OR Spectrum (2014) 36:837–878
DOI 10.1007/s00291-013-0356-1

REGULAR ARTICLE

Container packing problem with balance constraints

Ilkyeong Moon · Thi Viet Ly Nguyen

Published online: 15 December 2013

Abstract This paper presents a new hybrid algorithm that addresses current limitations of container packing problems (CPPs) to increase the efficiency of oceanic transport. The proposed algorithm considers not only the balance constraints, but also the trade-off between volume utilization and weight balance. We also present a mixed integer programming model which features upper and lower bounds for the three-dimensional CPP. The computational results show that our algorithm can efficiently and realistically help shippers make decisions between volume utilization and weight balance and thereby achieve a high rate of volume utilization.

Keywords Container packing · Balance constraints · Hybrid genetic algorithm

1 Introduction

With the rapid increase in global trade and localized economic growth, the transportation sector has grown significantly, becoming a key factor in marketplace success. The most important and most popular type of transit, oceanic transport, is most effective, convenient, and relatively inexpensive when containers are used to carry goods. Many studies on the transportation of goods in containers have developed methods for achieving high levels of economic efficiency, for example, numerous publications address the NP-hard problem of container packing by showing attempts to lower

I. Moon (✉)
Department of Industrial Engineering, Seoul National University, Seoul 151-744, Korea
e-mail: ikmoon@snu.ac.kr

T. V. L. Nguyen
Dalle Molle Institute for Artificial Intelligence (IDSIA-USI/SUPSI),
Galleria 2, 6928 Manno, Switzerland

transportation costs via increased container volume utilization by orthogonally and completely packing multiple items into containers without overlapping. This well-known problem can be classified as a single container packing problem (CPP) or a multi-container packing problem (MCPP). In the former, a set of three-dimensional items must be placed in a container to maximize volume utilization; that is, the ratio of the summed volumes of packed items to the volume of the container should approximate one. In the MCPP, the number of containers is unlimited, but all items must be packed into the minimum number of containers possible. In any CPP, the dimensionalities of the container and the items are categorized in four ways: one dimension (1D), two dimensions (2D), three dimensions (3D), and N dimensions (i.e., $N > 3$). Our study focuses on the three-dimensional container packing problem (3DCPP) because of its real-world applicability.

Many studies have attempted to solve the 3DCPP with different objective functions and constraints by using diverse methods as surveyed and discussed in Crainic et al. (2012), which is the most updated literature on the multidimensional packing problem. In particular, heuristic methods provide good solutions on a large scale presented in NP-hard 3DCPPs. Bortfeldt and Gehring (2001), for instance, used a two-phase method. In the first phase, they built towers from suitable boxes chosen through a greedy heuristic. In the second phase, they applied a genetic algorithm (GA) to place the towers into a container. Martello et al. (2000) presented a two-level branch and bound (BB) approach. In their method, a search tree first determines all possible combinations of containers and items. They then used the BB method to evaluate each node of the first-level search tree and verify the ability to pack items into a container. Many studies have used algorithms that are computationally fast and enable high volume utilization; that is, they show specific arrangements that maximize the number of items that can be packed into a container. For example, Bischoff and Ratcliff (1995) proposed a wall-building approach, which Pisinger (2002) also used to develop a heuristic algorithm. Bischoff et al. (1995) developed an algorithm based on a container filled with blocks, and Eley (2002) also used a block-loading approach. Gehring and Bortfeldt (1997) proposed stack building. Bortfeldt and Gehring (1998) presented an approach that employs layers. An extreme point (EP) rule and EP-based heuristics used to efficiently pack items inside a container were introduced in Crainic et al. (2008).

Recently, meta-heuristic methods have been widely employed to solve complex optimization problems. Zhang et al. (2007) proposed a heuristic algorithm based on simulated annealing (SA) to solve a 3DCPP. Bortfeldt et al. (2003) introduced a parallel tabu search algorithm and later Crainic et al. (2009) presented a two-level tabu search algorithm in which the number of containers is reduced in the first level, and in the second level container packing is optimized for items with fixed orientations based on an interval graph representation of packing. Parreno et al. (2008) proposed a new, greedy, randomized, adaptive search procedure (GRASP) algorithm for solving both the 2DCPP and the 3DCPP; this procedure employs a maximal-space heuristic method in the constructive phase and combines several newly designed steps with a variable neighborhood descent structure in the improvement phase. In 2010, they proposed a variable neighborhood search (VNS) method, an improvement of the variable neighborhood descent method. Regarding multidimensional multi-container packing, an efficient meta-heuristic, greedy adaptive search procedure (GASP) was introduced in Perboli et al. (2011). Hybrid approaches have been widely employed to solve the 3DCPP based on their robustness and adaptability for solving complex optimization problems. For instance, Lim et al. (2005) introduced a new algorithm that combines a tree search and a constructive heuristic, while Moura and Oliveira (2005) combined a constructive heuristic and a greedy, randomized, adaptive, search method. Finally, Karabulut and Inceoglu (2005) investigated a GA that is hybridized with the deepest bottom left with fill (DBLF) method.

Although many studies have addressed the 3DCPP, most have focused exclusively on volume utilization and ignored practical requirements, such as operational safety product handling, and the prevention of cargo damage during container shipping. However, in the context of transportation, practical considerations, particularly weight balance, are as important as volume utilization, but with a few exceptions (e.g., Davies and Bischoff 1999; Eley 2002; Liu et al. 2011) few have addressed it. In a notable work, Baldi et al. (2012) considered the 3D knapsack problem under balance constraints. The assumption that an object's center of gravity should approximately equal the geometric midpoint of the bottom of the container characterized past studies that acknowledged balance constraints; however, they often specified a relative value for the weight balance. Furthermore, researchers could help simplify shippers' ability to make optimal decisions by considering the largely uninvestigated trade-off between volume utilization and weight balance. Transportation firms are inclined to pack items in containers by maximizing volumes to reduce transportation costs, but they must simultaneously ensure operational safety during container transportation and handling. In fact, unexpected costs are prevented by ensuring operational safety. We developed an algorithm based on transportation firms' need to gain profits through an optimal trade-off between volume utilization and weight balance.

For our proposed algorithm, we combined of the DBLF, a greedy heuristic, and a basic GA to find the highest possible volume within the allowable range of weight balance. The framework comprises two main processes. First, a method is used to pack boxes into a container with the DBLF strategy and a greedy heuristic. This step allows us to determine the arrangement of boxes that generates the highest possible volume utilization while satisfying all related constraints except those related to balance. Second, we designed and utilized several particular GA operations to improve the quality of the Step 1 solution and to simultaneously address the balance constraint. Experiments were conducted to demonstrate the performance of the proposed algorithm using (BR) Bischoff and Ratcliff (1995) well-known test cases and Baldi et al. (2012) benchmark cases. We used two types of generated data sets: one based on Bischoff and Ratcliff (1995) idea of creating box objects and the other based on Liu et al. (2011) work in real-world circumstances.

The paper is structured as follows. Section 2 provides a problem description, including a definition and a description of the related constraints. Section 3 presents a mixed integer programming (MIP) model, and Sect. 4 illustrates the proposed algorithm. Section 5 reports the computational experiments, and Sect. 6 offers conclusions.

2 Problem description

We assume a set of n three-dimensional cubic items consisting of t types, where item i is of width w_i, length l_i, height h_i, and weight m_i, $i = (1, 2, \ldots, n)$ and a container of fixed width W, length L, and height H. All items of the same type have the same characteristics. The three-dimensional single container packing problem (3DSCPP) consists of orthogonally packing items into a single container to optimize the volume utilization of the container and to ensure that several packing principles are always permissible. These principles include the following: each item is parallel to the boundary surfaces of the container; no item can penetrate the boundary surfaces of the container; and no pair of packed items overlaps. In the Cartesian coordinate system, let the coordinates of the origin O(0,0,0) correspond to the lower back left corner of the container, and let the length, width, and height of the container be oriented with the x, y, and z axes, respectively. In addition, each item is packed into the container in a manner in which its lower back left corner is located at the selected position. Figure 1 illustrates the container and a box in the Cartesian coordinate system.

The final goal of the problem is to maximize the volume utilization R. The value of R is the ratio of the total volume of packed items to the volume of the container.

$$R = \frac{\sum_{i=1}^{k} V_i}{V_{\text{container}}} \tag{1}$$

where V_i is the volume of the ith item, $V_{\text{container}}$ the volume of the container, k the number of packed items, and R the volume utilization.

Additionally, the following practical constraints are also considered to make the 3DCPP more realistic. The first constraint is a rotation constraint that describes an object's ability to rotate items. Each 3D cubic item can be packed into a container, with the allowed orientations. We consider packing for both two-way and six-way rotations. Figures 2 and 3 show the possible rotations of an item in the case of two-way and six-way rotations, respectively.

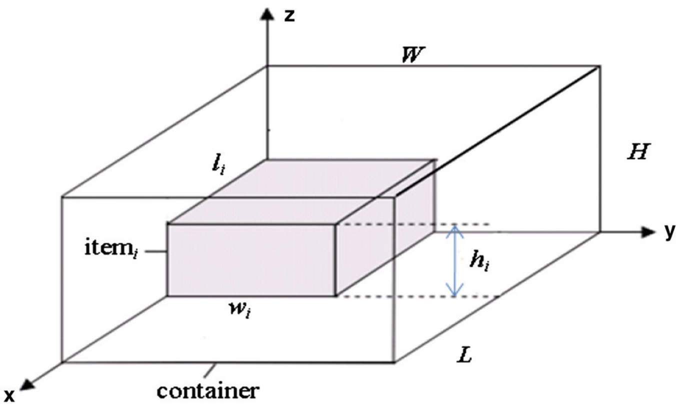


Fig. 1 Container and a box in the Cartesian coordinate system

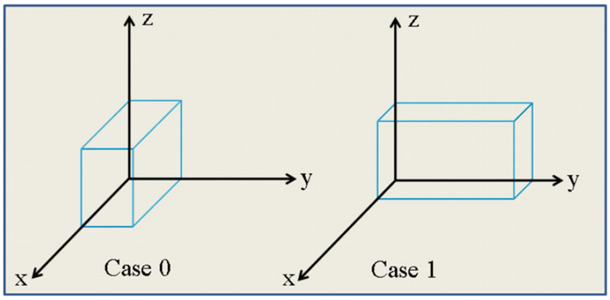


Fig. 2 Box rotational values in the case of two-way rotation

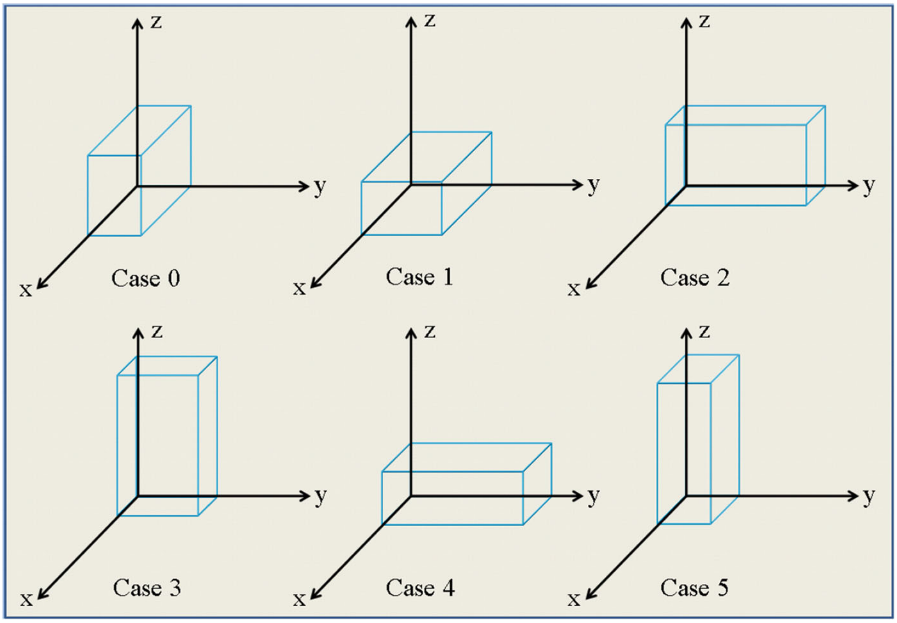


Fig. 3 Box rotational values in the case of six-way rotation

Case 0: No rotation is performed

Case 1: The length is replaced by the width of the item and vice versa; the height remains fixed.

Case 0: No rotation is performed

Case 1: The height is replaced by the width of the item and vice versa; the length remains fixed.

Case 2: The length is replaced by the width of the item and vice versa; the height remains fixed.

Case 3: The length is replaced by the width of the item, and the width is replaced by the height.

Case 4: The length is replaced by the height of the item, and the width is replaced by the length.

Case 5: The height is replaced by the length of the item and vice versa; the width remains fixed.

The second constraint pertains to the stability of the items in a container. This constraint prevents cargo damage during transportation. Boxes packed into the container are stable if they satisfy the following two conditions. First, their bottom faces are fully supported by either the floor of the container or the top faces of other boxes. Second, at least one side of the packed boxes must touch either the side of the container or the sides of other boxes. Figure 4 illustrates the stability constraint. The consideration of this constraint is also essential. However, in a real-world CPP, the volume utilization of the container is always the most crucial objective. Then, the stability of the items in a container increases as a result of the high volume utilization of the container. To our knowledge, most transportation firms deal with this issue by filling the remaining gaps with foam pieces. By applying this strategy, they can prevent cargo damage while achieving volume utilization as high as possible. Therefore, it is explicitly possible to ignore this constraint when we design our algorithm.

The final constraint is the balance constraint. This constraint is vital to prevent the container rotating around its geometric midpoint of the container floor. Davies and Bischoff (1999), Eley (2002), Liu et al. (2011), and Baldi et al. (2012) tried to balance the container by positioning the object's center of gravity as close as possible to the geometric midpoint of the container floor. In particular, in Baldi et al. (2012) where the 3D knapsack problem is considered, to move the center of gravity towards the desired position, the authors gave a 3D convex domain inside the bin and positioned the selected items so that the center of gravity is located inside the given domain.

In our study, the container is filled from the bottom to the top with the items, so that it is almost balanced along the vertical axis. Actually, it is very difficult to obtain an ideal balance along the z axis in which the center of gravity is as close as possible to the bottom of the container while achieving high volume utilization. To balance the container along the x axis, we try to obtain a feasible packing solution in which the relative difference in the moment of force along the x axis does not exceed the given allowable level. We treat the y axis similarly. According to real shipping companies, the given allowable levels that can be accepted are 20 and 10 % for the relative differences in the moment of force along the x axis and y axis. In addition, the expected allowable level is 5 %, and the ideal given allowable level is zero.

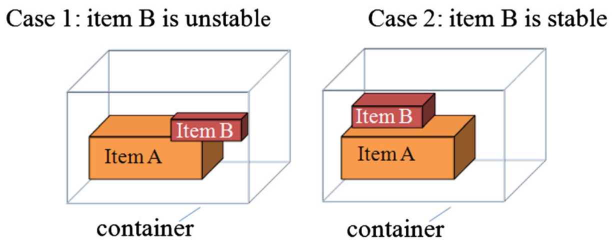


Fig. 4 Stable and unstable items in the container

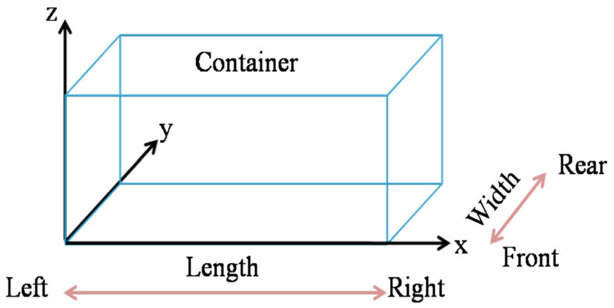


Fig. 5 The difference in behavior with respect to the left–right and front–rear directions corresponds to the dimensions of the container

Specifically, we consider this constraint in three cases: a balance constraint for the left and right sides, a balance constraint for the front and rear sides, and balance constraints for both the left–right and front–rear directions. The difference in behavior with respect to the left–right and front–rear directions corresponds to the dimensions of the container. The left–right direction corresponds to the length of the container and the front–rear direction corresponds to the width of the container (see Fig. 5).

We assume that the center of gravity of a box coincides with its geometric center. Thus, the moment of force is a multiple of the distance from the geometric center of the packed boxes to the rotational axis of the container and the weight of the box:

$$\text{Moment} = \text{Distance} \times \text{Weight} \tag{2}$$

where Moment is the moment of force; Distance is the distance from the geometric center of the packed box to the rotational axis of the container; and Weight is the weight of the packed box.

If the geometric center of the packed box falls in the left half of the container, the corresponding moment of force will be accumulated within the total moment of force on the left side of the container. If the geometric center falls in the right half of the container, the moment of force will be accumulated in the total moment of force on the right side of the container. If the geometric center falls in the front half of the container, the moment of force will be accumulated within the total moment of force in the front of the container. If the geometric center of the packed box falls in the rear half of the container, the moment of force will be accumulated within the total moment of force in the rear of the container. In the first case, the balance constraint for the left and right sides means that the relative difference in the moment of force between the left and right sides must be less than or equal to the allowable left–right level. In the second case, the balance constraint for the front and the rear means that the relative difference in the moment of force between the front and the rear of the container must be less than or equal to the allowable front–rear level. In the third case, the balance constraints for the left–right and front–rear directions mean that the solution must simultaneously satisfy the first and second cases.

One thing that should be mentioned is that, despite using different constraints to balance the container, our approach and those used in other studies have the same target, moving the center of gravity toward the desired position. In this analysis, instead of using the concepts used by other studies, we try to apply the concept of moment to make the trade-off between volume utilization and weight balance visible and applicable. For example, we consider the balance constraints for the left and right sides of the container. We obtain a solution that has the following characteristics:

I: The set of items i whose weight is m_i. The geometric center of each item i falls in the left half of the container and the distance between its geometric center and the geometric midpoint of the container floor is d_i

J: The set of items j whose weight is m_j. The geometric center of each item j falls in the right half of the container and the distance between its geometric center and the geometric midpoint of the container floor is d_j

The distance between the center of gravity of packing and the geometric midpoint of the container floor is d_c.

We can calculate d_c based on the following formula:

$$\frac{\left|\sum_{i\in I} m_i \times d_i - \sum_{j\in J} m_j \times d_j\right|}{\sum_{i\in I} m_i + \sum_{j\in J} m_j} = d_c \tag{3}$$

The relative difference in the moment of force between the left and right sides (RD) must be less than or equal to the allowable left–right level (AL) as follows:

$$\mathrm{RD} = \frac{\left|\sum_{i\in I} m_i \times d_i - \sum_{j\in J} m_j \times d_j\right|}{\sum_{i\in I} m_i \times d_i + \sum_{j\in J} m_j \times d_j} \le \mathrm{AL} \tag{4}$$

If AL is at the ideal allowable level AL $= 0$, we will have $\left|\sum_{i\in I} m_i \times d_i - \sum_{j\in J} m_j \times d_j\right| = 0$. Subsequently, $d_c = 0$ this means that the center of gravity of packing is positioned at the geometric midpoint of the container floor.

3 An MIP model of the container packing problem with balance constraints

In this section, we present an MIP model of the problem described above. Although the MIP model fails to solve the problem for large size instances, it would be useful to give the optimal solution for small size instances and to calculate upper and lower bounds for large size instances. Therefore, we develop this model to evaluate the performance of our proposed algorithm based on the comparison between the results obtained by the MIP model and those of our algorithm. In fact, the MIP model presented here is not the main contribution of our study. It is just a modification of the MIP model developed by Baldi et al. (2012).

First of all, the obvious difference between our model and that of Baldi et al. (2012) is the objective function: they consider the knapsack problem, which values the total profit of the selected items, whereas we consider the CPP, which values the total volume of the selected items. The second clear distinction is the balance constraints, in that different approaches are used to balance the container, as stated above. The remaining constraints are the very basic constraints for 3D packing; therefore, there are no significant differences between ours and those of Baldi et al. (2012). However, in the MIP model of Baldi et al. (2012), the authors gave a large number M used in Big-M constraints, such as the constraint used to make sure that if an item is not packed into the container, its coordinates are zero and the constraint ensuring that there is no overlap between two packed items. A very important consideration is that, when solving the MIP model, an arbitrarily large M will lead the relaxations used by the MIP solver to be weak; therefore, computation time is increased because branching is excessive. Accordingly, it would be a significant benefit for us to keep M as small as possible. In our MIP model, to make these Big-M constraints valid, we set M to the size of the container along the dimension considered in the constraint. The definitions of the indices, parameters, and decision variables used to formulate the MIP model are presented below.

Indices:

i, j	Indices for items ($i, j = 1, \ldots, n$)
d	Indices for dimensions ($d = 1, \ldots, 3$)
r	Indices for rotations ($r = 1, \ldots, 6$)

Parameters:

s_{ird}	Size of item i rotated with rotation r along dimension d
S_d	Size of container along dimension d
m_i	Weight of item i
AL_d	Given allowable level of difference in moment of force along dimension d
V_i	Volume of item i
V	Volume of container

Decision variables:

x_i	If item i is packed into the container,$x_i = 1$; otherwise, $x_i = 0$
R_{ir}	If item i is rotated with rotation r, $R_{ir} = 1$; otherwise, $R_{ir} = 0$
C_{id}	Coordinate of the packed position of the item i along dimension d
y^+_{ijd}	A binary variable. Along dimension d, if item i is packed and item j is not packed, $y^+_{ijd} = 0$
y^-_{ijd}	A binary variable. Along dimension d, if item j is packed and item i is not packed, $y^-_{ijd} = 0$

The MIP model for the 3DCPP with balance constraints is formulated as follows:

$$\text{Max} \sum_{i=1}^{n} V_i x_i \tag{5}$$

Subject to

$$\sum_{i=1}^{n} V_i x_i \leq V \tag{6}$$

$$C_{id} \leq S_d x_i \quad \forall i = 1, \ldots, n \text{ and } \forall d = 1, \ldots, 3 \tag{7}$$

$$C_{id} \geq 0 \quad \forall i = 1, \ldots, n \text{ and } \forall d = 1, \ldots, 3 \tag{8}$$

$$\sum_{r=1}^{6} R_{ir} = x_i \quad \forall i = 1, \ldots, n \tag{9}$$

$$C_{id} + \sum_{r=1}^{6} s_{ird} R_{ir} \leq S_d \quad \forall i = 1, \ldots, n \text{ and } \forall d = 1, \ldots, 3 \tag{10}$$

$$C_{id} + \sum_{r=1}^{6} s_{ird} R_{ir} - S_d (1 - y_{ijd}^{+}) \leq C_{jd} \quad \forall i, j = 1, \ldots, n \, and \, i < j \quad \forall d = 1, \ldots, 3 \tag{11}$$

$$C_{jd} + \sum_{r=1}^{6} s_{jrd} R_{jr} - S_d (1 - y_{ijd}^{-}) \leq C_{id} \quad \forall i, j = 1, \ldots, n \, and \, i < j \quad \forall d = 1, \ldots, 3 \tag{12}$$

$$x_i + x_j - 1 \leq \sum_{d=1}^{3} (y_{ijd}^{+} + y_{ijd}^{-}) \quad \forall i, j = 1, \ldots, n \, and \, i < j \tag{13}$$

$$y_{ijd}^{+} \leq x_j \quad \forall i, j = 1, \ldots, n \, and \, i < j \quad \forall d = 1, \ldots, 3 \tag{14}$$

$$y_{ijd}^{-} \leq x_i \quad \forall i, j = 1, \ldots, n \, and \, i < j \quad \forall d = 1, \ldots, 3 \tag{15}$$

$$\begin{aligned} &\sum_{i=1}^{n} \left[\left[C_{id} + \sum_{r=1}^{6} \frac{1}{2} (s_{ird} R_{ir}) - \frac{S_d}{2} \right]^{+} + \left[C_{id} + \sum_{r=1}^{6} \frac{1}{2} (s_{ird} R_{ir}) - \frac{S_d}{2} \right]^{-} \right] m_i \\ &\quad \leq AL_d \sum_{i=1}^{n} \left[\left[C_{id} + \sum_{r=1}^{6} \frac{1}{2} (s_{ird} R_{ir}) - \frac{S_d}{2} \right]^{+} \right. \\ &\qquad \left. - \left[C_{id} + \sum_{r=1}^{6} \frac{1}{2} (s_{ird} R_{ir}) - \frac{S_d}{2} \right]^{-} \right] m_i \quad \forall d = 1, 3 \end{aligned} \tag{16}$$

$$x_i \in \{0, 1\} \quad \forall i = 1, \ldots, n \tag{17}$$

$$R_{ir} \in \{0, 1\} \quad \forall i = 1, \ldots, n \text{ and } \forall r = 1, \ldots, 6 \tag{18}$$

$$y_{ijd}^{+} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n \text{ and } i < j \text{ and } \forall d = 1, \ldots, 3 \tag{19}$$

$$y_{ijd}^{-} \in \{0, 1\} \quad \forall i, j = 1, \ldots, n \text{ and } i < j \text{ and } \forall d = 1, \ldots, 3 \tag{20}$$

The objective function (5) of this model is to maximize the total volume of the packed items. Constraint (6) is the capacity constraint that ensures that the total volume of the packed items does not exceed the volume of the container. Constraints (7) state that if an item is not packed into the container its coordinates are zero. Constraints (8) indicate that the coordinates of the positions of packed items must not be negative. Constraints (9) state that, if an item is packed into the container, it has only one value of packed rotation; otherwise, it is not assigned a value of rotation. Constraints (10) ensure that items must lie inside the container.

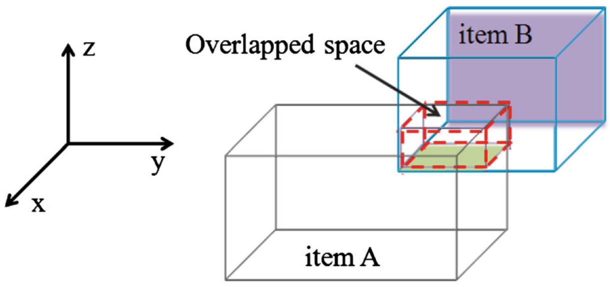


Fig. 6 Illustration of overlap

Figure 6 presents an illustration of overlap between two packed items. It can be seen that the overlap of two packed items A and B occurs when the x, y, and z-coordinates of the lower back left corner of item A are less than the x, y, and z-coordinates of the top front right corner of item B, and the x, y, and z-coordinates of the lower back left corner of item B are less than the x, y, and z-coordinates of the top front right corner of item A. Therefore, a set of constraints (11)–(15) expressing the overlap constraints are used to prevent the existence of the instance stated in the feasible solutions. Constraints (16) state the balance constraints in which $[a]^+ = \max(a, 0)$and $[a]^- = \min(a, 0)$. Finally, constraints (17)–(20) state that the decision variables considered in the constraints are binary variables.

As stated by Baldi et al. (2012), the MIP model is the only currently available approach that can give the optimal solution for the 3D packing problem with the balance constraints for small size instances. However, this model would be the best bet to obtain the lower and upper bounds for the 3D packing problem. The upper bound for this problem can be obtained through two approaches. The first method considers only Constraints (6) and (17), which indicates that we deal with this problem as a 1D packing problem to calculate an upper bound. The second method uses the linear relaxation of the MIP model to obtain an upper bound. As mentioned by Baldi et al. (2012), these two methods provide the same quality of the upper bound of the problem. Thus, we apply the first method to obtain an upper bound.

4 Hybrid genetic algorithm

The 3DCPP is an NP-hard problem, and it becomes more difficult as practical requirements are considered simultaneously. Therefore, exact algorithms such as MIP methods cannot solve this problem for large size instances. Instead, hybrid approaches are used to solve the 3DCPP based on their robustness and adaptability in solving complex optimization problems. Our proposed algorithm, a hybrid genetic algorithm (HGA), was developed in the spirit of hybrid approaches. It is a combination of the DBLF strategy, a greedy heuristic, and a basic GA. The aim of this algorithm is to solve the CPP with balance constraints. Its operation is separated into two main activities, as presented in Fig. 7. The first activity is called the packing strategy. It is used to pack boxes into a container using the DBLF strategy and a greedy heuristic in such a way

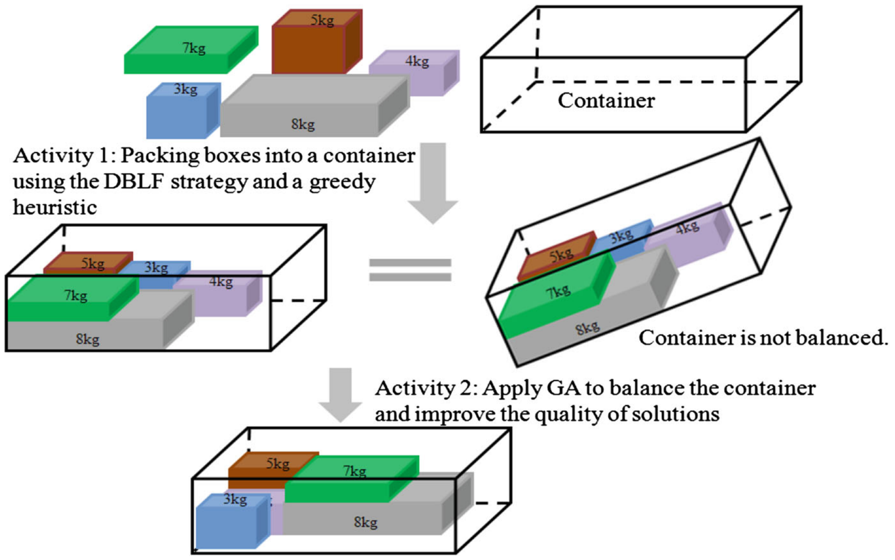

7kg
5kg
4kg
3kg
8kg
Container
Activity 1: Packing boxes into a container using the DBLF strategy and a greedy heuristic
Container is not balanced.
Activity 2: Apply GA to balance the container and improve the quality of solutions

Fig. 7 A summary of the proposed hybrid genetic algorithm

that the arrangement of boxes utilizes the most volume while satisfying all related constraints except the balance constraints. The second activity is to further improve the quality of solutions and to deal with balance constraints simultaneously via several special GA operations. The details of the packing strategy and the HGA are presented in Sects. 4.1 and 4.2, respectively.

4.1 Packing strategy

To ship the greatest volume while satisfying all related constraints, a company needs a good packing strategy. Published studies describe various methods of arrangement, such as blocks, walls, layers, towers, and boxes to pack a container. To reduce the number of fragmented spare spaces, which are difficult to fill, and thereby achieve high levels of cargo stability and volume utilization, Moura and Oliveira (2005) proposed a wall-building approach consisting of columns that contain identical box types. This approach overcomes the weaknesses of box loading by reducing the number of spare spaces that cannot be filled. In this paper, we propose a block-building approach to retain and improve the strengths of the wall-building approach, but adapt solution to the CPP by addressing balance constraints. Moreover, we specify a new placement strategy that consists of the DBLF strategy, a greedy heuristic, and a block-building approach.

4.1.1 Block-building approach

In the GRMod heuristic of Moura and Oliveira (2005), the container is filled with transversal walls consisting of columns with the same box type. To build a new layer, the procedure builds the highest possible column with the first box of a new layer determining the depth of the layer. Then, GRMod replicates this column along the width of the container. Based on the availability of the remaining boxes, incomplete columns are allowed. Therefore, gaps likely exist between any incomplete columns and the free space at the top of the layer, as shown in Fig. 9a. The procedure then fills the free spaces on top and beside the layers with boxes of smaller dimensions than those of the space. If no box can be packed into the free space, GRMod increases the dimensions of the free space by joining contiguous free spaces in previous layers, but this method seldom fills all the gaps and thus does not yield the best solution.

In this study, we modify the wall-building approach of Moura and Oliveira (2005) to improve the volume utilization in the container. First, we consider the available positions to pack boxes into the container instead of determining available spaces. This modification allows us to adapt to the balance constraint and to reduce the fragmented spare spaces that are difficult to fill with the remaining boxes as per the GRMod. Furthermore, we fill the container with 3D blocks instead of walls and do not allow incomplete columns or incomplete layers when building the block. This second modification helps diminish gaps.

Specifically, as boxes are packed into the selected position, the procedure begins by assembling a 3D block into the selected position; the cubic block consists of boxes of the same type and with the same rotation as the chosen box. The dimensions of the container, the occupied space, and the availability of the remaining boxes, which are of the same type as the chosen box, will limit the dimensions of the block. At the selected position, the container is filled to the widest extent possible with boxes represented as integers. Then, the length of the container is filled to the greatest extent possible with columns also represented as integers. After this step, a layer that is parallel to the floor of the container is created, and the height of the container is filled to the greatest extent possible with these layers (again represented by integers). Incomplete columns and layers are not allowed. In addition to achieving the best possible volume utilization, our model handles the weight distribution by building the blocks from layers parallel to the floor of the container instead of using transversal walls. By filling the container first along the width of the container and then along the length of it, our algorithm improves the weight distribution, particularly if the boxes are heavy.

In fact, as described in Sect. 1, use of blocks to model shipping container fulfillment is not new; however, we employ blocks in a unique algorithm to account for balance constraints, and therefore the methods employed to build the blocks are also different from those used by past researchers. In particular, we build the block to fit the given space. In addition, previous algorithms allow for incomplete columns in the block (see Bischoff et al. 1995; Davies and Bischoff 1999; Eley 2002; Bortfeldt et al. 2003; Parreno et al. 2008). In contrast, the spaces in our approach are used solely in the greedy heuristic to find a suitable position and do not influence the shape of the block. Once the position for the given box is determined, the algorithm builds a block that consists of boxes of the same type and with the same rotation as the given box so that the column is as wide, long, and tall as possible. Figures 8 and 9 present the procedure for building the block and the differences between our block-building approach and the wall-building approach of Moura and Oliveira (2005), respectively.

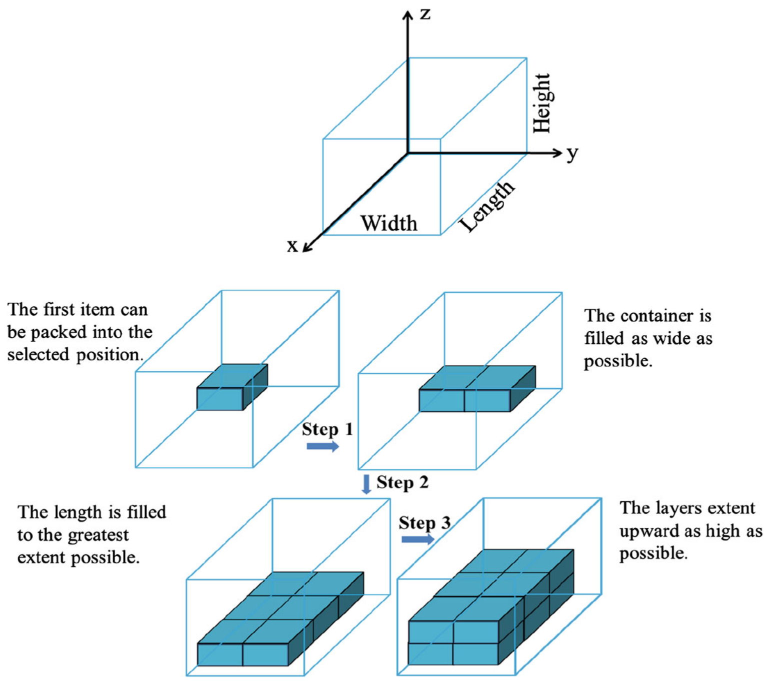

z
y
x
Height
Length
Width
The first item can be packed into the selected position.
Step 1
The container is filled as wide as possible.
Step 2
The length is filled to the greatest extent possible.
Step 3
The layers extent upward as high as possible.

Fig. 8 The procedure for building the block

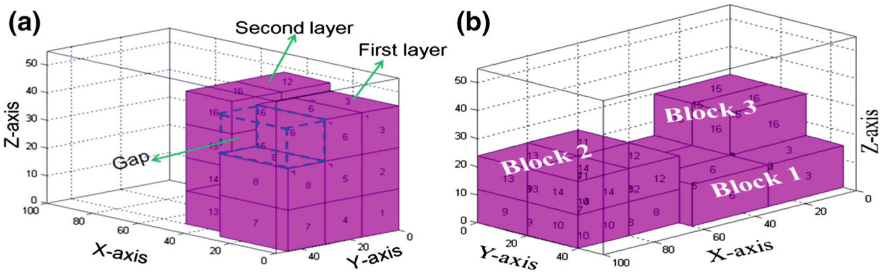


Fig. 9 The differences between the wall-building approach (**a**) of Moura and Oliveira (2005) and our block-building approach (**b**)

4.1.2 The DBLF strategy

Previous studies, e.g., Karabulut and Inceoglu (2005), indicate that the DBLF strategy is accepted and widely used to improve container volume utilization and lessen computation time. Karabulut and Inceoglu (2005) employ a mechanism that uses the DBLF strategy to pack the items into a container and requires that the items be packed into the first allowed DBLF position. The DBLF position is the position that satisfies the following conditions. First, its x-coordinate is the smallest x-coordinate among the available positions. If more than one position satisfies the first condition, then the second condition requires that the z-coordinate is the smallest z-coordinate among the

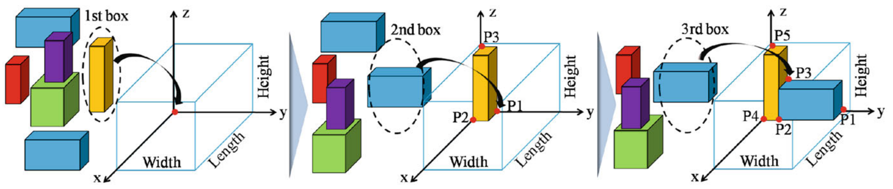


Fig. 10 Mechanism for packing items into a container based on the DBLF strategy

available positions. If more than one position satisfies the second condition, then the third condition requires that the *y*-coordinate is the smallest *y*-coordinate among the available positions. The first allowed DBLF position is that position where the next item can be packed into the container. As illustrated in Fig. 10, the first box is always packed at the lower back left corner of the container. Then, the second box is packed at position P1, and the third box is packed at position P3. From a balancing point of view, the DBLF strategy tends to move the center of gravity of the packed items to the deepest bottom left corner of the container; so, generating a solution that satisfies the balance constraint is quite difficult. However, the most important objective in the CPP is the container volume utilization. Therefore, we rigorously apply the DBLF strategy specified in Karabulut and Inceoglu (2005), thus sorting the available position list that is used by the greedy heuristic when the greedy heuristic attempts to find a position, specifically, position A, to produce a combination of the next item and a packed item.

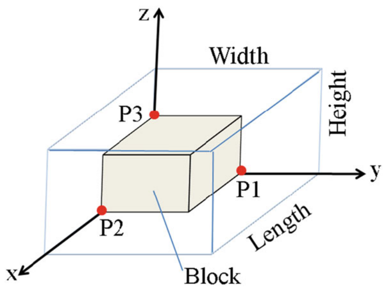
4.1.3 Greedy heuristic

We apply the main idea of the greedy heuristic of local optimization for each iteration in our algorithm to find the optimal packing position. The greedy heuristic is designed to reduce the spare space that is difficult to fill by seeking a position where the next item can be completely combined with a packed item and to scatter the weight distribution in the container. Before giving a further explanation for this heuristic, the generation of the available position list, the empty space containing an available position, and the criteria for choosing a suitable combination of position and orientation for packing an item are presented.

In our algorithm, the first item chosen for placement into a container is always placed in the lower back left corner of the container. In the Cartesian coordinate system, let the coordinate origin O(0,0,0) correspond to the lower back left corner of the container. Thus, the first position in the position list is (0,0,0). Three new positions are generated after packing a block into the container: the lower back right corner of the packed block (P1), the lower front left corner of the packed block (P2), and the top back left corner of the packed block (P3). Figure 11 presents these three new positions.

In our packing strategy, spaces do not influence the shape of the block; they are only used in the greedy heuristic to find a suitable position where the next item can be completely combined with a packed item to reduce the spare space that is difficult to fill. This consideration means that the next item might have a combination with a packed item in which the next item is on the top or in front or beside the packed item.

Fig. 11 The three new positions



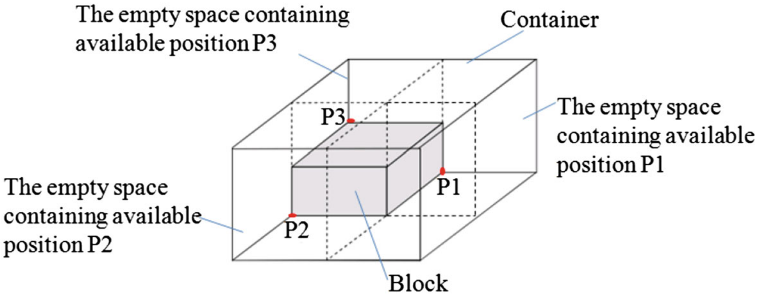

The empty space containing available position P3
Container
P3
The empty space containing available position P1
P1
The empty space containing available position P2
P2
Block

Fig. 12 Empty space containing available positions P1, P2, and P3 stated in Fig. 11 in the position list

Thus, the empty space containing the available positions in the position list is defined as follows.

- If the position is position P1 in Fig. 11, the length of the empty space will be equal to the length of the packed block, the width of the empty space will be equal to the distance between this position and the right side of the container, and the height of the empty space will be equal to the distance between this position and the top of the container.
- If the position is position P2 in Fig. 11, the length of the empty space will be equal to the distance between this position and the front of the container, the width of the empty space will be equal to the width of the packed block, and the height of the empty space will be equal to the distance between this position and the top of the container.
- If the position is position P3 in Fig. 11, the length of the empty space will be equal to the length of the packed block, the width of the empty space will be equal to the width of the packed block, and the height of the empty space will be equal to the distance between this position and the top of the container.

The empty space containing available positions P1, P2, and P3 stated in Fig. 11 in the position list is illustrated in Fig. 12.

In the first phase, the greedy heuristic aims at reducing the fragmented spare spaces that are difficult to fill with the remaining boxes by searching for a perfect combination of the next item and the packed items. The criteria used for the first goal include matching the width and length of the box to be packed and the available spaces within the container. Because there is no consideration for height, each box may have more than one suitable combination. In this case, the first combination found is chosen.

Because the DBLF strategy is used to sort the available position list to make use of position list A for the first phase, the container tends to be unbalanced, as explained earlier. Thus, in the second phase, the greedy heuristic tries to balance the container as much as possible. The criterion used for this phase is that the items must be packed into a position that is as close as possible to the right side, the front side, and the bottom of the container. This prioritization implies that position list B used for the second phase is created by sorting the available position list in an opposite manner to that employed by the DBLF strategy.

4.1.4 Procedure for finding a suitable position

We introduce the procedure of seeking a suitable position by combining the DBLF strategy and the greedy heuristic in two phases (see Fig. 13). Starting with an available position and item lists, the first item is given an original orientation. The first phase is conducted as follows:

- Position list A is created by sorting the available position list based on the DBLF strategy.
- All combinations of the positions in list A and the selected possible orientations of the item are checked. If the empty space containing the selected position satisfies the first criterion of the greedy heuristic (i.e., the width and the length of the box are packed to fit the available container space) and if the item can be packed into the selected position, then the position number (A) is recorded.
- The first phase ends when a suitable position is found or when all positions in list A have been checked.

The first phase creates an unbalanced container. When terminating the first phase, if the suitable position has not been found, the procedure will attempt to find another position (B) so that the container is balanced. The second phase proceeds as follows:

- The initial position, based on the second criterion of the greedy heuristic, namely, that the boxes are arranged in a right-bottom-front order, is found to create list B.
- All combinations of the positions in list B and the selected possible orientations of the item are checked.
- If the item can be packed based on this criterion, the position number (B) is recorded.
- The second phase is terminated if either a suitable position is found or if all positions in list B have been checked.

4.1.5 Procedure for the proposed packing strategy

The proposed placement strategy ensures that each item is placed such that the volume of unfillable spaces is minimized and the weight distribution in a container is easy to achieve. Figure 14 presents the procedure for the proposed packing strategy.

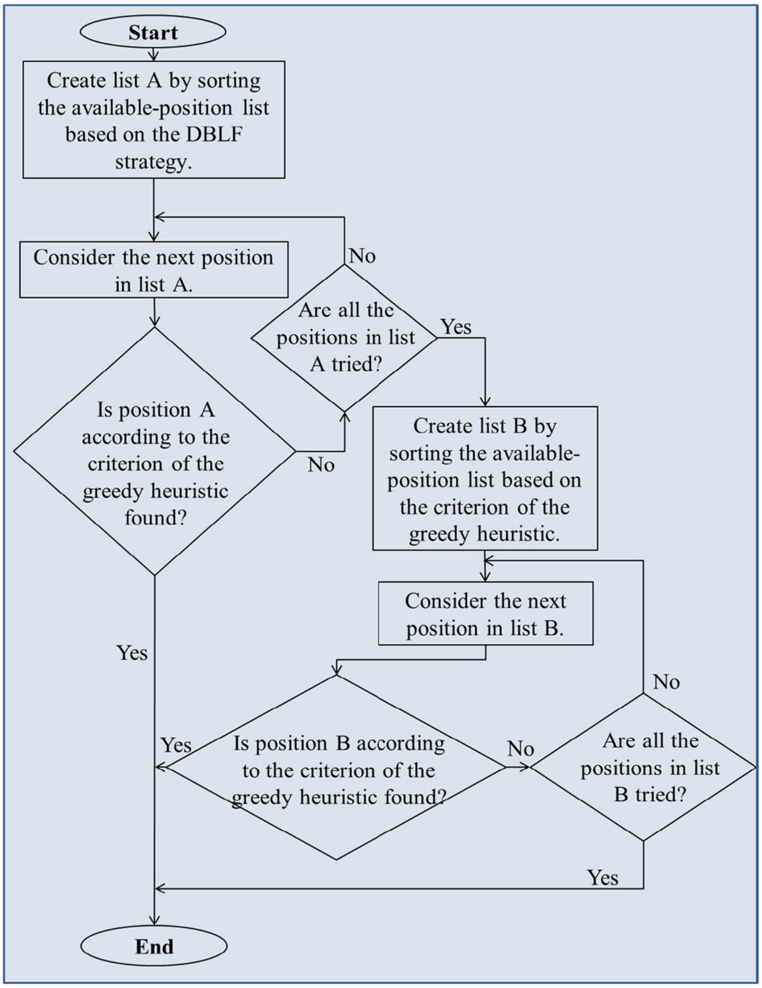

Start
Create list A by sorting the available-position list based on the DBLF strategy.
Consider the next position in list A.
Is position A according to the criterion of the greedy heuristic found?
No
Are all the positions in list A tried?
No
Yes
Create list B by sorting the available-position list based on the criterion of the greedy heuristic.
Consider the next position in list B.
Is position B according to the criterion of the greedy heuristic found?
No
Are all the positions in list B tried?
No
Yes
Yes
Yes
End

Fig. 13 Procedure for seeking a suitable position by a combination of the DBLF strategy and the greedy heuristic

4.2 A hybrid genetic algorithm

Consider a meta-heuristic algorithm; the genetic algorithm is inspired by principles of natural evolution, including inheritance, selection, crossover, and mutation. Genetic algorithms (GAs) can successfully find good solutions for many optimization problems in scientific and industrial contexts, although they are not certain to give the global optimum solution; thus, they have been studied by many researchers (e.g., Gen and Lin 2012). However, the basic GAs rarely achieve good performance when applied to complex problems. As stated by Karabulut and Inceoglu (2005), "since GAs are general methods, they do not contain valuable domain information; so hybridization with domain information can dramatically increase the quality of the solution and the performance of the genetic algorithm by "smarter" exploration of the search space".

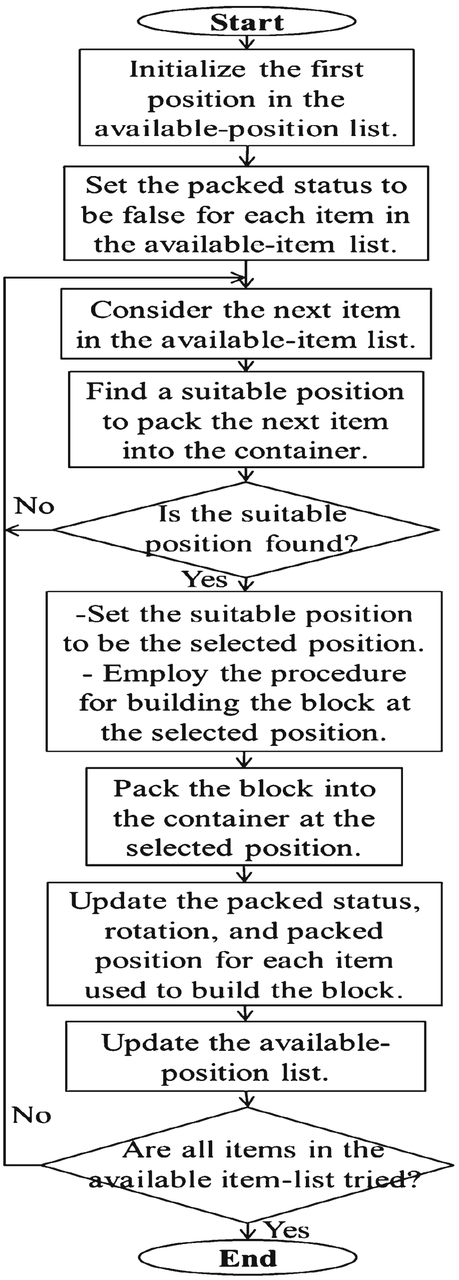

Start
Initialize the first position in the available-position list.
Set the packed status to be false for each item in the available-item list.
Consider the next item in the available-item list.
Find a suitable position to pack the next item into the container.
No
Is the suitable position found?
Yes
-Set the suitable position to be the selected position.
- Employ the procedure for building the block at the selected position.
Pack the block into the container at the selected position.
Update the packed status, rotation, and packed position for each item used to build the block.
Update the available-position list.
No
Are all items in the available item-list tried?
Yes
End

Fig. 14 Procedure for the proposed packing strategy

To our knowledge, there are several different approaches to conduct the hybridization of the genetic algorithm to greatly enrich the efficiency of the basic GA. In particular, Bortfeldt and Gehring (2001) presented a hybridization of a greedy heuristic and genetic algorithm to deal with the stability constraint and the balance constraint for

The sequence of the items	b_1	b_2	b_3	…	b_{n-1}	b_n
The rotations of the items	r_1	r_2	r_3	…	r_{n-1}	r_n

Fig. 15 A diploid encoding scheme for chromosomes

The sequence of the items	5	3	1	10	…	*n*
The rotations of the items	3	0	2	1		0

Fig. 16 An example for chromosomes

the single CPP. In addition, Karabulut and Inceoglu (2005) provided a hybrid genetic algorithm integrating the deepest bottom left as a fill method with the genetic algorithm for regular 3D strip packing in which the boxes are not allowed to rotate.

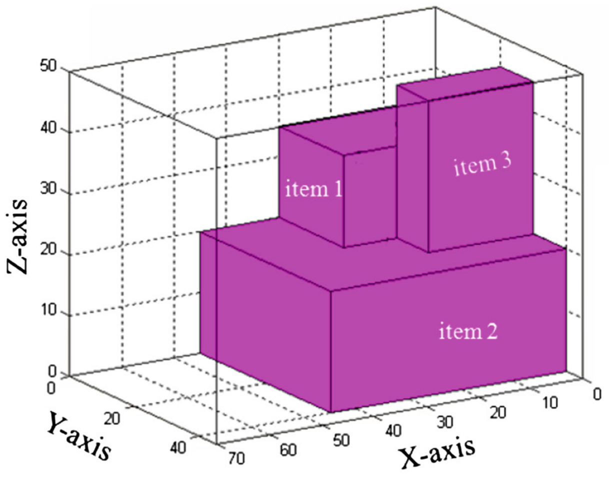
In our study, the container is filled using the packing strategy presented above. The drawback of this strategy is that it is possible to end up with a poor solution because of local criteria; thus, we apply the GA to maintain a set of different feasible solutions, called a population, to obtain the best solution. Therefore, a hybridization of the GA, a greedy heuristic, and the DBLF strategy was developed in this paper. Specifically, the hybridization here occurs during the decoding process for the GA.

Next, the representation and decoding procedure of chromosomes are discussed. In the proposed algorithm, to execute the GA, a chromosome is coded as a two-dimensional array with two rows and n columns where n is the number of items. This is a special diploid encoding scheme for chromosomes. As shown in Fig. 15, each row presents a different characteristic of the chromosome. The first row presents the sequence, which is an ordered series of indices of the item, and the second row presents the rotations of the items. Each component, or gene, of the chromosome is symbolized by a set of properties of an item; the set of properties of the item is a column of the two-dimensional array. From left to right along the chromosome, the first gene is represented by a set of properties of the first item, the second gene by a set of properties of the second item, and the Mth gene by a set of properties of the Mth item.

Figure 16 presents an example for chromosomes.

This representation of a chromosome is still an incomplete solution. In fact, it is just an ordered list of the items. A complete solution is a two-dimensional array with five rows and n columns, where n is the number of items. A column contains a set of properties of an item called a data structure. In a complete solution, this structure consists of five data elements instead of two. These five data elements are the index, rotation, x-coordinate, y-coordinate, and z-coordinate of an item. As illustrated in Fig. 17, a two-dimensional array with five rows and three columns completely describes the solution. To translate a solution of 3D CPPBC to chromosomes to obtain a complete solution, the chromosome decoding procedure is executed. The decoding procedure is exactly the packing strategy (see Fig. 14) in which the available item list is obtained from the representation of a chromosome.

The procedure of the hybrid GA proposed for the CPPBC is presented next.



The sequence of the items	2	1	3
The rotations of the items	0	1	0
The x-coordinates of the items	0	0	0
The y-coordinates of the items	0	0	20
The z-coordinates of the items	0	20	20

Fig. 17 Illustration of a solution

4.2.1 Generating the initial population

A number of chromosomes are created for the first generation. In this first generation, the aim of the proposed algorithm is to create chromosomes with specific schemes of items in which all items of the same type are schemed adjacently and have the same orientation. To achieve this purpose, the sequence of types of items is randomly generated. Then, a specific scheme for the sequence of items in a chromosome is achieved by following the sequence of types of items. To ensure a variety of chromosomes in the first generation, the sequence of items is also generated by sorting items in descending order according to the volume, length, width, or height. Then, the orientation of items is also generated randomly, but this generation must satisfy a constraint; for example, all items of the same type must have the same rotation.

After the sequences and orientations of items have been decided, the chromosomes must be translated into solutions of the 3DCPP through the decoding process. This task is executed based on the packing strategy proposed above. Then, the rate of difference in the moment of force between the left and right sides or the front and rear or between both directions (RD) is calculated. If RD is greater than the maximum allowable rate, the chromosome will be deleted. Otherwise, it will be added to the current population. The process is repeated until the number of chromosomes in the current population is equal to the population size. The process of generating the initial population is illustrated in Fig. 18.

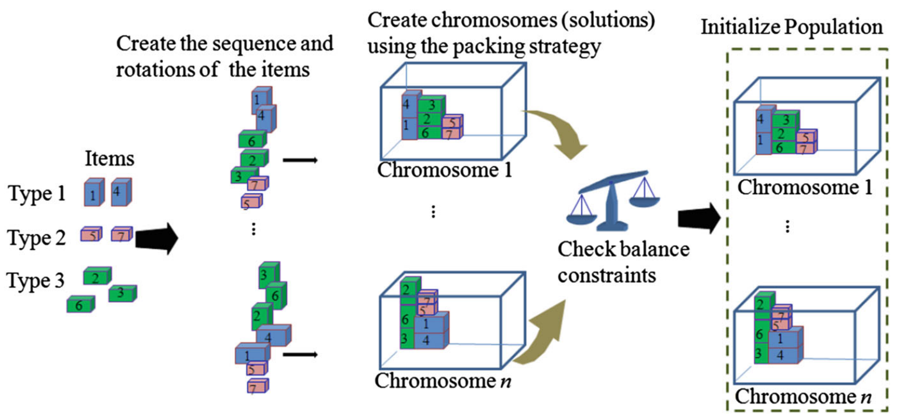

Create the sequence and rotations of the items
Create chromosomes (solutions) using the packing strategy
Initialize Population
Items
Type 1
Type 2
Type 3
Chromosome 1
Chromosome n
Check balance constraints
Chromosome 1
Chromosome n

Fig. 18 The process of generating the initial population

4.2.2 Calculating the fitness of each chromosome and evaluating the current population

This is the maximization problem. Therefore, the following formula is used to calculate the fitness of chromosome i:

$$\text{Fitness}_i = Z_i \Big/ \sum_{i=1}^{\text{PopulationSize}} Z_i \tag{21}$$

Z_i is the objective function value (volume utilization) of chromosome i.

Fitness$_i$ is the fitness value of chromosome i. The best chromosome is the one with the highest fitness value.

To prevent the algorithm from quickly reaching a local optimum, a diversity index (div) is used to evaluate the degree of convergence of the current population at each generation. This index div must measure the degree of convergence of the population. Thus, div is calculated as follows:

$$\text{div} = \frac{\text{BestFitness} - \text{AverageFitness}}{\text{BestFitness}} \tag{22}$$

From Eq. (22), the population tends to converge if the index div is near zero. We set the ratio of the reinitialized chromosomes to the population size at a certain threshold to evaluate the degree of convergence of the population. If div is smaller than the ratio of the reinitialized chromosomes to the population size, the chromosomes at the top of the chromosome list are reinitialized.

4.2.3 Updating the solution and checking the stopping criterion

The best chromosome is the one that yields the best solution to date. In the current population, the best chromosome is chosen, and its fitness value is compared with that of the overall best (incumbent) chromosome. If the current best chromosome has a

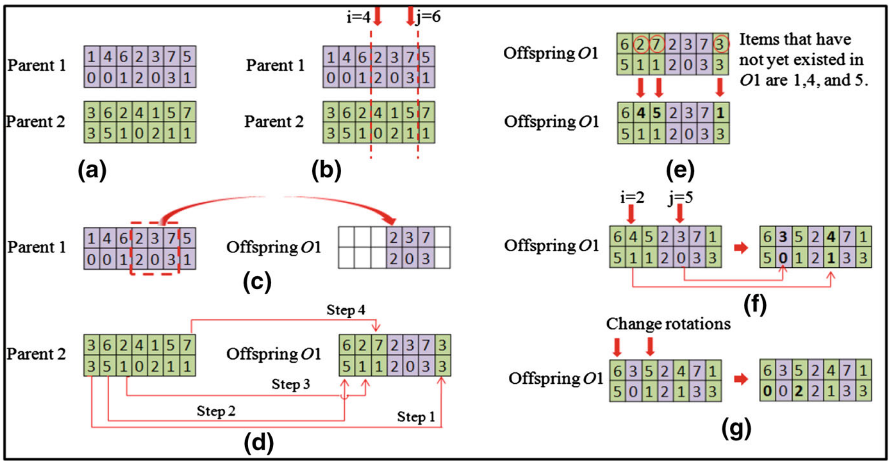


Fig. 19 The crossover and mutation operators

fitness value that is greater than that of the incumbent chromosome, the incumbent chromosome is updated.

The stopping criterion is invoked when the number of generations reaches an upper limit. If the stopping criterion is satisfied, the best chromosome provides the solution. Otherwise, a new generation is created.

4.2.4 *Creating a new generation*

Roulette wheel selection and elitist selection are used to select chromosomes for crossover. The fitness values of the chromosomes are used as the selection probabilities. The design of the crossover and mutation operations can influence the performance of the algorithm. In the proposed algorithm, because of the special diploid chromosome encoding scheme in which each gene of a chromosome encodes not only the sequence of items, but also the rotation of the items, the classical order crossover must be modified for use in the proposed hybrid GA. Therefore, as shown in Fig. 19, crossover and mutation are executed in accordance with the following rules.

- Two selected parents $P1$ and $P2$ are chosen through roulette wheel selection (e.g., Parwananta et al. (2013)) and elitist selection (see Fig. 19a)
- Two-point crossover is used with probability P_c.
- Two cutting sites i, $j (i < j)$ are randomly selected (see Fig. 19b).
- Offspring $O1$ is generated as follows:
  - First, the genes $P1(i)$, $P1(i+1), \dots, P1(j)$ of parent $P1$ are copied into the genes $O1(i)$, $O1(i+1), \dots, O1(j)$of offspring $O1$ (see Fig. 19c).
  - Second, the remaining genes $O1(j+1)$, $O1(j+2), \dots, O1(n)$, $O1(1), \dots,$ $O1(i-1)$ of offspring $O1$ are assigned the genes $P2(1)$, $P2(2), \dots, P2(i-1)$, $P2(j+1), \dots, P2(n)$ of parent $P2$, respectively (see Fig. 19d). During the second process of filling offspring $O1$, if the name of the item of the next gene

chosen from parent P2 is the same as that of the item of the ruling genes used for offspring $O1$, we retain the rotation value and replace the name of the item of that gene with the name of the item that has not yet existed in offspring O1 and is smallest (see Fig. 19e).
 - Two-step mutation is employed to slightly change offspring $O1$. First, we swap two genes with probability P_{m1}, and then we randomly change the rotation of the genes of offspring $O1$ with probability P_{m2}.
   - In the first step of mutation, two sites $i, j\,(i < j)$ are randomly selected. Then, the two genes located at positions i and j are swapped (see Fig. 19f).
   - In the second step of mutation, each gene in the lower half of the offspring is examined to determine whether the rotation should be changed based on a randomly generated probability (see Fig. 19g).

- Offspring $O2$ is filled in the same manner as offspring $O1$.

Then, to translate the solutions of the 3DCPP with balance constraints to chromosomes to obtain the complete solutions, the chromosome decoding procedure is executed, which means that items are then packed into the container based on the aforementioned packing strategies. The rate of difference in the moment of force between the left and right sides or the front and rear or between both directions (RD) is calculated for each offspring. If RD is greater than the maximum allowable rate, the offspring is deleted. Otherwise, it is added to the new population. The process proceeds until the number of chromosomes in the new population is equal to the population size. The current population is replaced by the new population in the next generation.

The proposed hybrid GA procedure is illustrated in Fig. 20.

5 Computational results

Experiments were conducted to evaluate the performance of the proposed algorithm. The parameters used to implement the GA depend greatly on the specific problem. However, in all cases, the population size depends on the chromosomes encoded and the problem considered. In fact, a large population is not useful for improving the GA performance. In addition, the probability of crossover is generally high, but by contrast the probability of mutation is usually rather low. Based on experiments that have already been performed, we designed the parameters used in our algorithm as follows: The population size was set to 100; the probability of crossover was set to 0.8; the mutation probability by swapping a pair of genes was set to 0.2; the mutation probability by changing the rotation of a few genes was set to 0.05; and the triggering threshold in the proposed re-initialization scheme was set to 0.1.

For each test-case experiment, 30 independent runs were executed. For each run, the best fit was recorded, and the maximum allowable number of generations was set to a) 100 for cases in which at least 100 items were to be packed, and b) 1,000, to account for limited computation time, for cases in which fewer than 100 items needed to be packed. The overall performance of the proposed algorithm was defined

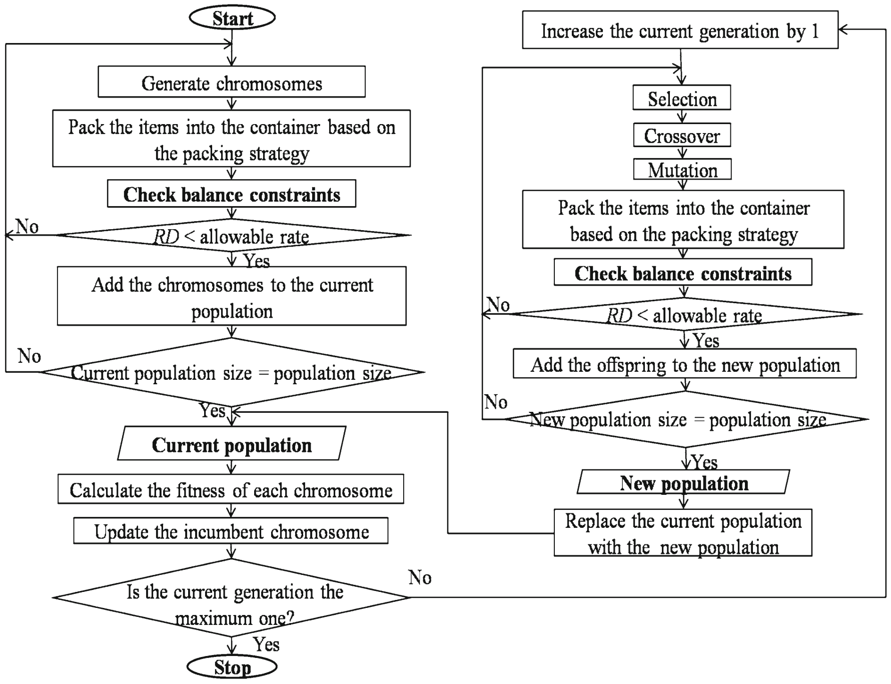

Start
Generate chromosomes
Pack the items into the container based on the packing strategy
Check balance constraints
No
RD < allowable rate
Yes
Add the chromosomes to the current population
No
Current population size = population size
Yes
Current population
Calculate the fitness of each chromosome
Update the incumbent chromosome
Is the current generation the maximum one?
No
Yes
Stop
Increase the current generation by 1
Selection
Crossover
Mutation
Pack the items into the container based on the packing strategy
Check balance constraints
No
RD < allowable rate
Yes
Add the offspring to the new population
No
New population size = population size
Yes
New population
Replace the current population with the new population

Fig. 20 Flowchart of the hybrid GA

as the average of the best fitness across the total number of runs, as formulated in Eq. (23):

$$\text{BestFitness} = \frac{1}{30}\sum_{i=1}^{30} \text{BestFitness}_i \tag{23}$$

All computational tests were conducted with Java in the Windows XP operating system on a PC with an Intel Core 2 Quad 2.4 GHz processor with 2 GB RAM.

5.1 Computational results with balance constraints

5.1.1 Testing the performance stability of the proposed algorithm when considering balance constraints

To confirm that the proposed algorithm is efficient and realistic compared with other algorithms designed to help shippers decide between volume utilization and weight balance, we conducted tests with our own data sets, which were developed based on Bischoff and Ratcliff (1995)'s idea of creating box objects. The number of boxes n was set to 100. The number of box types was set to 100. Therefore, the data set was strongly heterogeneous. The total volume of the boxes was 85,417,800 cm^3; it exceeded that of the container by more than 30 %. The range of box volumes was approximately 1:20 (maximum divided by the minimum). The total weight of the boxes in tons did

Table 1 Combinations of the ranges of volumes and box weights

Range of volumes (cm^{-3})		Range of weights (kg)	
Smallest	Largest	Lightest	Heaviest
		77	769
133,584	2,656,080	47	933
		19	798

Table 2 Combinations of the allowable left–right and front–rear rates

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Allowable left–right rate (%)	5	10	10	20	20	30	No limit
Allowable front–rear rate (%)	5	10	20	20	30	30	No limit

not exceed the weight limit of the container. To demonstrate performance stability when considering balance constraints, the proposed algorithm was executed for three cases, with varying box weights. Specifically, the ranges of the box weights were approximately 1:10, 1:20, and 1:50. The purpose of generating a data set in which the total volume of the boxes is greater than the volume of the container was to create more possible options for choosing boxes to be packed into the container. In addition, these assumptions are suitable for actual transportation operations. The container had the following dimensions: length L = 1200 cm, width W = 233 cm, and height H = 235 cm.

The proposed algorithm was examined for the following scenarios.

- Scenarios for combinations of the ranges of volumes and box weights, as presented in Table 1.
- Scenarios for balance constraints, including a balance constraint for the left and right sides, a balance constraint for the front and rear sides, and balance constraints for both the left–right and front–rear directions.
- Scenarios for values of the allowable left–right and front–rear rates of 5, 10, 20, 30, and 40 %. As stated earlier, the allowable rates that can be accepted by the real shipping companies are 5, 10, and 20 %. In this test, we try to examine the proposed algorithm with allowable rates up to 40 % to check its reality.
- Scenarios for combinations of the allowable left–right and front–rear rates for the third case of balance constraints, as presented in Table 2.

The experimental results under six-way rotation are shown in the tables below. Each table consists of volume utilization (%) results for three box weights. In each table, the first column refers to the range of volumes, the second column refers to the range of weights, and the first row refers to the allowable rates of difference in the moment of force (%). Because of the randomness of the algorithm, each allowable rate of difference in the moment of force is presented in terms of four values: the volume utilization of the best solution, the volume utilization of the worst solution, the average volume utilization, and the standard deviation of the volume utilization. The results under a balance constraint for the left and right sides, a balance constraint for the front and rear sides, and balance constraints for left–right and front–rear directions are presented in Tables 3, 4, and 5, respectively.

The results in Tables 3, 4, and 5 demonstrate the performance stability of our proposed algorithm when considering balance constraints. The average rate of volume utilization was slightly changed for the different combinations of volume and box weight ranges. As shown in Table 3, the limit on the allowable rate of difference in the moment of force significantly influenced volume utilization when the balance constraint for the left and right sides was considered. When the limit was 10 %, the volume utilization was significantly different from that in the case with no limit. When the limit on the allowable rate of difference in the moment of force was increased, the volume utilization approached the volume utilization in the case with no limit; when the allowable rates of difference in the moment of force were 30 and 40 %, the volume utilization was not significantly different from the volume utilization in the case with no limit. Furthermore, for each limit on the allowable rate of difference in the moment of force, the volume utilization changed only slightly for the different weight ranges. The above observations also apply to the cases with a balance constraint for the front and rear in Table 4 and the balance constraints for both directions in Table 5. However, in contrast with the results for the balance constraint for the left and right sides, the balance constraint for the front and rear only slightly influenced the volume utilization. These opposite influences impacted the results for balance constraints in both directions. According to the results in Table 5, the rate of volume utilization was highly variable when the balance constraint was constant for the front–rear direction and variable for the left–right direction. Conversely, the volume utilization changed only slightly when the balance constraint was constant for the left–right direction and variable for the front–rear direction. This result demonstrates that our proposed algorithm is applicable to real operations of packing cargo into containers. Achieving balance of the container for the left and right sides, which corresponds to the length of the container, is much more difficult than achieving balance for the front and rear, which corresponds to the width of the container.

According to the aforementioned observations for six-way rotation, the proposed algorithm works well with balance constraints. It suitably and reliably solves the CPP with balance constraints, and it achieves a fairly high rate of volume utilization while complying with the allowable rates of difference in the moment of force. Moreover, using the proposed algorithm, we can obtain a volume utilization that corresponds to a precise limit on the allowable rate of difference in the moment of force. Such a precise limit on the allowable rate is more significant than the relative values for weight balance that have been used in previous studies. In this limit, the center of gravity of the items in the container is as close as possible to the geometric midpoint of the container floor. Thus, our algorithm can be used to help shippers decide between volume utilization and weight balance when attempting to maximize profits. Because of the way in which the balance constraints are taken into account in our method, chromosomes that do not satisfy the balance constraints are deleted from both the initial and new populations; thus, if the precise limit on the allowable rate is stricter, a large proportion of solutions are discarded. Specifically, computational results show that the average proportions of solutions that are discarded are 62.58, 25.93, and 2.06 % when the balance constraints for both directions are 5–5, 10–10, and 20–20 %, respectively.

Table 3 Volume utilization with a balance constraint for the left and right sides (six-way rotation)

	Data set	Allowable rate of difference in the moment of force in % (left right)	10	20	30	40	No limit
Range of volumes (min–max) = (133,584–2,656,080)	Range of weights (min–max)= (77–769)	The best solution	92.85	93.75	94.40	94.89	95.34
		The worst solution	88.44	91.60	92.40	92.51	92.84
		Average	91.05	92.76	93.55	93.64	93.95
		Standard deviation	0.85	0.61	0.57	0.65	0.66
	Range of weights (min–max)= (47–933)	The best solution	93.42	94.43	94.83	94.63	95.00
		The worst solution	88.38	92.06	92.44	92.69	93.52
		Average	89.84	93.44	93.66	93.77	94.00
		Standard deviation	1.14	0.65	0.67	0.60	0.42
	Range of weights (min–max)= (19–798)	The best solution	92.06	94.41	94.57	94.86	95.06
		The worst solution	87.45	91.93	92.55	93.19	92.78
		Average	90.78	93.37	93.65	93.87	93.85
		Standard deviation	1.19	0.65	0.55	0.50	0.63

5.1.2 Comparisons between our HGA and the MIP model in terms of balance constraints

We evaluate our HGA by comparing its performance with that of the MIP model in terms of volume utilization and computation time. Regarding the balance constraint, the comparison is direct because the HGA and MIP model apply the same definition of it. The data sets used in this experiment are our own generated instances. We randomly generated the instances based on the same approach from the benchmark instances considered by Baldi et al. (2012). The description of the instances is presented as follows:

- The number of items is 20, 40, and 60.
- Items are generated based on two strategies. In the first strategy, five clusters (C) of items are independently generated. In the second strategy, items are randomly (R) generated.

Table 4 Volume utilization with a balance constraint for the front and rear sides (six-way rotation)

Data set		Allowable rate of difference in the moment of force in % (front rear)	10	20	30	40	No limit
Range of volumes (min–max) = (133,584–2,656,080)	Range of weights (min–max)= (77–769)	The best solution	94.26	94.55	94.65	94.97	95.34
		The worst solution	91.87	91.92	92.41	92.59	92.84
		Average	93.04	93.23	93.50	93.74	93.95
		Standard deviation	0.67	0.94	0.61	0.68	0.66
	Range of weights (min–max)= (47–933)	The best solution	94.28	94.72	95.15	94.68	95.00
		The worst solution	91.81	92.80	92.94	93.07	93.52
		Average	93.25	93.74	93.80	93.88	94.00
		Standard deviation	0.68	0.62	0.71	0.60	0.42
	Range of weights (min–max)= (19–798)	The best solution	94.52	95.55	94.52	94.61	95.06
		The worst solution	91.69	91.85	92.44	92.30	92.78
		Average	93.31	93.52	93.67	93.65	93.85
		Standard deviation	0.59	0.84	0.55	0.55	0.63

- The container height always equals its width plus one-half its length. The container volume equals a percentage p of the total volume of items: for example, $p = 50$ % or $p = 90$ %
- The item weights are uniformly distributed along the interval [70,100].
- The item dimensions belong to five distinct classes (see Table 6).
- The profit of item i is $\text{profit}_i = 200 + \text{volume}_i$.

We consider the balance constraints for both the left–right and the front–rear directions when the given allowable gaps are 5 % for the left–right direction and 5 % for the front–rear direction. The MIP model is solved using the commonly available commercial solver CPLEX 12.2. We retain the default parameter values and establish a maximum computation time of 18,000 s. Our HGA is terminated when the number of generations reaches 1,000. We compare the performances of the MIP model and our HGA using the percentage gaps between the upper bound (UB) and the volume utilization obtained by each model. Because of the limited computation time set for the CPLEX to solve the MIP model, we experimented with the benchmark instances such that the number of items is 20 and the volume of the container is 90 % of the total volume of the items. In Table 7, *gap* represents the difference between the UB and the

Table 5 Volume utilization with balance constraints for both directions (six-way rotation)

	Data set	Allowable rate of difference in the moment of force in % (left right–front rear)	5–5	10–10	10–20	20–20	20–30	30–30	No limit
Range of volumes (min–max) = (133,584–2,656,080)	Range of weights (min–max)= (77–769)	The best solution	91.91	92.75	92.97	94.11	94.42	94.45	95.34
		The worst solution	88.98	88.45	89.56	91.53	92.17	92.50	92.84
		Average	89.96	90.67	91.09	92.94	93.05	93.62	93.95
		Standard deviation	0.66	1.13	0.90	0.60	0.59	0.54	0.66
	Range of weights (min–max)= (47–933)	The best solution	90.85	93.16	93.42	94.07	94.58	95.13	95.00
		The worst solution	83.63	87.27	88.51	89.27	92.18	93.12	93.52
		Average	86.13	89.15	90.00	93.24	93.62	93.86	94.00
		Standard deviation	1.66	1.95	1.22	1.02	0.65	0.54	0.42
	Range of weights (min–max)= (19–798)	The best solution	90.73	92.86	92.97	94.10	94.29	94.82	95.06
		The worst solution	78.53	88.88	89.49	92.27	91.96	92.53	92.78
		Average	84.67	90.90	91.28	93.12	93.20	93.62	93.85
		Standard deviation	2.66	1.08	0.93	0.42	0.68	0.72	0.63

Table 6 Geometric classes of dimensions of the items

Class	Alias	Width	Height	Length
C	Cubes	[1, 100]	Equal to width	Equal to width
D	Diverse	[1, 50]	[1, 50]	[1, 50]
L	Long	[1, 200/3]	[1, 200/3]	[50, 100]
U	Uniform	[50, 100]	[50, 100]	[50, 100]
F	Flat	[50, 100]	[50, 100]	[25, 60]

Table 7 Comparison of the MIP model and our HGA

Data	MIP		HGA			
	Gap (%)	Time (s)	Average gap (%)	Min gap (%)	Max gap (%)	Time (s)
ep3d-20-C-C-90	0.00	17,800*	0.00	0.00	0.00	33
ep3d-20-C-R-90	42.50	18,000	47.33	42.55	55.34	145
ep3d-20-D-C-90	0.00	3,400*	0.00	0.00	0.00	32
ep3d-20-D-R-90	9.64	18,000	19.22	13.96	25.92	160
ep3d-20-L-C-90	17.10	18,000	34.52	24.51	48.26	32
ep3d-20-L-R-90	14.33	18,000	20.46	15.19	25.25	153
ep3d-20-U-C-90	16.77	18,000	14.86	14.52	15.11	42
ep3d-20-U-R-90	16.88	18,000	12.69	8.95	16.72	164
ep3d-20-F-C-90	16.91	18,000	16.15	14.58	18.66	42
ep3d-20-F-R-90	14.93	18,000	17.02	13.70	22.77	165

result obtained with the MIP model; *average gap* shows the difference between the UB and the average volume utilization obtained by our HGA; *min gap* points to the difference between the UB and the volume utilization of the best solution obtained by our HGA; and *max gap* denotes the difference between the UB and the volume utilization of the worst solution obtained by our HGA. An asterisk indicates that the optimal solution has been found and that the MIP model terminates before reaching 18,000 s. As stated, for small size instances requiring a limited computation time, the MIP model is efficient in generating good, but not necessarily optimal, solutions. Table 7 indicates that the performances of the MIP model and our HGA are competitive. In terms of computation time, our HGA is quite efficient compared with the MIP model.

To show how the container balances under the constraints, the position of the packing center of gravity is calculated based on Eqs. (24)–(26) for each data set. The number of items packed into the container is n. Item i packed into the container has coordinates $\{x_i, y_i, z_i\}$ and weight m_i. The packing center of gravity has coordinates $\{x_c, y_c, z_c\}$. We have the following equations:

$$\sum_{i=1}^{n} m_i \times x_i = x_c \times \sum_{i=1}^{n} m_i \tag{24}$$

$$\sum_{i=1}^{n} m_i \times y_i = y_c \times \sum_{i=1}^{n} m_i \tag{25}$$

Table 8 Coordinates of the packing center of gravity (COG) for the first comparison

Data	COG			GMC			CoM domain		
	x	*y*	*z*	*x*	*y*	*z*	*x*	*Y*	*z*
ep3d-20-C-C-90	116.98	56.39	35.26	116.00	58.00	0.00	[58.00, 174.00]	[29.00, 87.00]	[0.00, 58.00]
ep3d-20-C-R-90	133.89	66.97	43.98	134.00	67.00	0.00	[67.00, 201.00]	[33.50, 100.50]	[0.00, 67.00]
ep3d-20-D-C-90	41.86	19.83	15.87	41.00	20.50	0.00	[20.50, 61.50]	[10.25, 30.75]	[0.00, 20.50]
ep3d-20-D-R-90	48.91	24.20	23.20	49.00	24.50	0.00	[24.50, 73.50]	[12.25, 36.75]	[0.00, 24.50]
ep3d-20-L-C-90	83.44	38.93	35.14	85.00	42.50	0.00	[42.50, 127.50]	[21.25, 63.75]	[0.00, 42.50]
ep3d-20-L-R-90	82.31	41.31	39.87	83.00	41.50	0.00	[41.50, 124.50]	[20.75, 62.25]	[0.00, 41.50]
ep3d-20-U-C-90	158.34	78.02	*82.42*	158.00	79.00	0.00	[79.00, 237.00]	[39.50, 118.50]	[*0.00*, *79.00*]
ep3d-20-U-R-90	157.23	79.22	*78.66*	157.00	78.50	0.00	[78.50, 235.50]	[39.25, 117.75]	[*0.00*, *78.50*]
ep3d-20-F-C-90	139.59	68.97	63.78	141.00	70.50	0.00	[70.50, 211.50]	[35.25, 105.75]	[0.00, 70.50]
ep3d-20-F-R-90	90.28	45.06	*45.55*	91.00	45.50	0.00	[45.50, 136.50]	[22.75, 68.25]	[*0.00*, *45.50*]

The italicized values indicate that the z-coordinate of the packing center of gravity is out of the domain constraints that are used by Baldi et al. (2012)

$$\sum_{i=1}^{n} m_i \times z_i = z_c \times \sum_{i=1}^{n} m_i \tag{26}$$

Table 8 presents the coordinates of the packing center of gravity (COG) that are obtained from our HGA algorithm, the coordinates of the geometrical midpoint of the container floor GMC $\left\{\frac{\text{Length}}{2}, \frac{\text{Width}}{2}, 0\right\}$, and the domain constraints (CoM domain) $x_{\text{COG}} \in \left\{\frac{\text{Length}}{4}, \frac{3\text{Length}}{4}\right\}$, $y_{\text{COG}} \in \left\{\frac{\text{Width}}{4}, \frac{3\text{Width}}{4}\right\}$ and $z_{\text{COG}} \in \left\{0, \frac{\text{Height}}{2}\right\}$, which present the balance constraints from other studies (see Baldi et al. 2012) for each data set. Length, Width, and Height are the length, width, and height of the container, respectively.

The results presented in Table 8 show that with the balance constraints, in which the given allowable rates are 5 % for the left–right direction and 5 % for the front–rear direction, the container is extremely balanced along its length and width. The *x*-coordinate and *y*-coordinate of the packing center of gravity are very close to the *x*-coordinate and *y*-coordinate of the geometric midpoint of the container floor, and they satisfy the domain constraints that are used by Baldi et al. (2012) and other studies to express the balance constraints. As stated in Sect. 2, it is very difficult to obtain an ideal balance along the *z* axis in which the center of gravity is as close as possible to the bottom of the container while achieving high volume utilization. Thus, the *z*-coordinates of the COGs are far from the *z*-coordinates of the GMCs. Almost all of the *z*-coordinates of the COGs of the ten data sets considered satisfy the domain constraints, except those of the data sets UC, UR, and FR. Although the *z*-coordinates of the data sets COGs of UC, UR, and FR are out of the domain constraints, we can say that the container is still balanced along its height because the difference between the upper limitations and the *z*-coordinates of the data sets COGs of UC, UR, and FR are very small.

Table 9 Similarities between our own data set DS1 and RW1 instances

	Data sets	Box types	Total quantity of boxes	Total weight of boxes (ton)	Total volume of boxes (cm^3)	Varying range of box volume (cm^3)
Our data set (DS1)	1	5	230	35.879	76,272,000	55,000–2,871,000
	2			30.262	76,401,049	
	3			30.970	76,428,078	
	4			35.866	76,372,274	
	5			30.762	76,256,832	
	6			33.157	76,352,678	
	7			34.546	76,436,714	
	8			30.662	76,165,010	
	9			32.514	76,126,380	
	10			33.351	76,196,744	
Liu et al. (2011)	$RW1$	5	230	33.117	76,261,840	55,000–2,871,000

5.1.3 Comparisons between our HGA and Liu et al.'s HTS (2011) in terms of balance constraints

In Sect. 5.1.2, the comparison is based on small data sets. To make a comparison based on large data sets, we continue to compare the performance of our algorithm with that of the hybrid tabu search with the loading heuristic (HTS) algorithm of Liu et al. (2011) using data sets that were generated based on information about RW instances from Liu et al. (2011) The RW instances used for testing in Liu et al. (2011) are real-world data. For each instance, the boxes are packed into a container with dimensions of 12 m $\times$ 2.33 m $\times$ 2.65 m, a gross weight of 3.35 tons, and a gross weight limit of 27.13 tons. To ensure reliability in the comparison, we generated ten data sets for RW1 instance and set the allowable tolerance to 15 % for both total volume and weight. The similarities between our data sets and the RW instances are shown in Table 9.

The computational results were obtained for a six-way rotation. In this comparison, we also consider the balance constraints for both the left–right and front–rear directions when the given allowable rates are 5 % for the left–right direction and 5 % for the front–rear direction. Our HGA is terminated when the number of generations is 100. Comparisons between the results of our algorithm and those of the HTS algorithm of Liu et al. (2011) in terms of the average volume utilization for the RW benchmarks are presented in Table 10.

Although the measure used in our method to address the balance constraints in the 3DCPP is different from that of Liu et al. (2011), we can calculate the coordinates of the packing center of gravity based on Eqs. (24)–(26) above to make an acceptable comparison. Tables 8 and 11 present the coordinates of the packing center of gravity (COG) that are obtained from our algorithm, the coordinates of the geometric mid-point of the container floor GMC $\left\{\frac{\text{Length}}{2}, \frac{\text{Width}}{2}, 0\right\}$, and the domain constraints (CoM domain) $x_{\text{COG}} \in \left\{\frac{\text{Length}}{4}, \frac{3\text{Length}}{4}\right\}$, $y_{\text{COG}} \in \left\{\frac{\text{Width}}{4}, \frac{3\text{Width}}{4}\right\}$ and $z_{\text{COG}} \in \left\{0, \frac{\text{Height}}{2}\right\}$, which present the balance constraints from other studies (see Baldi et al. 2012) for

Table 10 Comparison in terms of the average volume utilization regarding the RW1 benchmark

	Data set	Allowable rate of difference in the moment of force in % (left right–front rear)	
		5–5	No limit
Our own data sets (DS1)	1	89.07	94.54
	2	89.14	93.78
	3	88.26	95.13
	4	88.87	95.55
	5	90.23	96.54
	6	91.51	95.45
	7	85.01	92.71
	8	89.58	96.42
	9	90.35	94.10
	10	89.48	95.02
Liu et al. (2011)	RW1	Consider balance	Neglect balance
		87.43	90.58

each data set. Length, Width, and Height are the length, width, and height of the container, respectively. From the results in Table 11, it can be seen that the container is extremely balanced along its length and width. At the same time, it is also quite balanced along its height. Although the z-coordinates of the data sets 2, 5, and 10 do not satisfy the domain constraints, we can describe the container as balanced because the differences between the upper limitations and the z-coordinates of the COGs of the data sets 2, 5, and 10 are very small. Moreover, Liu et al. (2011) tried to pack items into the container so that the COG was close to the geometric midpoint of the container floor as much as possible and did not impose any upper limitation; thus, we can make an acceptable comparison in this analysis. Table 10 shows that the performance of our proposed algorithm is superior to that of the algorithm of Liu et al. (2011) if the balance constraints are ignored. If the balance constraints are considered, the performance of our HGA also overcomes that of the HTS algorithm of Liu et al. (2011), except for data set number 7. In addition, in terms of computation time, it is difficult to compare because Liu et al. (2011) did not report the computation time used for only BR benchmarks. The HTS algorithm of Liu et al. (2011) provides results within the range of 0.044–280 s for all experiments used for both BR benchmarks and RW data sets, whereas our HGA algorithm provides results within the range of 34–99 s for data sets DS1 from class 1 to class 10.

5.1.4 Comparisons between our HGA and Baldi et al.'s (2012) 3BKP-H in terms of balance constraints

To qualify our results sufficiently, we compared in terms of the total profit between our HGA and Baldi et al. (2012) 3BKP-H, which is the most recent study of the balance constraints in a 3D knapsack problem. The data sets used are exactly the

Table 11 Coordinates of the packing center of gravity (COG) for the second comparison

Data (DS)	COG			GMC			CoM domain		
	x	*y*	*z*	*x*	*y*	*z*	*x*	*y*	*Z*
1	598.16	118.39	91.35	600.00	116.50	0.00	[300.00, 900.00]	[58.25, 174.75]	[0.00, 132.50]
2	609.82	121.98	*155.16*						
3	599.61	116.61	109.11						
4	598.80	115.83	112.96						
5	608.14	117.15	*132.81*						
6	600.86	115.83	113.56						
7	596.58	116.68	95.51						
8	601.58	116.01	115.31						
9	600.77	115.71	77.74						
10	598.23	118.25	*147.39*						

The italicized values indicate that the z-coordinate of the packing center of gravity is out of the domain constraints that are used by Baldi et al. (2012)

same instances as those used by Baldi et al. (2012). We also considered the balance constraints applicable to both the left–right and the front–rear directions when the given allowable gaps were 5 % for both.

In this comparison, our objective function was to maximize the total volume and consider the total profit as an additional attribute. As stated in Sect. 2, our study and Baldi et al. (2012) study differ in their approaches to balancing the container. Moreover, Baldi et al. (2012) study gives value to the total profit of the packed items $\text{profit}_i = 200 + \text{volume}_i$; our study focuses on maximizing the total volume of the packed items. Both of these two objective functions are common in the real world; however, efforts to maximize the total profit could yield different results than those that maximize the total volume of packed items such that high or low values of either do not necessarily correspond to high or low values of the other goal. In Baldi et al. (2012) study, for example, packing Items 1 and 2 into the container can increase the profit by $400+\text{volume}_1+\text{volume}_2$, while packing Item 3 can increase the profit by $200+\text{volume}_3$. If $400+\text{volume}_1+\text{volume}_2 > 200+\text{volume}_3$, one would select Items 1 and 2 to increase the profit. However, in our model, the two packing solutions can increase the objective value by $\text{volume}_1+\text{volume}_2$ and volume_3, respectively such that when $400+\text{volume}_1+\text{volume}_2 > 200+\text{volume}_3$ and $\text{volume}_1+\text{volume}_2 < \text{volume}_3$, the optimal solution differs from that of Baldi et al. (2012). It means that our HGA is specific for maximizing the total volume and may not show the best profit as sought by Baldi et al. (2012). These distinctions characterize an indirect comparison, but the side-by-side analysis is still useful to evaluate the performance of our algorithm in terms of profit, although maximizing the total profit was not our primary goal.

In Table 12, results under the column "UB_{1D} (Baldi et al. 2012)" present the upper bounds obtained from Baldi et al. (2012) by dealing with this problem as an 1D packing problem. Column 4 refers to the total profit obtained from Baldi et al. (2012), Column

5 refers to the total profit obtained from our HGA, and Column 6 refers to the gap between the total profit obtained from our algorithm and that obtained from Baldi et al. (2012). A negative gap indicates that the result obtained from our algorithm is worse than that obtained from the Baldi et al. (2012) algorithm, and a positive gap indicates that our algorithm outperforms that of Baldi et al. (2012). The sum of the gaps represented at the end of Table 12 shows the competitiveness of our algorithm. The results presented in Table 12 reveal that the performance of our algorithm competes with that of the 3BKP-H in Baldi et al. (2012). There are two instances in which both our method and that of Baldi et al. (2012) obtain the same value of the total profit.

In fact, the 3DCPP is a special case of the 3D knapsack problem in which an item's profit and volume are equal. Typically, the method designed for the 3D knapsack problem does not guarantee that volume utilization will be as good as that obtained through the method specifically designed for the 3DCPP. This situation accounts for the publication of many studies addressing the 3DCPP, while there are many studies in the field of 3D knapsack problem. Thus, although Baldi et al. (2012) provided a method to address the 3D knapsack problem with balance constraints, our proposed algorithm for the 3DCPP with balance constraints offers a new contribution to the field. Specifically, the performance of our method overcomes that of the HTS algorithm of Liu et al. (2011), which is the most recent study of the balance constraints in a 3DCPP. In addition, our algorithm can efficiently and realistically help shippers make decisions concerning volume utilization and weight balance.

5.2 Computational results without balance constraints

We experimented with the benchmark set of 1,500 BR instances, as developed by Bischoff and Ratcliff (1995), to compare the performance of our packing heuristic with that of Moura and Oliveira (2005) GRMod heuristic. We also compared the results of our algorithm with those of several other efficient algorithms. The findings, which contain 15 classes (BR1–BR15), are available from the OR-library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/). Each class consists of 100 different data sets (100 test instances). The total volume of the boxes in each test is approximately the same as the container volume. A container with a length, width, and height of 587, 233, and 220 cm, respectively, is used. The data sets (box sets) vary from weakly heterogeneous (e.g., BR1 contains 3 box types) to strongly heterogeneous (e.g.,BR15 has 100 box types). To facilitate a comparison with previous algorithms, computational results were obtained with a six-way rotation, ten test instances for each BR class, and the omission of balance constraints.

5.2.1 Comparisons with Moura and Oliveira's (2005) GRMod heuristic

As stated in Sect. 4.1, our block-building approach, which is used as a packing strategy, is a modification of the wall-building approach used in Moura and Oliveira (2005) GRMod heuristic. This comparison is used to demonstrate how block building is superior to wall building in terms of volume utilization. The results obtained by the GRMod heuristic and our packing strategy are presented in Table 13. These results

Table 12 Comparisons of our HGA and Baldi et al. (2012) 3BKP-H

Instance	Number of boxes	UB_{1D} (Baldi et al. 2012)	3BKP-H (Baldi et al. 2012)	*HGA*	*Gap*
ep3d-20-C-C-50	20	1026348	633672	591704.00	−41968.00
ep3d-20-C-C-90	20	1834340	916241	854048.00	−62193.00
ep3d-20-C-R-50	20	2188245	1492413	1274940.90	−217472.10
ep3d-20-C-R-90	20	3925057	2497691	1729741.00	−767950.00
ep3d-20-D-C-50	20	395916	315964	316492.00	528.00
ep3d-20-D-C-90	20	718692	526480	567732.00	41252.00
ep3d-20-D-R-50	20	240621	214227	216052.23	1825.23
ep3d-20-D-R-90	20	414188	335123	339420.50	4297.50
ep3d-20-F-C-50	20	2395087	1900250	2018436.40	118186.40
ep3d-20-F-C-90	20	4304020	3118132	3709254.57	591122.57
ep3d-20-F-R-50	20	2252037	1800399	1881965.03	81566.03
ep3d-20-F-R-90	20	4099982	3232580	3444781.67	212201.67
ep3d-20-L-C-50	20	1064487	891283	964231.20	72948.20
ep3d-20-L-C-90	20	1894489	1619190	1692745.20	73555.20
ep3d-20-L-R-50	20	718561	626381	608364.70	−18016.30
ep3d-20-L-R-90	20	1282710	1056699	1076047.60	19348.60
ep3d-20-U-C-50	20	4495440	3127252	3250520.00	123268.00
ep3d-20-U-C-90	20	8067424	6074000	6881984.40	807984.40
ep3d-20-U-R-50	20	4413077	3509748	3679962.37	170214.37
ep3d-20-U-R-90	20	8041072	6921250	7213531.77	292281.77
ep3d-40-C-C-50	40	2065540	1265664	1265664.00	0.00
ep3d-40-C-C-90	40	3652448	2828160	2632848.00	−195312.00
ep3d-40-C-R-50	40	4102972	3008658	2746076.30	−262581.70
ep3d-40-C-R-90	40	7335602	5972946	5177559.90	−795386.10
ep3d-40-D-C-50	40	788124	630996	687377.20	56381.20
ep3d-40-D-C-90	40	1423896	1126300	1292284.53	165984.53
ep3d-40-D-R-50	40	399894	349470	371077.80	21607.80
ep3d-40-D-R-90	40	728248	639819	663016.63	23197.63
ep3d-40-F-C-50	40	4816926	3590244	4163547.87	573303.87
ep3d-40-F-C-90	40	8664122	6435962	7566095.83	1130133.83
ep3d-40-F-R-50	40	4518343	3644680	3899976.53	255296.53
ep3d-40-F-R-90	40	8199224	7336067	7221413.00	−114654.00
ep3d-40-L-C-50	40	2127316	1760700	1934640.70	173940.70
ep3d-40-L-C-90	40	3819412	3032364	3456678.00	424314.00
ep3d-40-L-R-50	40	1784686	1609648	1583041.73	−26606.27
ep3d-40-L-R-90	40	3224295	2699629	2677196.50	−22432.50
ep3d-40-U-C-50	40	8988536	7355808	7751747.73	395939.73
ep3d-40-U-C-90	40	16241380	14065676	14581546.93	515870.93
ep3d-40-U-R-50	40	8666294	7766238	7928905.27	162667.27

Table 12 continued

Instance	Number of boxes	UB_{1D} (Baldi et al. 2012)	3BKP-H (Baldi et al. 2012)	*HGA*	*Gap*
ep3d-40-U-R-90	40	15531980	13077284	12502484.03	−574799.97
ep3d-60-C-C-50	60	3063219	1504980	1504980.00	0.00
ep3d-60-C-C-90	60	5517671	4475024	4130424.00	−344600.00
ep3d-60-C-R-50	60	6493464	5695120	4106756.53	−1588363.47
ep3d-60-C-R-90	60	11675188	10209801	9052634.67	−1157166.33
ep3d-60-D-C-50	60	1200408	1057032	1128754.67	71722.67
ep3d-60-D-C-90	60	2143544	1843584	1963866.13	120282.13
ep3d-60-D-R-50	60	538113	502275	468775.13	−33499.87
ep3d-60-D-R-90	60	966582	861655	810752.07	−50902.93
ep3d-60-F-C-50	60	7193700	6257697	6621909.93	364212.93
ep3d-60-F-C-90	60	12913715	10412682	11448241.80	1035559.80
ep3d-60-F-R-50	60	6780100	6146420	6032823.17	−113596.83
ep3d-60-F-R-90	60	12301636	10866326	10639468.17	−226857.83
ep3d-60-L-C-50	60	3211612	2656622	2939264.00	282642.00
ep3d-60-L-C-90	60	5736894	4832080	5279951.47	447871.47
ep3d-60-L-R-50	60	2391507	2158105	2047280.67	−110824.33
ep3d-60-L-R-90	60	4304649	3872594	3444593.80	−428000.20
ep3d-60-U-C-50	60	13508800	12033592	12039029.47	5437.47
ep3d-60-U-C-90	60	24342664	19787768	21917618.00	2129850.00
ep3d-60-U-R-50	60	12097660	10857656	9819246.30	−1038409.70
ep3d-60-U-R-90	60	21893096	19304585	18481367.07	−823217.93
Sum of the gaps					1951985

indicate that our packing strategy is superior to the GRMod heuristic for data sets from BR1 to BR10, except data sets BR2, BR3, and BR4. Our packing strategy is also superior to the GRMod heuristic for strongly heterogeneous data sets BR8, BR9, and BR10 because of the nature of the data sets, the GRMod heuristic, and our packing strategy.

5.2.2 Comparisons with other efficient algorithms

Table 14 presents a comparison between the results of our algorithm and those of several algorithms in terms of the average volume utilization for the BR benchmarks. For the BR benchmarks, the best result was obtained with the VNS algorithm of Parreno et al. (2010), with an average volume utilization of 94.53 % on BR1–BR7. From the experimental results for the benchmark data sets, although the volume utilization of our algorithm was slightly lower than that of the other algorithms, the proposed algorithm achieved a steady rate of volume utilization. In particular, for BR1–BR7, our algorithm achieved an average volume utilization of 92.61 %, which was lower than the values

Table 13 Comparisons in terms of the average volume utilization regarding the BR benchmarks

	GRMod heuristic		Our packing strategy	
Problem	Volume utilization (%)	Computation time (s)	Volume utilization (%)	Computation time (s)
BR1	86.67	<1	88.90	0.08
BR2	87.77	<1	87.71	0.11
BR3	87.19	<1	86.48	0.15
BR4	86.21	<1	85.60	0.19
BR5	85.54	<1	85.76	0.21
BR6	84.20	<1	85.67	0.28
BR7	82.37	<1	84.89	0.35
BR8	79.24	<1	84.52	0.55
BR9	76.81	<1	84.15	0.64
BR10	73.76	<1	84.54	0.81
Average (BR1–BR10)	82.97		85.82	

obtained by Parreno et al. (2008, 2010), but higher than those of the other algorithms. In terms of computation time, a direct comparison cannot be made due to the use of different computers. However, our algorithm and those of other studies have the same tendency with respect to computation time in which the computation time is lower for the weakly heterogeneous problems BR1–BR7 and higher for the strongly heterogeneous problems BR8–BR15. In general, the average computation time in Bortfeldt and Gehring (2001) is 11.7 s per problem. In Eley (2002), the computation time for data sets BR1–BR7 is limited at 600 s, which was reached in only 26 out of the 700 instances; in Bortfeldt et al. (2003), the computation time is 36–198 s for data sets BR1–BR7; in Lim et al. (2005), the computation time is not reported; in Moura and Oliveira (2005), the computation time is less than 200 s for data sets BR1–BR15; in Parreno et al. (2008), the computation time is 1.11–387 s for data sets BR1–BR15; in Parreno et al. (2010), the computation time is 2–800 s for data sets BR1–BR15; in Liu et al. (2011), the computation time is not reported precisely for data sets BR1–BR15 as these authors simply reported a computation time of 2–800 s in general for all data sets used in their paper; and the computation time is 5.3–873 s for data sets BR1–BR10 in our study.

6 Conclusions

This paper presented a new hybrid algorithm for handling balance constraints in the CPP while achieving superior volume utilization. The proposed algorithm is a hybrid approach, combining the DBLF strategy, a greedy heuristic, and a basic GA. The performance and the reliability of the proposed algorithm were evaluated with computational experiments conducted with the well-known BR test cases, so that the volume utilizations obtained by our algorithm could be compared with several other

Table 14 Comparisons in terms of the average volume utilization regarding the BR benchmarks

Problem	Bortfeldt and Gehring (2001)	Eley (2002)	Bortfeldt et al. (2003)	Lim et al. (2005)	Moura and Oliveira (2005)	Parreno et al. (2008)	Parreno et al. (2010)	Liu et al. (2011)	HGA
BR1	87.81	88.05	93.52	87.40	89.07	93.27	*94.93*	88.14	93.24
BR2	89.40	88.44	93.77	88.70	90.43	93.38	*95.19*	89.52	94.10
BR3	90.48	89.23	92.58	89.30	90.86	93.39	*94.99*	90.53	93.53
BR4	90.63	89.24	93.05	89.70	90.42	93.16	*94.71*	90.75	92.74
BR5	90.73	88.97	92.34	89.70	89.57	92.89	*94.33*	90.79	92.19
BR6	90.72	88.91	91.72	89.70	89.71	92.62	*94.04*	90.74	91.56
BR7	90.65	88.36	90.55	89.40	88.05	91.86	*93.53*	90.07	90.92
BR8	89.73	–	90.26	–	86.13	91.02	*92.78*	88.89	89.30
BR9	89.06	–	89.50	–	85.08	90.46	*92.19*	88.51	89.28
BR10	88.40	–	88.73	–	84.21	89.87	*91.92*	87.76	88.86
Average (BR1–BR7)	90.06	88.74	92.50	89.13	89.73	*92.94*	*94.53*	90.08	*92.61*

The significance of the italicized values is to make the best results and the competitive results more visual algorithms. In addition, we compared the performances of Moura and Oliveira (2005) GRMod heuristic and our packing strategy, which extended the GRMod heuristic. Furthermore, to analyze the performance of the algorithm with balance constraints, we conducted tests with the benchmark instances considered by Baldi et al. (2012) and with two of our own data sets, one based on Bischoff and Ratcliff (1995) idea of creating box objects and the other type based on Liu et al. (2011) information concerning RW instances.

A direct analysis of volume utilization as determined by comparison of the MIP model with balance constraints showed that the proposed algorithm yielded good volume utilization within the range of allowable weight balance, but also could inform shippers deciding if volume utilization or weight balance results would maximize profit. Based on the experimental results of the benchmark data sets, although the volume utilization of our algorithm was slightly lower than those of other algorithms, it achieved a steady and high rate of volume utilization, and the difference in volume utilization between the best and the worst solutions was not excessive. Moreover, our packing strategy outperformed Moura and Oliveira (2005) GRMod heuristic. Regarding the total profit of the selected items, our proposed algorithm competes with that of Baldi et al. (2012). In the future, we plan to improve the performance of the proposed algorithm to achieve greater volume utilization and to extend the current problem to cases of multiple containers.

Acknowledgments The authors are grateful for the useful comments from an associate editor and two anonymous referees. The authors are also grateful to Professor Perboli for sending us his benchmarking data, and Dr. Feng for his comments on the paper. This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (**NRF**) funded by the Ministry of Education, Science and Technology (2010-0025714).

References

Crainic T, Perboli G, Tadei R (2012) Recent advances in multi-dimensional packing problems. In: Volosencu C (ed) New technologies—trends, innovations and research. InTech, Chennai, pp 91–111. ISBN: 978-953-51-0480-3

Bortfeldt A, Gehring H (2001) A hybrid genetic algorithm for the container loading problem. Eur J Oper Res 131(1):143–161

Martello S, Pisinger D, Vigo D (2000) The three-dimensional bin packing problem. Oper Res 48(2):256–267

Bischoff EE, Ratcliff MSW (1995) Issue in the development of approaches to container loading. OMEGA Int J Manag Sci 23(4):377–390

Pisinger D (2002) Heuristics for the container loading problem. Eur J Oper Res 141:382–392

Bischoff EE, Janetz F, Ratcliff MSW (1995) Loading pallets with non-identical items. Eur J Oper Res 84(3):681–692

Eley M (2002) Solving container loading problems by block arrangement. Eur J Oper Res 141(2):393–409

Gehring H, Bortfeldt A (1997) A genetic algorithm for solving the container loading problem. Int Trans Oper Res 4(5–6):401–418

Bortfeldt A, Gehring H (1998) A tabu search algorithm for weakly heterogeneous container loading problems. OR Spect 20(4):237–250

Crainic TG, Perboli G, Tadei R (2008) Extreme-point-based heuristics for the three-dimensional bin packing problem. Informs J Comput 20:368–384

Zhang D, Wei L, Chen Q, Chen H (2007) A combinatorial heuristic algorithm for the three-dimensional packing problem. J Softw 18(9):2083–2089

Bortfeldt A, Gerhring H, Mack D (2003) A parallel tabu search algorithm for solving the container loading problem. Parallel Comput 29(5):641–662

Crainic T, Perboli G, Tadei R (2009) TS^{2}PACK: a two-level tabu search for the three-dimensional bin packing problem. Eur J Oper Res 195:744–760

Parreno F, Alvarez-Valdes R, Oliveira JF, Tamarit JM (2008) A maximal-space algorithm for the container loading problem. Informs J Comput 20(3):412–422

Parreno F, Alvarez-Valdes R, Oliveira JF, Tamarit JM (2010) Neighborhood structures for the container loading problem: a VNS implementation. J Heuristics 16(1):1–22

Perboli G, Crainic TG, Tadei R (2011) An efficient metaheuristic for multi-dimensional multi-container packing. In: IEEE Conference on Automation Science and Engineering (CASE). doi:10.1109/CASE.2011.6042476:563-568

Lim A, Rodrigues B, Yang Y (2005) 3-D container packing heuristics. Appl Intell 22(2):125–134

Moura A, Oliveira JF (2005) A GRASP approach to the container-loading problem. IEEE Intell Syst 20(4):50–57

Karabulut K, Inceoglu MM (2005) A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method. Comput Sci 3261(2005):441–450

Davies AP, Bischoff EE (1999) Weight distribution considerations in container loading. Eur J Oper Res 114(3):509–527

Liu J, Yue Y, Dong Z, Maple C, Keech M (2011) A novel hybrid tabu search approach to container loading. Comput Oper Res 38:797–807

Baldi MM, Perboli G, Tadei R (2012) The three-dimensional knapsack problem with balancing constraints. Appl Math Comput 218(19):9802–9818

Gen M, Lin L (2012) Multiobjective genetic algorithm for scheduling problems in manufacturing systems. Ind Eng Manag Syst 11(4):310–330

Parwananta H, Maghfiroh MFN, Yu VF (2013) Two-phase genetic algorithm for solving the paired single row facility layout problem. Ind Eng Manag Syst 12(3):181–189