Leung, 2012; Zhang, Peng, & Zhang, 2012)	Xeon X5460@3.16 GHz, Debian Linux, C++, gcc 4.3.2
HBMLS (Zhang, Peng, & Leung, 2012; Zhang, Peng, & Zhang, 2012)	Xeon X5460@3.16 GHz, Debian Linux, C++, gcc 4.3.2
HBTS	Core2 Q8300@2.5 GHz, Windows XP, C#, visual studio 2005

orlib/thpackinfo.html). The numbers of box types in the 16 cases are 1, 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90, and 100, respectively. The box sets in from BR0 to BR15 vary from homogeneous through weakly heterogeneous to strongly heterogeneous. Usually the instances from BR1 to BR7 are considered as weakly heterogeneous loading problems, while the instances from BR8 to BR15 are considered as strongly ones.

The instances from BR1 to BR7 were tested in H_BR (by Bischoff & Ratcliff, 1995), GA_GB (by Gehring & Bortfeldt, 1997), PTSA (by Bortfeldt et al., 2003), GRASP (by Moura & Oliveira, 2005), MFB (by Lim, Rodrigues, & Yang, 2005), RHA (by Juraitis, Stonys, Starinskas, Jankauskas, & Rubliauskas, 2006), H_B (by Bischoff, 2006) and SPBBL-CC4 (by Bortfeldt & Mack, 2007) which are devoted into the weakly heterogeneous loading problem.

MSA (by Parreno et al., 2007), HSA (by Zhang, Peng, Zhu, & Chen, 2009), VNS (by Parreno, Alvarez-Valdes, Oliveira, & Tamarit, 2010), CLTRS (by Fanslau & Bortfeldt, 2010), FDA (by He & Huang, 2011), MLHS (by Zhang, Peng, & Zhang, 2012), and HBMLS (by Zhang, Peng, & Leung, 2012) tested all the instances from BR1 to BR15 while A2 (by Huang & He, 2009) tested the ones from BR8 to BR15.

All the above algorithms fulfilled the orientation constraint (C1). Some of them fulfilled both the support constraint (C2) and the orientation constraint (C1). The computational results listed later come from aforementioned literatures. All the instances from BR1 to BR15 are tested by HBTS. Both the orientation constraint (C1) and the support constraint (C2) are fulfilled in HBTS. In particular, the guillotine cutting constraint (C3) is also fulfilled in HBTS (See Figs. 1, 4 and 5). The example solutions for heterogeneous instances are illustrated in Figs. 7 and 8, respectively.

4.1. Comparison with other algorithms

Table 1 reports the platforms for testing HBTS and other algorithms. The data regarding other algorithms are from literatures cited at the beginning of this section. Some algorithms also described the platforms they used in details, while others did not reveal this information.

Table 2 reports the computational results of HBTS and other algorithms for the instances from BR1 to BR7. All the data in Table 2 denote the average filling rate (%) for one case. HBTS is worse than HBMLS (by Zhang, Peng, & Leung, 2012) which achieved the best solutions when C1 and C2 are considered. Also, HBTS is worse than CLTRS (by Fanslau & Bortfeldt, 2010) and MLHS (by Zhang, Peng, & Zhang, 2012). The results indicate block-building approaches achieve higher volume utilization than wall-building approaches for weakly heterogeneous instances up to now.

Table 3 reports the computational results of HBTS and other algorithms for BR8-BR15. All the data in Table 3 denote the average filling rate (%) for one case. For BR8–BR11, HBTS is worse than

Table 2
Results of HBTS and other methods for BR1–BR7.

Algorithm	Constraint	Filling rate (%)						
		BR1	BR2	BR3	BR4	BR5	BR6	BR7
H_BR	C1&C2	85.40	86.25	85.86	85.08	85.21	83.84	82.95
GA_GB	C1&C2	85.80	87.26	88.10	88.04	87.86	87.85	87.68
PTSA	C1&C2	93.52	93.77	93.58	93.05	92.34	91.72	90.55
GRASP	C1	93.52	93.77	93.58	93.05	92.34	91.72	90.55
MSA	C1	93.85	94.22	94.25	94.09	93.87	93.52	92.94
HSA	C1&C2	93.81	93.94	93.86	93.57	93.22	92.72	91.99
A2	C1	–	–	–	–	–	–	–
VNS	C1	94.93	95.19	94.99	94.71	94.33	94.04	93.53
CLTRS	C1	95.05	95.39	95.45	95.18	94.96	94.80	94.26
	C1&C2	94.50	94.67	94.74	94.41	94.05	93.83	93.15
FDA	C1	92.92	93.93	93.71	93.68	93.73	93.63	93.14
MLHS	C1	94.92	95.48	95.69	95.53	95.44	95.38	94.95
	C1&C2	94.49	94.89	95.20	94.94	94.78	94.55	93.95
HBMLS	C1	94.92	95.48	95.69	95.53	95.44	95.38	95.00
	C1&C2	94.43	94.87	95.06	94.89	94.68	94.53	93.96
HBTS	C1,C2&C3	90.57	91.46	92.39	92.33	92.42	92.35	92.11

Table 3
Results of HBTS and other methods for BR8–BR15.

Algorithm	Constraint	Filling rate (%)							
		BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15
H_BR	C1&C2	–	–	–	–	–	–	–	–
GA_GB	C1&C2	–	–	–	–	–	–	–	–
PTSA	C1&C2	–	–	–	–	–	–	–	–
GRASP	C1	–	–	–	–	–	–	–	–
MSA	C1	91.02	90.46	89.87	89.36	89.03	88.56	88.46	88.36
HSA	C1&C2	90.56	89.7	89.06	88.18	87.73	86.97	86.16	85.44
A2	C1	88.41	88.14	87.9	87.88	87.92	87.92	87.82	87.73
VNS	C1	92.78	92.19	91.92	91.46	91.2	91.11	90.64	90.38
CLTRS	C1	93.70	93.44	93.09	92.81	92.73	92.46	92.40	92.40
	C1&C2	92.26	91.48	90.86	90.11	89.51	88.98	88.26	87.57
FDA	C1	92.92	92.49	92.24	91.91	91.83	91.56	91.3	91.02
MLHS	C1	94.54	94.14	93.95	93.61	93.38	93.14	93.06	92.90
	C1&C2	93.12	92.48	91.83	91.23	90.59	89.99	89.34	88.54
HBMLS	C1	94.66	94.30	94.11	93.87	93.67	93.45	93.34	93.14
	C1&C2	93.27	92.60	92.05	91.46	90.91	90.43	89.80	89.24
HBTS	C1, C2&C3	91.93	91.61	91.39	91.13	**90.96**	**90.59**	**90.25**	**89.79**

The filling rates marked in 'bold' are higher than all the ones that are obtained by algorithms in existing literature.

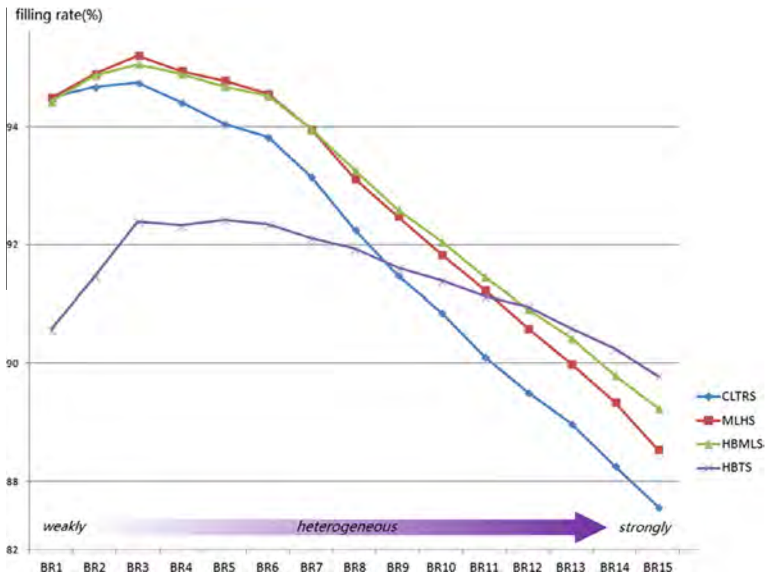

Fig. 9. The trends of filling rates of CLTRS, MLHS, HBMLS and HBTS for BR1–BR15.

Table 4
Computation times of HBTS and other methods for BR1–BR7.

Algorithm	Constraint	Computation time (s)							
		BR1	BR2	BR3	BR4	BR5	BR6	BR7	Mean
H_BR	C1&C2	–	–	–	–	–	–	–	
GA_GB	C1&C2	–	–	–	–	–	–	–	
PTSA	C1&C2	36	48	97	138	179	150	198	
GRASP	C1	–	–	–	–	–	–	–	
MSA	C1	1.27	2.32	4.62	6.52	8.58	12.23	19.25	
HSA	C1&C2	20.33	35.68	59.00	75.05	80.63	88.89	101.52	
A2	C1	–	–	–	–	–	–	–	
VNS	C1	2.98	5.60	11.09	15.12	22.62	31.71	58.00	
CLTRS	C1	–	–	–	–	–	–	–	52
	C1&C2	–	–	–	–	–	–	–	320
FDA	C1	1.16	2.54	5.14	7.66	10.38	16.66	29.54	
MLHS	C1	–	–	–	–	–	–	–	197.33 (BR1–BR15)
	C1&C2	–	–	–	–	–	–	–	187.26 (BR1–BR15)
HBMLS	C1	14.1	34.18	79.43	115.59	155.1	217.57	327.88	
	C1&C2	14.71	36.43	80.33	116.13	153.38	204.15	295.69	
HBTS	C1,C2&C3	61.13	64.37	64.40	63.34	59.52	73.63	86.80	67.60

Table 5
Computation times of HBTS and other methods for BR8–BR15.

Algorithm	Constraint	Computation time (s)								
		BR8	BR9	BR10	BR11	BR12	BR13	BR14	BR15	Mean
H_BR	C1&C2	–	–	–	–	–	–	–	–	–
GA_GB	C1&C2	–	–	–	–	–	–	–	–	–
PTSA	C1&C2	–	–	–	–	–	–	–	–	–
GRASP	C1	–	–	–	–	–	–	–	–	–
MSA	C1	38.20	63.10	97.08	136.50	183.21	239.80	307.62	394.66	–
HSA	C1&C2	261.87	276.12	288.90	287.25	306.36	307.68	305.82	301.26	–
A2	C1	–	–	–	–	–	–	–	–	–
VNS	C1	122.05	141.84	218.05	309.12	375.65	502.25	640.32	788.24	–
CLTRS	C1	–	–	–	–	–	–	–	–	54 (BR1–BR15)
	C1&C2	–	–	–	–	–	–	–	–	320 (BR1–BR15)
FDA	C1	82.94	160.77	298.95	497.79	861.37	1775.79	2218.17	3531.71	–
MLHS	C1	–	–	–	–	–	–	–	–	197.33 (BR1–BR15)
	C1&C2	–	–	–	–	–	–	–	–	187.26 (BR1–BR15)
HBMLS	C1	537.41	730.33	874.59	1050.7	1161.61	1145.13	1256.03	1255.71	–
	C1&C2	454.76	603.94	722.46	842.52	956.2	1019.06	1129.06	1152.71	–
HBTS	C1, C2&C3	125.43	157.17	201.24	236.33	289.22	336.52	355.08	403.08	263.01

HBMLS (by Zhang, Peng, & Leung, 2012). Compared to HBMLS (by Zhang, Peng, & Leung, 2012), HBTS improves performance by 0.05%, 0.16%, 0.45% and 0.55% for BR12–BR15, respectively. Compared to CLTRS (by Fanslau & Bortfeldt, 2010), HBTS obtains the improvement of 0.13%, 0.53%, 1.02%, 1.45%, 1.61%, 1.99% and 2.22% for BR9–BR15.

The average filling rates of CLTRS, MLHS, HBMLS and HBTS for BR1–BR15 are 91.89%, 92.66%, 92.81%, and 91.42%, respectively. Obviously HBTS is weaker than CLTRS, MLHS and HBMLS in most case. From Fig. 9 we can find that the filling rate of HBTS falls slower than the ones of other three algorithms when the instances become more heterogeneous. It proves that HBTS is more suitable for strongly heterogeneous instances than for weakly heterogeneous instances. And HBTS obeys the guillotine cutting constraint. Thus it can be used in the context in which the guillotine cutting constraint is mandatory. Certainly CLTRS, MLHS and HBMLS can be adapted to meet the guillotine cutting constraint. But until now we have not found any publications that present experimental results with this constraint met.

Table 4 and Table 5 report the computation times of HBTS and other algorithms for BR1–BR7 and BR8–BR15, respectively. From the two tables, we can find the computation times of HBTS for BR1–BR15 are reasonable. Despite the fact that HBMLS (by Zhang, Peng, & Leung, 2012) is tested on a platform with a faster CPU, HBTS obtains shorter average computation time for BR1–BR15 than HBMLS (by Zhang, Peng, & Leung, 2012).

As the container loading problems are greatly valuable in practice, many researchers have devoted themselves to dealing with them. The obtained volume utilizations are more and more close to the optimum such that it is increasingly difficult to improve them. HBTS still outperforms all the compared algorithms when the numbers of box types are more than 70, when C1 and C2 are considered.

5. Conclusions

This paper presents a heuristic binary tree search method HBTS for the three-dimensional container loading problem. The algorithm guarantees full support constraint, orientation constraint and guillotine cutting constraint. The algorithm includes several steps. Firstly all the boxes are grouped into strips, and then the strips are further grouped into layers. After this grouping, a binary tree is created. Each of its tree nodes represents a container loading plan which is described as a layer set. In the set some layers are perpendicular to left side of the container while others are to parallel it. For each node, the layer set of its left (or right) child is obtained by inserting a directed layer into its layer set. The directed layer is parallel (or perpendicular) to the left side of the container. Each leaf node denotes a complete container loading plan. A container loading plan is considered as complete if the residual space of the container cannot accommodate any remaining boxes. The solution is the layer set whose total volume of the boxes is the greatest in the tree. The proposed algorithm achieves good results for the well-known 3D-CLP instances suggested by Bischoff and Ratcliff with reasonable computing time.

Besides its obvious advantage in solving strongly heterogeneous problems, HBTS can be further improved to solve weakly heterogeneous ones more effectively. The computational time relies on the searching depth of the binary tree, and the searching depth depends on the number of the layers placed in the container. When the volume of each box decreases, the number of the placed layers will increase. As a result, the computational time of HBTS is relatively long when the boxes are small in size. In addition, since we don't know how to create a strip set that can lead to a good solution, we create multiple strip sets by utilizing the *attraction points* (defined in Formula (11)). In future work we plan to find a method to generate the strip set more efficiently and effectively. At last, each box is supported by only a box and each box also supports only a box in the proposed algorithm. This situation results in a considerable waste of space. In future research, small boxes will be combined into temporary big boxes before being grouped into strips to achieve a better space utilization.

Acknowledgments

The authors thank the editors and the anonymous referees for constructive reviews. This work was supported by the National Nature Science Foundation of China (Grant Nos. 61104054, 71232006 and 61233001).

References

Bischoff, E. E. (2006). Three-dimensional packing of items with limited load bearings strength. *European Journal of Operational Research, 168*(3), 952–966.

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.

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

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*, 1–20.
Davies, A. P., & Bischoff, E. E. (1998). Weight distribution considerations in container loading. *Technical report, Statistics and OR Group, European Business Management School*. Swansea, UK: University of Wales.
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*, 881–892.
Egeblad, J., & Pisinger, D. (2009). Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers and Operations Research, 36*, 1026–1049.
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.
Fekete, S. P., Schepers, J., & van der Veen, J.-C. (2007). An exact algorithm for higher-dimensional orthogonal packing. *Operations Research, 55*, 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 and Operations Research, 7*(3), 147–156.
Hadjiconstantinou, E., & Christofides, N. (1995). An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operations Research, 83*, 39–56.
He, K., & Huang, W. Q. (2010). A caving degree based flake arrangement approach for the container loading problem. *Computers and Industrial Engineering, 59*(2), 344–351.
He, K., & Huang, W. Q. (2011). An efficient placement heuristic for three-dimensional rectangular packing. *Computers and Operations Research, 38*(1), 227–233.
Hemminki, J. (1994). Container loading with variable strategies in each layer. *Presentation, ESI-X, July 2–15, EURO Summer Institute, Jouy-En-Josas, France*.
Hifi, M. (2002). Approximate algorithms for the container loading problem. *International Transactions in Operational Research, 9*(6), 747–774.
Huang, W. Q., & He, K. (2009). A caving degree approach for the single container loading problem. *European Journal of Operational Research, 196*(1), 93–101.
Juraitis, M., Stonys, T., Starinskas, A., Jankauskas, D., & Rubliauskas, D. (2006). A randomized heuristic for the container loading problem: Further investigations. *Information Technology and Control, 35*(1), 7–12.
Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research, 14*(4), 699–719.
Lim, A., Rodrigues, B., & Wang, Y. (2003). A multi-faced buildup algorithm for three-dimensional packing problems. *Omega, 31*(6), 471–481.
Lim, A., Rodrigues, B., & Yang, Y. (2005). 3-D container packing heuristics. *Applied Intelligence, 22*(2), 125–134.
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., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science, 44*, 388–399.
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.
Narendra, P. M., & Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers, 100*(9), 917–922.
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.
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.
Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research, 183*(3), 1109–1130.
Zhang, D. F., Peng, Y., & Leung, S. C. H. (2012b). A heuristic block-loading algorithm based on multi-layer search for the container loading problem. *Computers and Operations Research, 39*, 2267–2276.
Zhang, D. F., Peng, Y., & Zhang, L. L. (2012a). A multi-layer heuristic search algorithm three dimensional container loading problem. *Chinese Journal of Computers, 35*(12), 2553–2561.
Zhang, D. F., Peng, Y., Zhu, W. X., & Chen, H. W. (2009). A hybrid simulated annealing algorithm for the three-dimensional packing problem. *Chinese Journal of Computers, 32*(11), 2147–2156.