Flex Serv Manuf J (2015) 27:451–477
DOI 10.1007/s10696-013-9181-8

Hybrid genetic algorithms for the three-dimensional multiple container packing problem

Xuehao Feng · Ilkyeong Moon · Jeongho Shin

Published online: 25 August 2013

Abstract The three-dimensional multiple container packing problem (3DMCPP) is used to determine non-overlapping packing of a set of finite three-dimensional rectangular items into the minimum number of identical containers. The decision framework consists of two main activities: item assignment and packing. This paper presents new hybrid genetic algorithms (HGAs) that address current limitations related to the 3DMCPP and enable use of relatively few containers. Rotation constraints are also addressed. An HGA is developed for small problems, and another HGA is created for large problems. Both of the HGAs combine the largest left space first item assignment strategy, a basic genetic algorithm, and the deepest bottom left with fill strategy. Experiments were conducted to demonstrate the performances of the HGAs with two different types of data sets. The results show that the proposed HGAs yield solutions, in a reasonable amount of time, in which relatively few containers are needed.

Keywords Multiple container packing · Hybrid genetic algorithm · DBLF strategy

X. Feng
Engineering Research Institute, Seoul National University, Seoul 151-744, Korea
e-mail: fengxuehao@pusan.ac.kr

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

J. Shin
Dongkuk Steel Group, Seoul 100-210, Korea
e-mail: jeongho.shin@dongkuk.com

1 Introduction

The ocean transportation sector has grown significantly, becoming an important factor that influences the performances of companies. To the best of our knowledge, in modern ocean transportation, no choice is more commonly employed than that used to pack convenient, highly standardized, and relatively inexpensive containers because, clearly, use of fewer containers can lead to lower container leasing/ purchasing and transportation costs. Therefore, managers must determine the best way to pack all the items into a minimum number of containers. The three-dimensional multiple container packing problem (3DMCPP) involves an orthogonal packing process of items into a minimum number of identical containers without overlapping. Wäscher et al. (2007) proposed a typology and introduced new categorization criteria wherein the 3DMCPP is described as an input minimization problem. According to the intermediate problem types of the input minimization problem, the 3DMCPP is referred to as single stock-size cutting stock problem or single bin-size bin packing problem when the items are weakly or strongly heterogeneous. Chen et al. (1995) proposed a mixed integer programming model for a generalization of the problem with different-sized containers. A two-dimensional multiple container packing problem (2DMCPP) has been widely studied (e.g., Martello and Vigo 1998). However, it is difficult to extend the approaches developed for the 2DMCPP to address issues raised in the 3DMCPP (Crainic et al. 2009). Consequently, some new algorithms have been proposed for the 3DMCPP.

Most previous studies focused on the 3DMCPP with relatively large items that cannot be rotated. Martello et al. (2000) showed that the 3DMCPP is an NP-hard problem, and suggested that the Martello, Pisinger, and Vigo (MPV) algorithm is effective when a few large items need to be packed. In their algorithm, the main branching tree is used to assign items to the containers, and a branch-and-bound algorithm is used to decide the positions of each item. In addition, for a given set of packed items, corner points are defined where items can only be packed. The authors determined feasible corner points efficiently by repeatedly solving two-dimensional problems. In that paper, different classes of instances were generated and tested in the experiments, which have been used as benchmarks in many studies. Faroe et al. (2003) presented a heuristic algorithm based on a guided local search (GLS) algorithm. At the beginning of the algorithm, a greedy heuristic is used to find an initial number of containers to use. Then, the GLS algorithm improves the solution iteratively by using a GLS to find a solution for feasibly packing of items. In these experiments, the instances were generated by using the method of Martello et al. (2000) and the maximum number of items was increased to 200. Lodi et al. (2002) discussed a heuristic called *height first-area second* for the 3DMCPP, in which a tabu search approach and the concept of layers were used. The best two possible solutions are selected based on the heuristic. In the first solution, several clusters are obtained by partitioning the items by non-increasing height and the layers are combined into finite containers. In the second solution, the items are resorted by non-increasing area of their base and new layers are generated. den Boef et al. (2005) showed that the method of filling a single container used by Martello et al. (2000) may miss some feasible packing positions. Crainic et al. (2008)

proposed the extreme point first fit decreasing (EP-FFD) heuristic and the extreme point best fit decreasing (EP-BFD) heuristic for the 3DMCPP. In these heuristics, the concept of extreme points for packing items is used and the items are sorted according to an externally specified rule. Crainic et al. (2009) proposed a two-level tabu search that separates the search for the optimal number of containers based on the way the items are packed in each container. In the first-level heuristic, the items are assigned to containers. In the second-level heuristic, the positions of items in each container are obtained. The authors claimed that their approach is flexible and can be modified to take into account additional constraints in filling a single container without changing the first-level heuristic. Parreño et al. (2010) developed a hybrid greedy randomized adaptive search procedure/variable neighborhood descent (GRASP/VND) algorithm for solving the 3DMCPP; it consists of constructive and improvement phases. In the constructive phase, solutions are obtained through an iterative process, each of which is improved in the second phase. All of these above authors assumed that the items have fixed orientations such that the items cannot be rotated. Mack and Bortfeldt (2012) presented a heuristic algorithm that consists of five modules. The algorithm can address the 3DMCPP in which items may be rotated by 90° and new types of instances were generated in that paper.

In the above papers (Martello et al. 2000; Lodi et al. 2002; Faroe et al. 2003; Crainic et al. 2009; Parreño et al. 2010; Mack and Bortfeldt 2012), the same eight classes of instances were used in the experiments, and the items are heterogeneous and relatively large. The average volume of one item was approximately 15 % of the container's volume and the maximum number of items was no greater than 200. The maximum number of packed items in each container was usually smaller than 7, and hereafter, we refer to this situation as a *small problem*. In small problems with a set of assigned items, an optimal packing solution for each container is relatively easy to find, especially when rotations are not allowed. Because the items are big, different assignments of items can significantly influence the packing solutions. Consequently, appropriate item assignments can play a more important role than the way the items are packed into each container.

On the other hand, Thapatsuwan et al. (2011) proposed three approaches to the 3DMCPP and tested them for heterogeneous and relatively small items. They used the wall building approach to pack the items and considered a six-way rotation. The experimental results showed that the artificial immune system (AIS) can obtain a better solution than the genetic algorithm (GA) and the particle swarm optimization (PSO). In the experiments by Thapatsuwan et al. (2011), the average volume of one item was approximately 0.7 % of the container's volume and the maximum number of items was 5000. Hereafter, we refer to this situation as a *large problem*. These large problems may involve more than 100 items packed into each container, for which it is difficult to find an optimal packing solution, especially when a six-way rotation is allowed. Because the average volume of each item is small and many items are to be packed, the assignments of items cannot significantly influence the packing solutions, and placement of each item in the container plays a more important role. Based on this difference between small and large problems, we developed two HGAs. Figure 1 shows an example in which large and small items

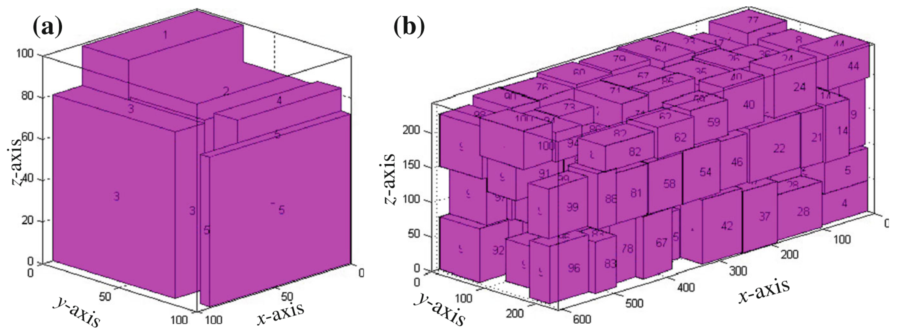


Fig. 1 Example of *large* and *small* items packed into separate containers: **a** *large* items in a container and **b** *small* items in a container

are packed into two separate containers. Figure 1a shows a container with packed items generated from Class 1, as described by Martello et al. (2000). Figure 1b shows a container with packed items generated as per Thapatsuwan et al. (2011). One can see that few packing solutions apply to small problems but many packing solutions can be obtained for large problems.

The solution to the 3DMCPP requires an orthogonal packing of items in a single container that influences the performances of the algorithms. Many researchers have attempted to solve the three-dimensional single container packing problem (3DSCPP), which involves packing a set of items into one container to maximize volume utilization. Ngoi et al. (1994) proposed a spatial matrix approach to record the information of the container and the packed items. Bischoff and Ratcliff (1995) used the concept of a layer to pack the items; this approach is widely used in other research on container packing problems. Bortfeldt and Gehring (2001) solved this problem by using a two-stage method. In the first stage, towers were built from suitable items based on a greedy heuristic. In the second stage, the towers were placed into a container based on a GA. Karabulut and Inceoğlu (2004) proposed a hybrid GA by combining a GA with the deepest bottom left with fill (DBLF) method. The authors also compared the hybrid GA with some heuristics. Parreño et al. (2008) developed a GRASP algorithm, where constructive and improvement phases are used. In the constructive phase, a maximal-space heuristic method generates solutions that will be improved in the second phase through the consideration of two different objective functions. Fanslau and Bortfeldt (2010) developed a tree search algorithm based on the block building approach. A special fashion called partition-controlled trees search is used that enables a sufficient search width and a suitable degree of foresight. Kang et al. (2012) developed a hybrid GA by adopting an extended DBLF method that can address two-way and six-way rotations of items. They also showed that the hybrid GA can achieve high volume utilization. Zhu and Lim (2012) proposed a new block building approach in which one block is placed at a time until no more boxes can be loaded in the container. In the same paper, a new heuristic is developed to select the correct block to place into a specific residual space. Gonçalves and Resende (2012) presented a multi-population biased random-key genetic algorithm (BRKGA). The BRKGA evolves the order in which the item types are loaded and the corresponding type of layers involved in the placement procedure.

The framework of the 3DMCPP contains two activities. In the first activity, the items are assigned to the containers. In the second activity, the assigned items are packed into the containers. GA has proven to be efficient in the 3DSCPP. However, there is not adequate research to show whether GA can be an efficient approach for the 3DMCPP. In this paper, we propose two hybrid genetic algorithms (HGAs) that are specifically designed for large and small problems wherein six-way rotation is allowed for all items. We refer to our HGA for small problems as HGA-S, and to our HGA for large problems as HGA-L. We compare the HGA-S and HGA-L with existing algorithms, and the experimental results demonstrate that our algorithms are competitive. A comparison of our paper to other relevant papers is summarized in Table 1.

This paper is organized as follows. The next section describes the 3DMCPP, including a problem definition and a description of the constraints. In Sect. 3, the proposed algorithms are presented, and Sect. 4 shows the experiments. We offer conclusions in Sect. 5.

2 Problem definition and filling a single container

2.1 Problem definition

We are given a set of n three-dimensional rectangular items, and item i is of length l_i, width w_i, and height h_i, $i = 1, 2, \ldots, n$. There are an unlimited number of identical cubic containers with length L, width W, and height H. We assume that each item can be packed into an empty container in at least one orientation variant. The objective is to orthogonally pack all of the items into the minimum number of containers without overlapping. These conditions mean that:

1. no item can penetrate the boundary surfaces of the container;
2. each item is parallel to the boundary surfaces of the container;
3. no pair of packed items overlaps.

In the Cartesian coordinate system, Point O(0, 0, 0) is the left-bottom-back corner of a container. In this paper, we assume that the length, width, and height of the

Table 1 Summary of relevant studies

	Small problems	Large problems	Rotation
Martello et al. (2000)	$\surd$	$\times$	$\times$
Lodi et al. (2002)	$\surd$	$\times$	$\times$
Faroe et al. (2003)	$\surd$	$\times$	$\times$
Crainic et al. (2009)	$\surd$	$\times$	$\times$
Parreño et al. (2010)	$\surd$	$\times$	$\times$
Thapatsuwan et al. (2011)	$\times$	$\surd$	$\surd$
Mack and Bortfeldt (2012)	$\surd$	$\times$	$\surd$
This study	$\surd$	$\surd$	$\surd$

$\surd$ covered, $\times$ not covered

container are oriented with the x, y, and z axes. Figure 2 illustrates one container and item i in the Cartesian coordinate system, $i = 1, 2, \ldots, n$.

Rotation of items is an important operational factor in both of the research and the real world. It can be proven that there are, at most, six different rotation types for each item packed in a container. The assumption of a two-way rotation is also common in the real world. The studies on 3DSCPP illustrate that the GAs that address two-way rotation can be easily extended to the problem wherein six-way rotation is allowed. In this paper, we assume that six-way rotation is allowed for all of the items. We denote l_i', w_i', and h_i' as the length, width, and height of item i after it is rotated, $i = 1, 2, \ldots, n$. The relationships between the sizes and rotation types of items are shown as follows.

Case 0: No rotation is performed, i.e., $l_i' = l_i$, $w_i' = w_i$, and $h_i' = h_i$

Case 1: The length is replaced by the width of the item and vice versa; the height remains fixed, i.e., $l_i' = w_i$, $w_i' = l_i$, and $h_i' = h_i$

Case 2: The height is replaced by the width of the item and vice versa; the length remains fixed, i.e., $h_i' = w_i$, $w_i' = h_i$, and $l_i' = l_i$

Case 3: The length is replaced by the height, the width is replaced by the length, and the height is replaced by the width of the item, i.e., $l_i' = h_i$, $w_i' = l_i$, and $h_i' = w_i$

Case 4: The height is replaced by the length of the item and vice versa, the width remains fixed, i.e., $h_i' = l_i$, $l_i' = h_i$, and $w_i' = w_i$

Case 5: The length is replaced by the width, the width is replaced by the height, and the height is replaced by the length of the item i.e., $l_i' = w_i$, $w_i' = h_i$, and $h_i' = l_i$

Figure 3 shows the six-way rotations of an item. In the problems wherein two-way rotation is allowed, only Cases 0 and 1 will be considered.

As noted in Martello et al. (2000), it is easy to obtain a lower bound for the 3DMCPP. V_c is the volume of the container and $V_c = W \times L \times H$ and the continuous lower bound, LB_C, is as follows:

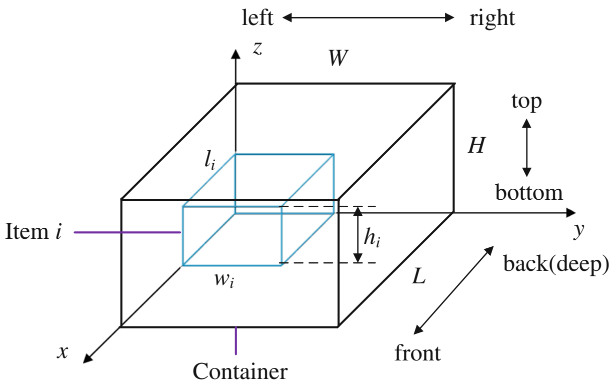


Fig. 2 *Container* and *boxes* in the Cartesian coordinate system

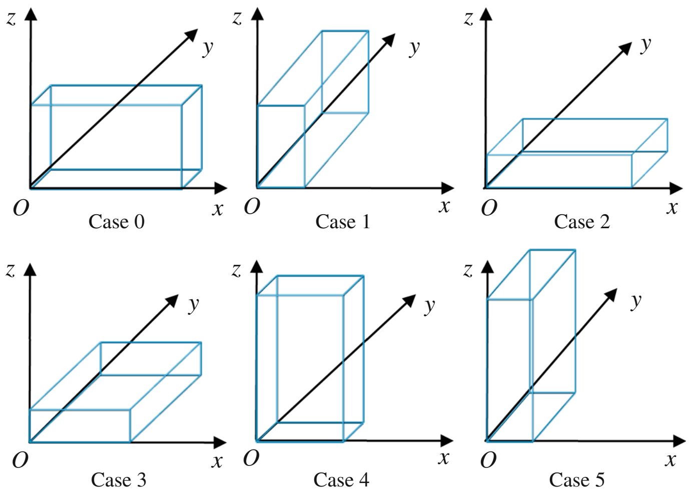


Fig. 3 An item in the case of a six-way rotation

$$LB_C = \left\lceil \frac{\sum_{i=1}^{n} (w_i l_i h_i)}{V_c} \right\rceil \tag{1}$$

Fekete and Schepers (1998), Martello et al. (2000), and Boschetti (2004) discussed new lower bounds for the 3DMCPP with no rotation. Boschetti (2004) also proposed new lower bounds (LB_B) for the problems wherein six-way rotation is allowed and the time complexity is $O(n^4)$. All of these lower bounds were tested on small problems.

2.2 Filling a single container

For a sequence of rotated items assigned to the containers, how is a packing strategy designed to obtain a solution for the 3DMCPP? Karabulut and Inceoğlu (2004) discussed the DBLF strategy to decide the positions of the items that proved efficient in filling a single container. In this paper, the DBLF strategy is used to pack the assigned items into containers sequentially. The DBLF strategy used in this paper consists of two steps. The first step is to generate a series of potential positions (PP) for the items that have not been packed. A PP list is created for each container. For a rotation type, the sizes of item i are obtained as l_i', w_i', and h_i'. When item i is placed with its left-bottom-back corner in position (x_i, y_i, z_i) in one container, at most four new PP are generated and added in List *PP*. We generate two PP by projecting points ($x_i + l_i'$, y_i, z_i) and (x_i, $y_i + w_i'$, z_i) on the items between the bottom of the container and item i. If there is more than one item under item i, we select the position on the item that is the nearest to item i. If there is no item under item i, we obtain two positions: ($x_i + l_i'$, y_i, 0) and (x_i, $y_i + w_i'$, 0). The third potential position is obtained as (x_i, y_i, $z_i + h_i'$). Let maximum packed length (*MPL*) be the maximum ($x_k + l_k'$) of all of the packed items. The fourth position is generated as (*MPL*, 0, 0), if ($x_i + l_i'$) is greater than the *MPL* before it is packed. In particular, if the container is empty, List *PP* only contains one position, i.e., O(0, 0, 0). In this case, item i is placed at position O(0, 0, 0), which generates three positions with coordinates (l_i', 0, 0), (0, w_i', 0), and (0, 0, h_i'). In addition, if $x_i + l_i' = L, y_i + w_i' = W$, or $z_i + h_i' = H$, then corresponding positions will not be generated. Figure 4 shows an example of the PP in one container when three items have been packed.

Algorithm 1 updates List *PP* when an item (item *i*) is packed.

```
Algorithm 1
begin
  Input IP: List of items that have been packed into the container;
  Input PP: List of the potential positions corresponding to the items in List IP;
  Input i: Item to be placed at position ($x_i, y_i, z_i$);
  Input MPL: The maximum ($x_k + l_k'$), $k \in IP$;
    Set $MPH\_x \leftarrow 0$ and $MPH\_y \leftarrow 0$;
    if $z_i$=0 then
    add positions ($x_i + l_i'$, $y_i$, 0) and ($x_i$, $y_i + w_i'$, 0) into List PP;
    else
      for $j = |IP|$ to 1 do
    if $x_j < x_i + l_i'$, $y_j < y_i + w_i'$, $x_j + l_j' > x_i + l_i'$, and $y_j + w_j' > y_i$ then
      if $z_j + h_j' <= z_i$ and $z_j + h_j' > MPH\_x$ then
       Set $MPH\_x \leftarrow z_j + h_j'$;
      end if
    end if
        if $x_j < x_i + l_i'$, $y_j < y_i + w_i'$, $x_j + l_j' > x_i$, and $y_j + w_j' > y_i + w_i'$ then
      if $z_j + h_j' <= z_i$ and $z_j + h_j' > MPH\_y$ then
       Set $MPH\_y \leftarrow z_j + h_j'$;
      end if
    end if
    if $MPH\_x = z_i$ and $MPH\_y = z_i$ then
      break
   end if
  end for
  add positions ($x_i + l_i'$, $y_i$, $MPH\_x$) and ($x_i$, $y_i + w_i'$, $MPH\_y$) into List PP;
  add position ($x_i$, $y_i$, $z_i + h'$) into List PP;
   If $x_i + l_i' > MPL$ then
     remove position (MPL, 0, 0) from List PP;
     Set $MPL \leftarrow x_i + l_i'$;
     add position (MPL, 0, 0) into List PP;
   end if
  remove position ($x_i$, $y_i$, $z_i$) from List PP;
end.
```

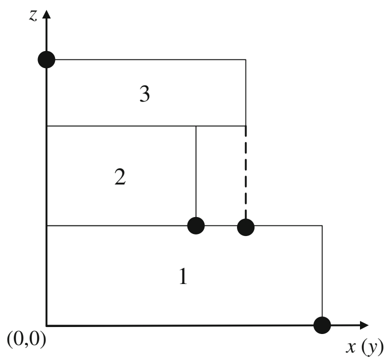


Fig. 4 Example of potential positions

The second step is to place a given item in one of the PP in List *PP*. The positions in List *PP* are sorted in the deepest-bottom-left order. In this case, the positions are sorted in ascending order according to their x coordinates. Positions with the same x coordinate are sorted in ascending order according to their z coordinates. Positions with the same x and y coordinates are sorted in ascending order according to their y coordinates. We attempt to place an item and rotation in the first position of List *PP*. An item can be accommodated in a position if, after placing its left-bottom-back corner on the position, (1) it does not overlap any other item that has been packed into the container, and (2) it does not penetrate the boundary surfaces of the container. If the item cannot be placed at the left-bottom-back position, we will attempt to place the item on the next position in the List *PP*. When an item is placed at one position, the position will be removed from List *PP*. This process terminates when either a feasible position is found or we have attempted to place the item in all possible positions.

As Fig. 5 shows, the first item is placed at O(0, 0, 0) when the container is empty. Then, three new PP are generated and sorted, i.e., P1, P2, and P3. Next, we attempt to place the second item at P1. Because the second item does not overlap with any packed item or penetrate the boundary surfaces of the container, we place it at P1 and remove P1 from List *PP*. Three new positions are generated, i.e., P4, P5, and P6, all of which are added into the list. All of the positions in the list are sorted, i.e., P4, P5, P2, P3, and P6 (in this order). We attempt to place the third item at P4. In this case, the item penetrates the boundary surface of the container so, we place the item at P5. The DBLF strategy is outlined by the following pseudo-code:

```
DBLF strategy:
begin
  Input one item (e.g., item $i$) and the rotation type;
  Input List PP;
      update $w_i'$, $h_i'$, and $l_i'$ based on the rotation type of item $i$;
      sort List PP according to the deepest-bottom-left order;
      for $j = 1$ to $|PP|$ do
          if item $i$ can be placed at position $j$ then
              place item $i$ in position $j$;
              update List PP with Algorithm 1;
              break;
          end if
      end for
  end for
end.
```

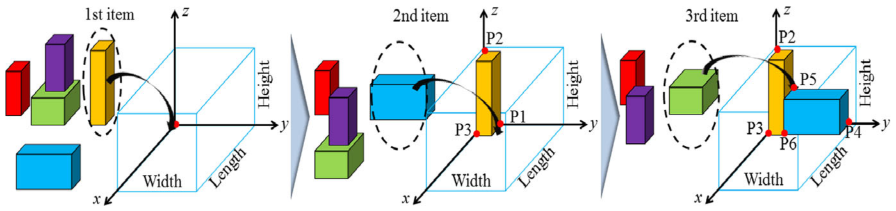


Fig. 5 An example of packing items into a container based on the DBLF strategy

3 Hybrid genetic algorithms for 3DMCPP

In this section, we discuss our HGAs for small and large problems. Our HGAs combine the *largest left space first* (LLSF) assignment strategy, the DBLF strategy, and a basic GA. In the HGA-S and HGA-L, the LLSF strategy is used to assign the items to the containers with the largest left space. With this strategy, the volumes of the items assigned to the containers are balanced to help reduce the number of containers needed. The GA is used to decide the sequence of items. Then, the positions of the items in each container are determined based on the DBLF strategy.

3.1 The LLSF strategy

We generate a list of containers to use, i.e., List *C*. The original number of containers in List *C* is *LB* (lower bound on the number of containers). The left space of the container is determined by the difference between V_c and the total volume of the items that have been packed into the container. *I* is the sequence of the items for which the basic idea of the LLSF strategy can be shown as follows:

```
LLSF strategy:
 begin
  Input $I$: List of items wherein $|I|=n$;
    initialize the left spaces of the used containers in List $C$;
    for $i$=1 to $n$ do
        select the container with the largest left space;
        if the volume of item $i$ is smaller than or equal to the left space then
            assign item $i$ to that container;
            update the left space of that container;
        else add a new container into List $C$;
             assign item $i$ to the new container;
             update the left space of the new container;
        end if
    end for
 end.
```

One can see that at the beginning, all of the containers have the same left space, which is equal to V_c. Then, we assign the first item to the container and update the left space of the container. This step is repeated until all of the items are assigned. The idea of this LLSF strategy has also been used in many other fields of operations research. In the field of sequencing and scheduling, for example, the problem of minimizing the make span in multi-machine models (parallel systems) involves assigning a set of given jobs to identical machines (e.g., see Baker and Trietsch 2009). Through use of the LLSF strategy, the algorithms can limit attention to a search for an optimal sequence of jobs (items). In the next two subsections, we present how to combine this strategy with a GA for small and large problems.

3.2 HGA for small problems (HGA-S)

Figure 6 illustrates the proposed procedure of the HGA-S.

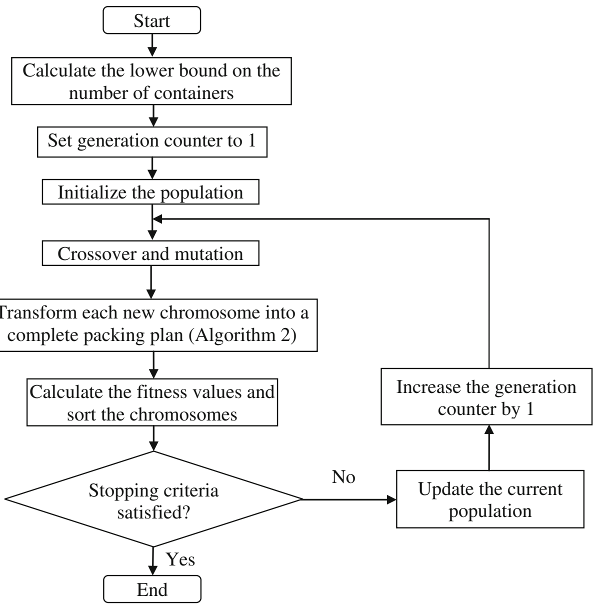
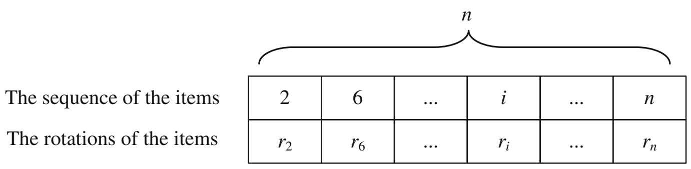

Start
Calculate the lower bound on the number of containers
Set generation counter to 1
Initialize the population
Crossover and mutation
Transform each new chromosome into a complete packing plan (Algorithm 2)
Calculate the fitness values and sort the chromosomes
Stopping criteria satisfied?
No
Update the current population
Increase the generation counter by 1
Yes
End

Fig. 6 Flowchart of the HGA-S



	$\overbrace{\qquad}^{n}$					
The sequence of the items	2	6	...	i	...	n
The rotations of the items	r_2	r_6	...	r_i	...	r_n

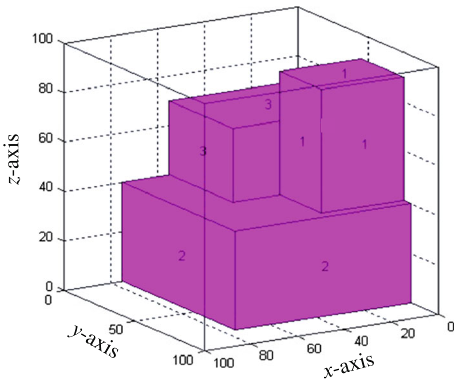
Fig. 7 The chromosome used in the HGA-S

3.2.1 Chromosome representation

In the HGA-S, a chromosome with two rows and n columns is coded. The first row presents the sequence of the items, and the second row presents the rotations of the items. Each gene of the chromosome consists of the properties of an item, i.e., the index and rotation type. Let r_i be the rotation type of item i and r_i can be 0, 1, …, 5. For r_i, the rotation of item i can be found in Fig. 3, $i = 1, 2, \ldots, n$. Figure 7 shows an example. There are n genes in the chromosome. At first, we assign Item 2 to the container based on the LLSF strategy and pack it with the rotation type of r_2 based on the DBLF strategy. Second, we assign Item 6 and pack it with the rotation type of r_6. We repeat this procedure until all items in the chromosome are assigned and packed.

This representation of a chromosome does not give a complete solution and it only shows the sequence and rotation types of the items. In a packing solution the number of containers is equal to NC such that a complete solution consists of NC matrixes with five rows and m_j columns, where m_j is the number of items packed

Fig. 8 Illustration of a solution of one container



The index of the container	#1		
The sequence of the items	2	3	1
The rotations of the items	5	2	0
The *x*-coordinates of the items	0	0	0
The *y*-coordinates of the items	0	0	45
The *z*-coordinates of the items	0	40	40

into container $j, j = 1, 2, \ldots, NC$. A column contains a set of properties of a packed item. This set consists of five data elements that include the index, rotation, *x* coordinate, *y* coordinate, and *z* coordinate of an item. In Fig. 8, the matrix with five rows and three columns completely describes a feasible packing solution of the first container. The solution shows that Items 2, 3, and 1 are packed into the container sequentially. As discussed above, we can obtain a complete packing solution by using the LLSF and DBLF strategies with a given sequence of items and their rotations. Consequently, for a chromosome a complete packing solution can be obtained through the HGA-S. As mentioned in Wäscher et al. (2007), the 3DMCPP is an input minimization problem, and the following formula is used to calculate the fitness value of chromosome $k, k = 1, 2, \ldots, K$, wherein K is the population size.

$$Fitness_k = 1/the\ number\ of\ containers\ in\ chromosome\ k \tag{2}$$

3.2.2 Chromosome decoding

We use the chromosome information to assign and pack the items sequentially into the containers based on the LLSF and the DBLF strategies. Because the left space is the absolute value of the left volume, the following case is possible: An item cannot be packed into a container if it can be packed into another container that has a smaller left space. As a consequence, if we fail to pack the item into the container with the largest left space, we need to attempt to pack it into other containers. If the item cannot be packed into any existing container, an additional container is added into which the item is packed. After packing the item, we repeat this step until all of the items are packed. For any specific chromosome, Algorithm 2 assigns and packs the items based on the LLSF and DBLF strategies.

Algorithm 2

```
begin
  Input I: List of items wherein $|I|=n$, including the rotation types of the items;
    initialize the left spaces and List PP of containers in List C;
    for $i=1$ to $n$ do
      for $j=1$ to $|C|$ do
        select the container with the largest left space that has not been attempted for item $i$,
        e.g., container $k$;
          if item $i$ can be packed into container $k$ based on the DBLF strategy then
            pack item $i$ into container $k$ based on the DBLF strategy;
            update List PP of container $k$ with Algorithm 1;
            update the left space of container $k$;
            break;
          end if
      end for
      if item $i$ cannot be packed into any container in List C then
        add a new empty container into List C;
        initialize the left space and List PP of the new container;
        pack item $i$ into the new container based on the DBLF strategy;
        update List PP of the new container with Algorithm 1;
        update the left space of the new container;
      end if
    end for
end.
```

3.2.3 Population initialization

To guarantee an initial population with a certain quality, the items with large volumes should be assigned and packed into a container first. Based on this idea, four special chromosomes can be created by sorting the items in descending order according to the values of volumes, lengths, widths, and heights of the items, respectively. In the three chromosomes based on length, width, and height, the rotation types of all items are set to 0. Then, we randomly generate the remaining chromosomes including the sequences and rotation types of all items, until the number of chromosomes is equal to the population size.

3.2.4 Crossover and mutation

From Eq. (2), one can see that the best chromosome has the highest fitness value, which will lead to the fewest number of containers used. Roulette wheel selection (Goldberg 1989) is used to select chromosomes for crossover. The fitness values of the chromosomes are used as the selection probabilities. The probability of selecting a chromosome with a higher fitness value is also relatively higher. Because this approach has been widely used in GAs (e.g., Thapatsuwan et al. 2011; Al-Hinai and ElMekkawy 2011), details will not be discussed in this paper.

For two chromosomes selected via roulette wheel selection, $P1$ and $P2$, both crossover and mutation are used to generate new chromosomes, $O1$ and $O2$. We use the well-known two-point crossover with probability P_c. During the crossover, two cutting sites i and j are randomly selected, $i < j$. The genes $P1(i) \ldots P1(j)$ are copied into $O1(i) \ldots O1(j)$. Then, $P2$ is swept circularly from the $(j + 1)$th gene onward to complete $O1$ with the missing genes and $O1$ is filled circularly from the $(j + 1)$th gene. By copying the genes $P2(i) \ldots P2(j)$ into $O2$, we can obtain a complete $O2$ with the same approach. To this point in the process, the rotation type of each item had not been changed. A two-step mutation is used wherein two randomly selected sites are swapped with probability P_{m1}, and each rotation parameter of the genes of $O1$ and $O2$ is randomly changed with probability P_{m2}. If the crossover and mutation are not implemented, $P1$ and $P2$ are copied onto $O1$ and $O2$, respectively. If the population size is K, then we can obtain other K chromosomes after crossover and mutation. The chromosomes for the next generation will be selected from these $2K$ chromosomes.

3.2.5 Updating the solution and checking the stopping criterion

An elitist selection strategy is used to select chromosomes that will be used in the next generation. Therefore, the best solution is always in the population. We sort the $2K$ chromosomes in descending order according to the fitness values and keep the first K chromosomes for the next generation. The best chromosome is the one with the highest fitness value. We compare the best chromosome in the current population with the overall best (incumbent) chromosome. If the best chromosome in the current population has a greater fitness value, we update the overall best chromosome.

To prevent the GA from reaching a local optimum, a triggered re-initialization scheme is utilized. If the population converges into a small search space, some of the worst chromosomes in the population will be reinitialized. In each generation, we use a diversity index (*div*), determined by the fitness values, to evaluate the degree of convergence of the current population. We used f_{best} as the fitness values of the best chromosome and f_{ave} as the average fitness value of all of the chromosomes in one population. The index *div* can be calculated as follows:

$$div = \frac{f_{best} - f_{ave}}{f_{best}} \tag{3}$$

From Eq. (3), one can see that the population tends to converge if *div* is close to zero. Consequently, we used a triggered generator and set a certain threshold (θ). If *div* is smaller than θ, some of the worst chromosomes (with the lowest fitness values) will be replaced by the same number of new randomly generated chromosomes. The appropriate values of θ and the number of replaced chromosomes can be obtained through pilot experiments.

The stopping criterion is invoked when we obtain a solution wherein the number of used containers is equal to the lower bound, or the number of generations reaches a limit. If the stopping criterion is not satisfied, we create a new generation.

3.3 The HGA for large problems (HGA-L)

In large problems, the number of items is relatively large and each item is much smaller than the items in small problems. We generate a sequence of items by sorting the items in descending order according to their volumes. The assignment of the items exactly follows the LLSF strategy discussed in Sect. 3.1. Based on the

LLSF strategy, the volumes of assigned items of the containers are balanced. Due to this assignment of the items, we obtain the chromosomes for each container. Then, the GA is run separately for each container. The procedure of the HGA-L is shown in Fig. 9. We start our algorithm at the lower bound on the number of containers. If we cannot obtain a feasible solution under the current number of containers, we will increase the number by 1 and repeat the assignment and packing methods. As far as we know, there is no study that can guarantee the exact lower bound for the 3DMCPP. Therefore, we start to search in the number of containers that have been proved to be a lower bound that may be smaller than the exact one. Consequently, it is not easy to say whether the starting point is feasible or not. However, the effectiveness of our HGA-L can be improved with an improved lower bound that may be developed in the future studies.

For each container with a set of assigned items created by LLSF, the GA is run to obtain the packing solution. If all of the assigned items can be packed into the container, we run GA for the next container. If one item cannot be packed into the container under the DBLF strategy, we will attempt to pack the left items in the set into the container. If there is any item that cannot be packed into the container in the maximum allowed generation number, we increase the number of containers and generate a new assignment based on the LLSF strategy. As Fig. 10 shows, the classic procedure for a GA is used, including population initialization, crossover and mutation, updating the population, and checking the stopping criterion. The GA uses

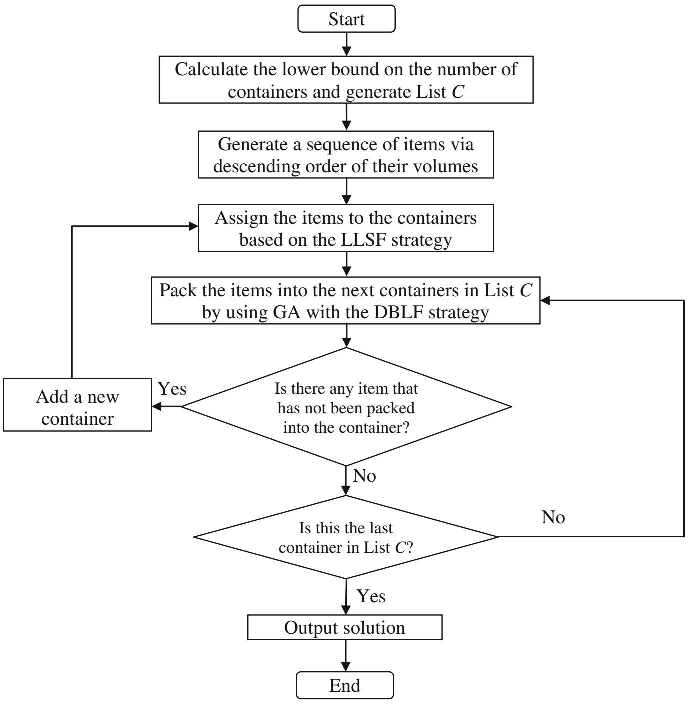

Start
Calculate the lower bound on the number of containers and generate List C
Generate a sequence of items via descending order of their volumes
Assign the items to the containers based on the LLSF strategy
Pack the items into the next containers in List C by using GA with the DBLF strategy
Is there any item that has not been packed into the container?
Yes
Add a new container
No
Is this the last container in List C?
No
Yes
Output solution
End

Fig. 9 Flowchart of the HGA-L

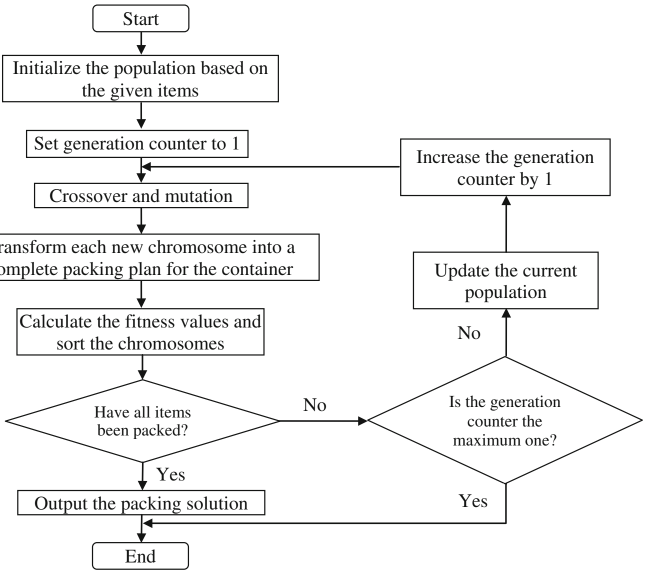

Start
Initialize the population based on the given items
Set generation counter to 1
Crossover and mutation
Transform each new chromosome into a complete packing plan for the container
Calculate the fitness values and sort the chromosomes
Have all items been packed?
No
Yes
Output the packing solution
End
Is the generation counter the maximum one?
No
Yes
Update the current population
Increase the generation counter by 1

Fig. 10 Flowchart of the GA used in the HGA-L

the same mechanisms of population initialization, crossover, mutation, and updating the solution as used in the HGA-S, which is described in Sect. 3.2.

3.3.1 Chromosome representation

In the HGA-L, the chromosomes have the same structure as the ones used in the HGA-S. The difference is that in the HGA-L, a group of chromosomes for each container is generated and each chromosome only contains the items assigned to that container. Figure 11 shows an example of one chromosome generated for container j, where n_j is the number of items assigned to container j. At first, we pack Item 2 with rotation type of r_2 into container j based on the DBLF strategy. Second, we pack Item 7 into container j with rotation type of r_7. We repeat this procedure until all of the n_j items are packed. If one item cannot be packed, we will attempt to pack further items in that chromosome based on the DBLF strategy.

Equation (4) is used to calculate the fitness value of chromosome k. Because we attempt to pack all of the assigned items into the corresponding containers, the objective of the GA is to maximize the fitness value. This fitness definition relates to each of the single container packing problems that are embedded in the whole large problem. The HGA-L also has the same structure of complete solutions as the HGA-S.

$$Fitness_k = the\ number\ of\ packed\ items\ in\ chromosome\ k \tag{4}$$

3.3.2 Checking the stopping criterion

In the GA for one container, the stopping criterion is invoked when we pack all of the assigned items into the container or the number of generations reaches a limit. If

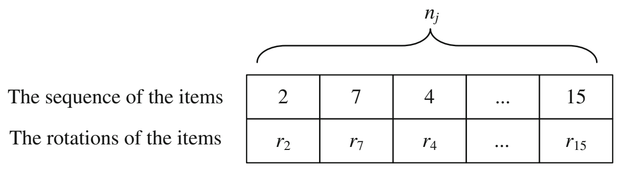


Fig. 11 The chromosome used in the HGA-L

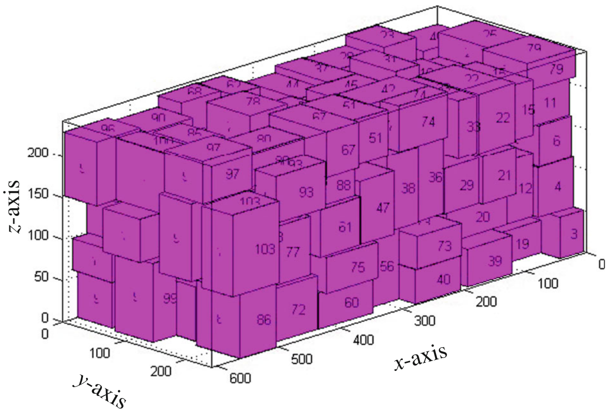


Fig. 12 An example of the packing solution of one container

the stopping criterion is not satisfied, we start a new generation. If all of the assigned items are packed, the best chromosome provides the packing solution for that container and we start to pack the items into the next container. Otherwise, the algorithm terminates and we attempt the process using more containers.

3.4 Comparison of the HGA-S and HGA-L

Three differences characterize the uniqueness of the HGA-S and HGA-L. First, in the HGA-S, the sequence of item assignments is determined by the GA. Different assignments of items can be obtained from different chromosomes. Consequently, the HGA-S can generate more assignment scenarios, while the assignment of items in the HGA-L is fixed. Intuition suggests that the HGA-S can obtain more solutions than the HGA-L. Therefore, the HGA-S is more efficient than the HGA-L with few assigned items, but it may be less efficient for large problems, i.e. the container has many assigned items, especially when rotations are allowed. In large problems, the assignment of items may not influence the solution as significantly as the manner of the positioning of each item, which is the focus of the HGA-L.

Second, in terms of computational time, the HGA-L is more sensitive to the lower bounds on the number of containers than the HGA-S. In the HGA-S, if one item cannot be packed into any existing container, a new container will be added into which the item is packed, and the algorithm continues to assign and pack the remaining items. Therefore, one need not spend a long time obtaining a better lower bound. We use LB_C in the HGA-S. In the HGA-L, however, if one item cannot be packed into the container, the algorithm will generate a new assignment scenario based on a new number of containers and run the GA from the first container. Therefore, a good lower bound can save time as one avoids testing meaningless number of containers; therefore, time invested to obtain a good lower bound repays the researcher. The same advantage also exists in the MPV algorithm (see Boschetti 2004). Therefore, we use LB_B as the lower bound on the number of containers in the HGA-L.

Third, in the HGA-S, the GA is used to decide the sequence and rotations of all of the items. Then, the items are assigned to the containers based on the LLSF strategy, and the positions are determined based on the DBLF strategy simultaneously. Consequently, the fitness value of each chromosome is determined by the number of containers used. In the HGA-L, the items are assigned to the containers based on the LLSF strategy without checking the feasibility of the assignment. Then, the GA and DBLF strategy are used for each container separately to determine the sequence, rotations, and positions of the items. Therefore, the fitness value of each chromosome is the number of packed items.

4 Computational experiments

As far as we know, two main types of tests have been used in related studies. We designed two experiments and tested our algorithms based on the same approaches used in other studies. The proposed algorithms were coded in JAVA, and the experiments were run on a Pentium 4 3.4 GHz processor with 1 GB RAM. We denote the instances used in Martello et al. (2000) as small problems and the instances used in Thapatsuwan et al. (2011) as large problems.

4.1 Experiment on small problems

In the small-problem experiment, we tested our HGA-S on the eight different instances that were generated by the approach of Martello et al. (2000). For Classes 1–5, the container size is $W = L = H = 100$, and five types of items need to be packed as follows:

Type 1: w_i follows a uniform distribution in [1, 1/(2W)], l_i in [2/(3L), L], h_i in [2/(3H), H];
Type 2: w_i follows a uniform distribution in [2/(3W), W], l_i in [1, 1/(2L)], h_i in [2/(3H), H];
Type 3: w_i follows a uniform distribution in [2/(3W), W], l_i in [2/(3L), L], h_i in [1, 1/(2H)];
Type 4: w_i follows a uniform distribution in [1/(2W), W], l_i in [1/(2L), L], h_i in [1/(2H), H];
Type 5: w_i follows a uniform distribution in [1, 1/(2W)], l_i in [1, 1/(2L)], h_i in [1, 1/(2H)]

For Class k ($k = 1$, 2, 3, 4, and 5), each item is of type k with a probability of 60 % and has one of the other four types with a probability of 10 % each. Classes 6, 7, and 8 are generated as follows.

Class 6: $W = L = H = 10$, and w_i, l_i, and h_i follow a uniform distribution in [1, 10];
Class 7: $W = L = H = 40$, and w_i, l_i, and h_i follow a uniform distribution in [1, 35];
Class 8: $W = L = H = 100$, and w_i, l_i, and h_i follow a uniform distribution in [1, 100]

Lodi et al. (2002), Faroe et al. (2003), Crainic et al. (2009), and Parreño et al. (2010) tested their algorithms (TS, GLS, TS^{2}PACK, and GRASP, respectively) on the instances generated by the above approach. Faroe et al. (2003) and Crainic et al. (2009) also tested the MPV algorithm (Martello et al. 2000) when n is equal to 50, 100, 150, and 200. Crainic et al. (2008) tested the EP-FFD and EP-BFD heuristics on the same instances. We only compared our HGA-S with the TS, GLS, TS^{2}PACK, and GRASP because the MPV, EP-FFD, and EP-BFD algorithms were outperformed by the other four algorithms. TS (Lodi et al. 2002) was run on a Digital Alpha 533 MHz CPU. GLS (Faroe et al. 2003) was run on a Digital 500au workstation with a 500 MHz 21164 CPU. In each of these two experiments, a time limit of 1000 s was given for the test. TS^{2}PACK (Crainic et al. 2009) was run on a Pentium4 2,000 MHz CPU, and the computational time was transformed to those on a Digital 500 workstation. GRASP (Parreño et al. 2010) was run on a Pentium Mobile at 1,500 MHz with 512 MB of RAM and the computational time limit was set to 150 s. In the above experiments, 10 instances were generated for each class and size of a problem. The average number of containers is used to evaluate the algorithms and the maximum number of items is 200. We compared the HGA-S with the TS, GLS, TS^{2}PACK, and GRASP algorithms with no rotation. Ten instances were generated for each class and size of a problem based on different random seeds. For each instance, 10 independent runs were executed. Because different CPUs were used, we set the maximum allowable generation number to 200 so that the time of the experiments was limited to 100 s. The maximum number of generations was set based on the time limit. Fine-tuning of the HGA-S showed the appropriate settings of robust parameters for all of the classes and items in the experiment. The values are shown in Table 2. P_{m2} was set to 0 in the experiments without considering rotations of items.

The performance comparisons between our HGA-S and the four algorithms from other studies are shown in Table 3. The first three columns report the classes of the instances, the sizes of the containers, and the number of items respectively. The fourth and fifth columns report the average solution value (number of containers)

Table 2 Parameter settings for the HGA-S

Population size	P_c	P_{m1}	P_{m2}	θ
100	0.8	0.2	0.02	0.1

Table 3 Results for the instances using various algorithms with 50, 100, 150, and 200 items

Class	Container	n	HGA-S (6)	HGA-S (0)	GRASP	TS^{2}PACK	GLS	TS
1	100 × 100 × 100	50	11.7	**13.1**	13.4	13.4	13.4	13.4
		100	24.2	**26.2**	26.6	26.7	26.7	26.6
		150	33.5	**36.3**	36.4	37.0	37.0	36.7
		200	47.4	**50.7**	50.9	51.1	51.2	51.2
2	100 × 100 × 100	50	11.8	**13.2**	13.8	–	–	13.8
		100	23.6	26.1	**25.7**	–	–	**25.7**
		150	32.3	**36.4**	36.9	**–**	**–**	37.2
		200	44.7	50.0	**49.4**	–	–	50.1
3	100 × 100 × 100	50	11.9	**13.2**	13.3	–	–	13.3
		100	23.9	**25.7**	26.0	–	–	26.0
		150	35.3	**37.5**	37.6	–	–	37.7
		200	44.4	50.2	**50.0**	–	–	50.5
4	100 × 100 × 100	50	28.8	**29.1**	29.4	29.4	29.4	29.4
		100	58.2	**58.7**	59.0	58.9	59.0	59.0
		150	86.5	**86.8**	**86.8**	**86.8**	**86.8**	**86.8**
		200	118.3	**118.7**	118.8	118.8	119.0	118.8
5	100 × 100 × 100	50	7.3	**8.0**	8.3	8.3	8.3	8.4
		100	13.9	**15.0**	**15.0**	15.2	15.1	**15.0**
		150	18.8	**19.9**	20.1	20.1	20.2	20.4
		200	25.9	**26.8**	27.1	27.4	27.2	27.6
6	10 × 10 × 10	50	9.1	**9.5**	9.8	9.8	9.8	9.9
		100	18.3	19.1	**19.0**	19.1	19.1	19.1
		150	28.2	29.4	**29.2**	**29.2**	29.4	29.4
		200	37.0	37.6	**37.4**	37.7	37.7	37.7
7	40 × 40 × 40	50	6.3	**7.0**	7.4	7.4	7.4	7.5
		100	11.2	**12.2**	12.5	12.3	12.3	12.5
		150	15.0	**15.6**	16.0	15.8	15.8	16.1
		200	22.5	**23.5**	**23.5**	**23.5**	**23.5**	23.9
8	100 × 100 × 100	50	8.5	**9.1**	9.2	9.2	9.2	9.3
		100	18.3	**18.6**	18.9	18.8	18.9	18.9
		150	22.5	**23.6**	24.1	23.9	23.9	24.1
		200	29.4	30.1	**29.8**	30.0	29.9	30.3
Average Classes 1, 4–8			29.20	**30.19**	30.36	30.41	30.43	30.50
Total Classes 1, 4–8			700.8	**724.6**	728.6	729.8	730.2	732.0
Average all Classes			29.0	**30.53**	30.67	–	–	30.82
Total all Classes			928.7	**976.9**	981.3	–	–	986.3

The best values without rotation consideration appear in bold obtained from our HGA-S over 10 instances, with six-way rotation and no rotation. The values in the next four columns are taken from Parreño et al. (2010), Crainic et al. (2009), Faroe et al. (2003), and Lodi et al. (2002). The symbol "–" means that the experiments were not conducted in those papers. From Table 3, one can see that our algorithm obtains better solutions than the four algorithms on most problems when no rotation is allowed. In the cases in which the GRASP algorithm obtains better solutions than our HGA-S, the maximum gap is around 1 %. When a six-way rotation is allowed, our HGA-S obtains better solutions on all of the problems. In addition, the HGA-S takes into account the six-way rotation without significantly increasing the computation time. This phenomenon is consistent with the conclusion in the 3DSCPP research that the time spent on implementing a six-way rotation is negligible (e.g., see Kang et al. 2012). Consequently, the HGA-S is more competitive on small problems when rotations are allowed. We show a solution for the problem of Class 1 with 50 items in the "Appendix", in which "con." presents the container index and "*r*" presents the rotation type.

In the experiments, we set the population size and maximum allowable generation number as 100 and 200, respectively. To show the effectiveness, we tested our HGA-S with a larger population size of 200 and a smaller maximum allowable generation number of 100. We used the instances of Classes 1 and 7 as examples. For each problem size, we generated 10 different instances. Table 4 shows the average number of used containers obtained from the HGA-S of 10 instances under these two pairs of parameters. "100–200" presents the pairs in which population size is 100 and maximum allowable generation number is 200. "200–100" presents the pairs in which population size is 200 and maximum allowable generation number is 100. The third and fourth columns show the number of used containers obtained from the HGA-S. As shown in Table 4, the HGA-S with "100–200" obtained better or equivalent solutions compared with that with "200–100" in all of the instances.

4.2 Experiment on large problems

In this experiment, we tested our HGA-L on large problems. We generated instances with the same approach used in Thapatsuwan et al. (2011). A standard container that

Table 4 Comparison of different pairs of population size and generation number for small problems

	n	100–200	200–100
Class 1	50	11.7	12.0
	100	24.2	24.4
	150	33.5	33.5
	200	47.4	47.8
Class 7	50	6.3	6.4
	100	11.2	11.2
	150	15.0	15.2
	200	22.5	22.5

is 20 ft long, 8 ft wide, and 8 ft high was considered in all of the instances. The length, width, and height of the items follow uniform distributions in [70, 100 cm], [50, 80 cm], and [30, 60 cm], respectively. All of the instances are available on the website http://scm.snu.ac.kr/research/Data_Large.rar

Pilot experiments showed that, for these cases, the parameters used in the HGA-S also performed well in the HGA-L, perhaps because the average number of items assigned to each container is approximately 100. It means that the chromosomes in the HGA-L have similar numbers of genes as the HGA-S. Therefore, we set the population size, P_c, θ, and P_{m1}, to the values in Table 2. Because a six-way rotation is allowed in this experiment, P_{m2} was set to 0.05 based on the pilot experiment. The maximum generation number was 200 for each container. Thapatsuwan et al. (2011) used wasted space to evaluate the algorithms. From the mixed integer programming model proposed by Chen et al. (1995), one can observe that, when the containers are identical, the objective of minimizing the number of containers is the same as minimizing the wasted space. Consequently, we also used wasted space to evaluate our solutions and allowed six-way rotation for all of the items. Fifteen instances were generated for each problem size. For each instance, five independent runs were executed. The results are shown in Table 5. The first column reports the number of items. The second column reports the average solution values (wasted space)

Table 5 Comparison of the HGA-L and AIS

n	Average waste space (m^3)	Computational time (min)
100	12.46	1.01
250	48.06	2.18
500	60.09	6.77
750	108.86	9.03
1,000	118.95	13.46
1,250	167.19	17.22
1,500	181.43	20.71
1,750	212.05	25.90
2,000	241.61	29.10
2,250	264.30	35.67
2,500	300.73	39.14
2,750	311.89	49.91
3,000	360.36	52.14
3,250	370.41	59.88
3,500	411.37	63.35
3,750	431.41	69.61
4,000	472.29	74.94
4,250	493.95	80.83
4,500	532.47	86.04
4,750	550.05	91.45
5,000	593.53	98.17

obtained from our HGA-S over the 15 instances for each problem size. The last column shows the computational time of the HGA-L.

As shown in Sect. 1, the proposed AIS can obtain better solutions than the GA and PSO for all of the problem sizes. Hence, we compared our results with the results of the AIS that have been presented in Thapatsuwan et al. (2011). The AIS was run on a personal computer with a Core2Quad 2.66 GHz CPU and 4 GB DDRIII RAM. The AIS and our HGA-L were tested on similar instances (generated by the same approach). From Table 5, one can see that the average results of our HGA-L are better than those of the AIS. Note that the volume of each container is approximately 36.25 m^3, and we can infer the number of saved containers in our algorithm compared to the number in the AIS. When $n = 500$, for example, we can infer that the solution of the HGA-L requires approximately five fewer containers than the solution of the AIS. This improvement might be achieved for the following two reasons. First, we combined the GA with the efficient heuristic, the LLSF strategy. As a consequence, the length of each chromosome in our GA is appropriate, i.e., around 100 genes in each chromosome, and the GA can focus on finding good packing solutions for the containers. Second, we used the DBLF strategy to fill each container, a strategy proven to be efficient in the 3DSCPP for strongly heterogeneous items (e.g., see Kang et al. 2012). For the instances in this experiment, the GA can easily find a packing solution for a container filled with approximately 100 items in 100 generations (in 1 min). Within 200 generations, the GA can only obtain packing solutions for the containers with fewer than 104 items. For example, when n is equal to 1,750, the HGA-L can obtain solutions requiring 17 containers in seven instances wherein the average number of items assigned to each container is 102.9. Figure 12 shows one container with 103 items when $n = 1{,}750$.

In each instance the time that our HGA-L used to find the solution is similar to the time the AIS used to find a solution reported in Thapatsuwan et al. (2011). However, our HGA-L spent around a quarter of the computation time on finding the packing solutions when the number of containers is between the lower bound and

Table 6 Comparison of different pairs of population size and generation number for large problems

n	Instance	100–200	200–100
2,500	1	43,100	61,800
	2	36,500	47,000
	3	52,300	64,200
2,750	1	94,900	132,900
	2	96,700	163,400
	3	112,200	167,100
3,000	1	70,600	110,000
	2	79,100	115,000
	3	90,600	125,800
3,250	1	165,200	257,000
	2	158,300	230,400
	3	162,600	200,400

the final number utilized. Therefore, if a better lower bound can be obtained in a meaningful time limit, the performance of our HGA-L can also be improved.

We compared different pairs of population size and maximum generation number for our HGA-L. We tested the instances with 2,500, 2,750, 3,000, and 3,250 items. For each problem size, three different instances were generated. The results showed that the HGA-L with "100–200" obtained the same number of used containers to the HGA-L with "200–100" for most of the instances. In the left few instances, the HGA-L with "100–200" obtained slightly better solutions. In addition, the HGA-L with "100–200" used fewer chromosomes and shorter computational time to obtain the best solution. Table 6 shows the number of used chromosomes to obtain the best solution under these two pairs of parameters.

5 Conclusions

We propose two HGAs that consist of the LLSF strategy, the DBLF strategy, and a basic GA. The LLSF strategy is used to assign the items to the containers such that the volumes are balanced. In the HGA-S, the sequence of the items is generated by a basic GA. Based on this GA-generated sequence, the items are assigned to the containers via the LLSF strategy and the positions are simultaneously determined through the DBLF strategy. In the HGA-L, the process for assigning items is separated from the process for deciding the positions of the items. We generated a fixed sequence of the items and assigned them to the containers based on the LLSF strategy. The basic GA is then used to find a packing solution for each container, including the sequence, rotations, and positions of the items. The comparison of the HGA-S and HGA-L shows the reasons these two algorithms perform well in small and large problems, respectively.

The experimental results demonstrate the performances of the HGA-S and HGA-L in small and large problems. Experiments on small problems were conducted with widely used test cases to compare our HGA-S with existing algorithms (e.g., Parreño et al. 2010; Crainic et al. 2009). The experimental results show that compared with the existing algorithms, our HGA-S is competitive even when rotations are not allowed, but we also show that HGA-S is more competitive in the cases in which item rotations, which can be easily implemented, are allowed. In the large problems wherein six-way rotation is allowed, we compare our HGA-L with the algorithms proposed by Thapatsuwan et al. (2011). The experimental results show that our HGA-L can bring significant improvement in the solutions, and the computational time is acceptable. Based on these improvements, the GA can be a competitive tool to resolve the 3DMCPP through adoption of heuristics. In addition, our HGA-L can be improved by using the advanced lower bounds for the 3DMCPP and the GA for the 3DSCPP.

Our HGAs could be improved in several ways. Our HGAs are more tailored to the strongly heterogeneous case where items are of many different sizes. New strategies or heuristics may improve the performance of the HGAs for the weakly heterogeneous case. Next, a new heuristic for assigning the items to the containers may increase the effectiveness of the HGA. In addition, designing a new framework of assignment and packing may be another fruitful direction for future research.

Acknowledgments The authors are grateful for the useful comments from an associate editor and two anonymous referees. 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).

Appendix

See Table 7.

Table 7 Solution of a small problem of Class 1

Item	Con.	*l*	*w*	*h*	*x*	*y*	*z*	*r*	Item	Con.	*l*	*w*	*h*	*x*	*y*	*z*	*r*
24	1	57	80	92	0	0	0	2	35	7	38	17	47	0	73	0	0
38	1	90	42	68	57	0	0	1	29	8	42	80	90	0	0	0	2
20	1	74	19	80	0	0	80	3	9	8	91	55	69	42	0	0	5
22	1	81	8	66	0	92	0	0	42	8	72	8	80	0	0	91	3
10	2	95	76	75	0	0	0	2	43	9	78	31	89	0	0	0	0
31	2	98	23	81	0	0	76	2	33	9	65	61	71	0	31	0	0
50	2	73	23	75	0	75	0	4	13	9	68	33	69	65	31	0	1
5	3	100	98	72	0	0	0	3	39	9	83	15	67	0	31	71	2
40	3	79	80	19	72	0	0	3	32	10	69	34	90	0	0	0	3
14	3	72	71	6	91	0	0	3	45	10	53	60	83	0	0	34	3
36	4	97	45	95	0	0	0	5	8	10	74	30	99	0	69	0	0
23	4	81	32	97	45	0	0	1	37	11	89	40	69	0	0	0	2
30	4	15	87	75	77	0	0	2	21	11	26	78	71	0	0	40	5
12	4	80	5	86	0	95	0	0	41	11	66	97	21	0	71	0	5
48	4	4	44	40	45	81	0	3	15	11	98	73	17	0	0	66	0
2	5	96	43	87	0	0	0	4	1	11	77	15	88	0	0	83	2
19	5	76	49	95	0	43	0	0	28	12	54	83	64	0	0	0	2
3	5	91	6	72	0	92	0	0	7	12	84	15	87	0	64	0	4
47	6	47	97	78	0	0	0	3	16	12	85	9	95	0	79	0	0
26	6	44	87	82	0	47	0	3	18	12	72	10	87	87	0	0	5
11	6	79	17	70	82	0	0	5	17	12	78	8	73	0	88	0	0
34	6	6	76	81	0	91	0	1	46	12	76	80	3	0	96	0	5
6	7	81	48	73	0	0	0	5	4	12	94	78	1	0	99	0	2
27	7	88	47	74	48	0	0	5	25	12	14	14	21	0	0	83	3
44	7	74	8	91	0	0	88	2	49	12	4	33	12	0	14	83	5

References

Al-Hinai N, ElMekkawy TY (2011) An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem. Flex Serv Manuf J 23:64–85

Baker KR, Trietsch D (2009) Principles of sequencing and scheduling. John Wiley & Sons, Hoboken, NJ

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

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

Boschetti MA (2004) New lower bounds for the three-dimensional finite bin packing problem. Discret Appl Math 140:241–258

Chen CS, Lee SM, Shen QS (1995) An analytical model for the container loading problem. Eur J Oper Res 80:68–76

Crainic TG, Perboli G, Tadei R (2008) Extreme point-based heuristics for three-dimensional bin packing. INFORMS J Comput 20(3):368–384

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

den Boef E, Korst J, Martello S, Pisinger D, Vigo D (2005) Erratum to "the three-dimensional bin packing problem": robot-packable and orthogonal variants of packing problems. Oper Res 53:735–736

Fanslau T, Bortfeldt A (2010) A tree search algorithm for solving the container loading problem. INFORMS J Comput 22(2):222–235

Faroe O, Pisinger D, Zachariasen M (2003) Guided local search for the three-dimensional bin packing problem. INFORMS J Comput 15(3):267–283

Fekete SP, Schepers J (1998) New classes of lower bounds for bin-packing problem. IPCO 98, Springer Lecture Notes in Computer Science 1412:257–270

Goldberg DE (1989) Genetic algorithms in search, optimisation and machine learning. Addison-Wesley, Reading, MA

Gonçalves JF, Resende MGC (2012) A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput Oper Res 39:179–190

Kang KD, Moon IK, Wang HF (2012) A hybrid genetic algorithm with a new packing strategy for the three-dimensional bin packing problem. Appl Math Comput 219:1287–1299

Karabulut K, Inceoğlu MM (2004) A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method. Proc ADVIS 2004:441–450

Lodi A, Martello S, Vigo D (2002) Heuristic algorithms for the three-dimensional bin packing problem. Eur J Oper Res 141:410–420

Mack D, Bortfeldt A (2012) A heuristic for solving large bin packing problems in two and three dimensions. CEJOR 20:337–354

Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. Manag Sci 44:388–399

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

Ngoi BKA, Tay ML, Chua ES (1994) Applying spatial representation techniques to the container packing problem. Int J Prod Res 32(1):111–123

Parreño 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

Parreño F, Alvarez-Valdes R, Oliveira JF, Tamarit JM (2010) A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. Ann Oper Res 179:203–220

Thapatsuwan P, Pongcharoen P, Hicks C, Chainate W (2011) Development of a stochastic optimisation tool for solving the multiple container packing problems. Int J Prod Econ 140:737–748

Wäscher G, Hauβner H, Schumann H (2007) An improved typology of cutting and packing problems. Eur J Oper Res 183:1109–1130

Zhu WB, Lim A (2012) A new iterative-doubling Greedy-Lookahead algorithm for the single container loading problem. Eur J Oper Res 222:408–417

Author Biographies

Xuehao Feng is a research fellow of Engineering Research Institute at Seoul National University. He received his B.S. and M.S. from Beijing Jiaotong University in China, and Ph.D. in Industrial Engineering from Pusan National University, Korea. His research interests lie in the areas of supply chain management, inventory management, and logistics.

Ilkyeong Moon is a Professor of Industrial Engineering at Seoul National University in Korea. He received his B.S. and M.S. in Industrial Engineering from Seoul National University, Korea, and Ph.D. in Operations Research from Columbia University. He is an area editor of International Journal of Industrial Engineering, and an associate editor of European Journal of Industrial Engineering and several international journals. He was a former Editor-in-Chief of Journal of the Korean Institute of Industrial Engineers a flagship journal of Korean Institute of Industrial Engineers.

Jeongho Shin is a production manager at Dongkuk Steel Group. He received his B.S. in Industrial Engineering from Ulsan University, Korea, and M.S. in Industrial Engineering from Pusan National University, Korea. He is involved in production scheduling of his company.