European Journal of Operational Research 214 (2011) 526–535

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A tree search method for the container loading problem with shipment priority

Jidong Ren *, Yajie Tian, Tetsuo Sawaragi

Graduate School of Engineering, Kyoto University, Sakyo-ku, Kyoto 606-8501, Japan

ARTICLE INFO

Article history:
Received 8 February 2010
Accepted 22 April 2011
Available online 30 April 2011

Keywords:
Packing
Container loading problem
Heuristic
Tree search
Shipment priority

ABSTRACT

This paper addresses a special kind of container loading problem with shipment priority. We present a tree search method, which is based on a greedy heuristic. In the greedy heuristic, blocks made up of identical items with the same orientation are selected for packing into a container. Five evaluation functions are proposed for block selection, and the different blocks selected by each evaluation function constitute the branches of the search tree. A method of space splitting and merging is also embedded in the algorithm to facilitate efficient use of the container space. In addition, the proposed algorithm covers an important constraint called shipment priority to solve practical problems. The validity of the proposed algorithm is examined by comparing the present results with those of published algorithms using the same data.

1. Introduction

The container loading problem (CLP) is important in production and distribution processes. The problem, in its basic form, can be described as follows: a set of three-dimensional, rectangular items is to be packed in a container in such a way that the volume of the packed items is maximized. Using the typology for cutting and packing problems proposed by Wäscher et al. (2007), the CLP can be classified as a three-dimensional rectangular single large object placement problem (3D SLOPP, weakly heterogeneous) or as a single knapsack problem (3D SKP, strongly heterogeneous).

Additional constraints can be added to the general CLP (i.e., CLP in basic form) to take into account some aspects of real-life problems. These constraints include load bearing strength, multi-drop, load stability, shipment priority, weight distribution and so on (Bischoff and Ratcliff, 1995). In this paper the container loading problem with the constraint of shipment priority (CLPSP) is addressed. The constraint of shipment priority can be described as follows: some items have higher priority over others; an item with low priority should not be packed into the container if it leads to high-priority items being left behind. In some practical situations, the shipment of some items may be more important than that of others, e.g., because of the delivery deadlines or the shelf life of the product concerned. In such situations, the prior loading of the important items must be taken into account.

The CLP is known to be an NP-hard problem, as it contains the well-described knapsack problem. In the last years, several types of methods have been proposed to solve the CLP, most of which are heuristics (George and Robinson, 1980; Bischoff and Ratcliff, 1995), or meta-heuristics such as genetic algorithm (GA) (Gehring and Bortfeldt, 1997; Bortfeldt and Gehring, 2001; Gehring and Bortfeldt, 2002; Techanitisawad and Tangwiwatwong, 2004), tabu search (TS) (Bortfeldt and Gehring, 1998), simulated annealing (SA) (Jin et al., 2004) and greedy randomized adaptive search procedure (GRASP) (Moura and Oliveira, 2005; Parreño et al., 2008). Tree search methods (Pisinger, 1998; Pisinger, 2002; Eley, 2002; Wang et al., 2008; Fanslau and Bortfeldt, 2010) have also been proposed for solving the CLP.

Most existing CLP methods are based on different heuristic packing approaches, such as the wall-building approach, stack-building approach, guillotine-cutting approach or block-building approach (i.e., the cuboid arrangement approach). Pisinger (2002) gave an excellent overview of these approaches. For instance, the wall-building approach fills the container with vertical layers (“walls”) that follow along the longest side of the container (George and Robinson, 1980; Pisinger, 1998; Bortfeldt and Gehring, 2001; Pisinger, 2002). The block-building approach fills the container with cuboid blocks that mostly contain only identical items with the same spatial orientation (Bortfeldt and Gehring, 1998; Eley, 2002; Mack et al., 2004; Parreño et al., 2008; Fanslau and Bortfeldt, 2010).

In recent literature, increased attention has been focused on the CLP. For instance, Gehring and Bortfeldt (1997), Davies and Bischoff (1999) and Eley (2002) took into account the weight distribution within a container. Bischoff (2006) considered the load bearing strength. Christensen and Rousøe (2009) addressed the container loading problem with multi-drop constraints.

However, no single study has addressed the CLPSP. Bischoff and Ratcliff (1995) suggested that the priorities for shipment can be viewed in a knapsack model simply as objective function coefficients which define or adjust the value ratings of the items. However, they did not propose any suitable algorithm based on this idea.

* Corresponding author. Tel.: +81 80 38486168.
E-mail address: renjd111@hotmail.com (J. Ren).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.04.025

The authors have had several years of experience in the development of practical container loading systems. Shipment priority has been considered in most of these practical systems. In this paper, a tree search algorithm based on a greedy heuristic is proposed for solving both the general CLP and the CLPSP. The greedy heuristic, as the basis of the proposed tree search method, is a kind of block-building approach. Different blocks (selected according to certain criteria that will be mentioned later) constitute the branches of the search tree. The tree search framework facilitates the allocation of the high- and low-priority items.

The paper is organized as follows: In Section 2 the problem statement is presented; Section 3 provides the algorithm to solve the general CLP and CLPSP; Section 4 is dedicated to the computational results of this algorithm; and Section 5 summarizes the paper.

2. Problem statement

The container is placed in a three-dimensional coordinate system with the origin in the lower back left corner. The length, width and height of the container are parallel to the x-, y- and z-axis, respectively (Fig. 1).

The items to be packed are categorized into types. Two items are the same type if they have the same dimensions, permissible orientations (as mentioned below) and shipment priority (if considered).

Similar to many existing CLP algorithms (e.g., Bortfeldt and Gehring, 2001; Eley, 2002; Moura and Oliveira, 2005), the following conditions should be respected when packing items into the container:

- Each item is placed completely within the container.
- Each item does not overlap with another item.
- Each item lies on the container floor or is completely supported by other items.
- Only orthogonal packing is considered, i.e., each items is placed parallel to the edges of the container.
- Items can be rotated, i.e., each item has six possible orientations, but some of these orientations may be not permitted.

The six possible orientations of item are shown in Fig. 2. An item with length l, width w and height h has six possible orientations whose dimensions on the x-, y- and z-axis are (l,w,h), (w,l,h), (l,h,w), (h,l,w), (h,w,l) and (w,h,l), respectively.

The CLP is to pack a subset of items into the container such that the volume of the packed items is maximized, i.e., the volume utilization of the container is maximized.

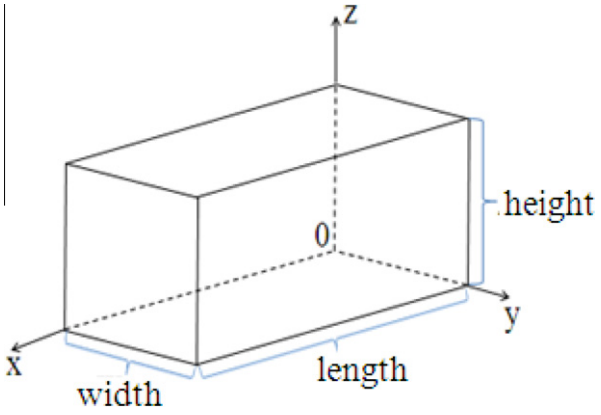


Fig. 1. Container in three-dimensional coordinate system.

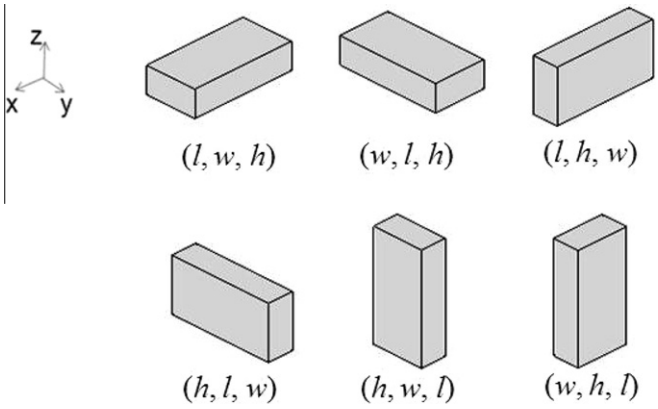


Fig. 2. Item orientations.

In the CLPSP, each item has a high or low priority. Without loss of generality, we assume the following:

(i) The high-priority items can be fully packed into one container (otherwise, the CLP algorithm can be used to pack the high-priority items).
(ii) All the items (high- and low-priority items) cannot be fully packed into one container (otherwise, the CLP algorithm can be used to pack all of them).

Consequently, the objective of the CLPSP is to maximize the volume of the packed items under the constraint that the high-priority items are not left behind.

3. Description of the algorithm

This algorithm hierarchically consists of a subordinated and a superior module. The subordinated module is a greedy heuristic, which serves the complete loading of the container. The superior module is a tree search which improves the solution generated by the greedy heuristic. The proposed algorithm focuses not only on the criteria for block selection but also on the appropriate allocation of high- and low-priority items.

3.1. Greedy heuristic with five evaluations

The proposed greedy heuristic is similar to other existing block-building approaches (e.g., Eley, 2002; Parreño et al., 2008). The container is filled with blocks. Each block is composed of identical items that have the same orientation. As is usual in block-building approaches, each feasible placement position where a block may be packed is called a (n) (empty) space. Beginning with an empty container, which is initialized as the first space, new spaces are generated when a block is packed in a space. The main difference between the proposed heuristic and other existing approaches lies in the evaluation functions for selecting blocks. Five evaluations are defined for block selection. Therefore, the heuristic is called GH5E (greedy heuristic with five evaluations). Before the description of the entire procedure of the heuristic, we highlight the strategies of block building and block evaluation as follows.

3.1.1. Block building

As shown in Fig. 3, let *sl*, *sw* and *sh* be the length, width and height, respectively, of a space *S*. For a block that is composed of items of a specific type and orientation, let *il*, *iw* and *ih* be the dimensions of the items on the x-, y- and z-axis, respectively; *N* be the residual number of the items of the given type; *xN*, *yN*

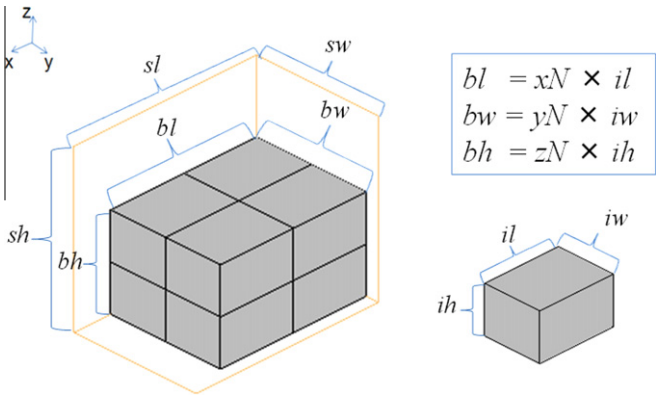


Fig. 3. Block in space.

and zN be the number of items along the x-, y- and z-axis in the block, respectively; and bl, bw and bh be the length, width and height of the block, respectively.

A block can be packed into S if the following constraints are satisfied:

$$xN \times yN \times zN <= N, \quad (1)$$

$$bl = xN \times il <= sl, \quad (2)$$

$$bw = yN \times iw <= sw, \quad (3)$$

$$bh = zN \times ih <= sh. \quad (4)$$

Example 1. Assume that the three dimensions of the space S are $sl \times sw \times sh = 370 \times 250 \times 220$ along the x-, y- and z-axis, respectively. There are two types of items: I_1 and I_2. For I_1 only the orientation "(l,w,h)" is permitted; for I_2 the orientations "(l,w,h)" and "(w,l,h)" are permitted.

The length, width and height of I_1 are 100, 120 and 90, respectively.
The length, width and height of I_2 are 240, 45 and 200, respectively.
The residual numbers of I_1 and I_2 are 4 and 2, respectively.

Fig. 4 shows all the possible blocks (B1–B12) that satisfy constraints (1)–(4). Note that the list of blocks is sorted according to the following criteria:

Main criterion: The order of item types (I_1,I_2).
Tie-breaker 1: The order of orientations $((l,w,h),(w,l,h),(l,h,w),(h,l,w),(h,w,l),(w,h,l))$.
Tie-breaker 2: Decreasing order of bw.
Tie-breaker 3: Decreasing order of bh.
Tie-breaker 4: Decreasing order of bl.

Blocks B1–B8 are made up of $I_1$with orientation (l,w,h), i.e., $il = 100$, $iw = 120$, $ih = 90$ (Fig. 4 (a));
Blocks B9 and B10 are made up of I_2 with orientation (l,w,h), i.e., $il = 240$, $iw = 45$, $ih = 200$ (Fig. 4 (b));
Blocks B11 and B12 are made up of I_2 with orientation (w,l,h), i.e., $il = 45$, $iw = 240$, $ih = 200$ (Fig. 4 (c)).

Therefore, for items of the same type with identical orientation, multiple types of blocks can be built by changing the item numbers along the x-, y- and z-axis. Considering all of the item types and permissible orientations, a large number of blocks can be built. As mentioned in Example 1, the list of all the possible blocks is sorted according to certain criteria. It is crucial to evaluate the possible blocks and select a block for packing into the space.

3.1.2. Block evaluation

Five evaluation functions for block selection are defined as follows:

(E1) – min ((*sl*–*bl*), (*sw*–*bw*),(*sh*–*bh*))
(E2) $bw \times bh$
(E3) $bl \times bh$
(E4) $bl \times bw$
(E5) $bl \times bw \times bh$

All of the above evaluations are greedy criteria and should be maximized. E1 evaluates the utilization of one dimension (i.e., the length, width or height of the space), E2–E4 evaluate the utilization of two dimensions (i.e., the YZ-Area, XZ-Area or XY-Area of the space), and E5 evaluates the utilization of three dimensions (i.e., the volume of the space). A number of experiments have shown that no single evaluation is efficient for all situations, and the alternative, that makes use of five evaluations, is much more efficient. The frequencies of the five evaluations in the numerical experiments are shown in 4.2.1.

The five evaluations can be applied in either a simultaneous manner or a sequential manner, as described below.

- Simultaneous manner: a block that achieves the highest value for one of the five evaluations is called a feasible block, and it may be selected for packing into the space.
- Sequential manner: a block is selected from the list of possible blocks according to the following criteria:

Main criterion: Largest value of E1.
Tie-breaker 1: Largest value of E2.
Tie-breaker 2: Largest value of E3.
Tie-breaker 3: Largest value of E4.

If multiple blocks satisfy the above criteria, only the first one among them is selected. Evaluation E5 is not mentioned here because if two blocks achieve identical values for E2, E3 and E4, respectively, they must also achieve an identical value for E5.

In the simultaneous manner, usually multiple blocks can be selected as feasible blocks. However, in the sequential manner, only one block is selected for packing into the space. The simultaneous manner is used in the tree search, which will be described in Section 3.2, and the sequential manner is used in the greedy heuristic.

3.1.3. Procedure of the greedy heuristic

Step 0: Initialize
The current space S:=whole empty container.
The space list *sList*:={S}.
Step 1: Select block
If there is no residual item, stop. Otherwise, build all of the possible blocks (i.e., blocks that satisfy constraints (1)–(4)) in the current space S. Then, from the list of possible blocks, select a feasible block by using the five evaluations E1–E5 (in the sequential manner described before) and pack it into S. A detailed explanation of this step has been provided in Sections 3.1.1 and 3.1.2.
Step 2: Generate new spaces
Split the residual space of S into new spaces, and add them into *sList*. Delete S from *sList*. Merge any two contiguous spaces in *sList* if they satisfy certain conditions. Mark any space in *sList* as "unusable" if no residual item can be

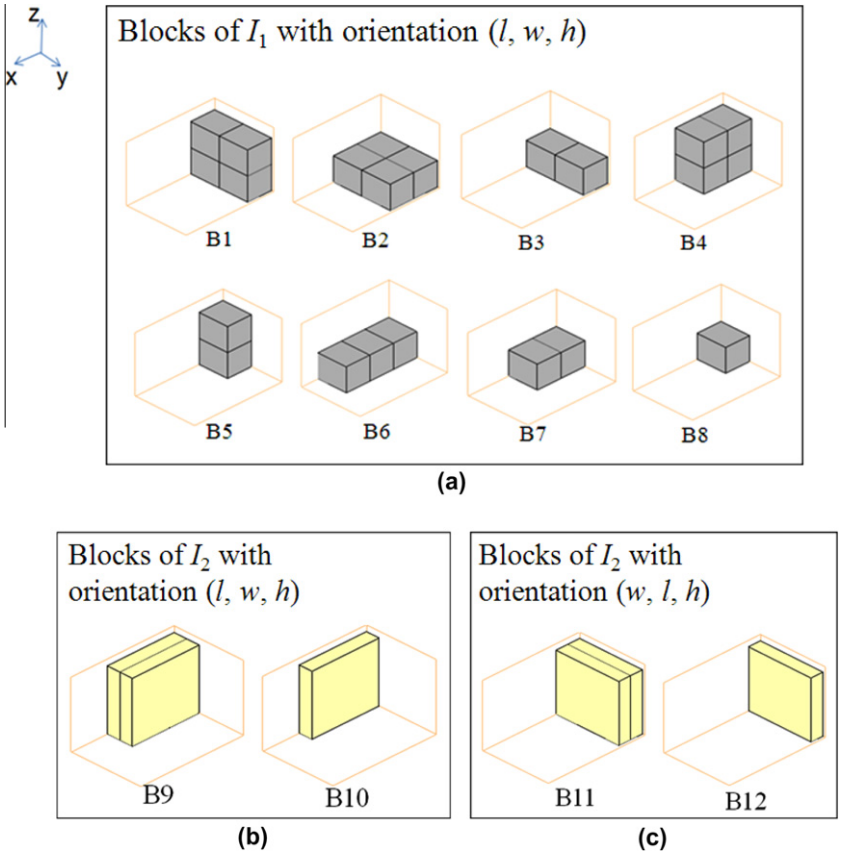


Fig. 4. Possible blocks.

packed into it. This procedure will be described in more detail in Section 3.1.4, which discusses space splitting and merging.

Step 3: Select space

If all of the spaces in *sList* are marked as "unusable", stop. Otherwise select the usable space (i.e. the space not marked as "unusable") that has the smallest Euclidean distance between its lower back left corner and the origin. Assign the selected space to *S* and go to *Step 1*.

The entire process from *Step 1* to *Step 3* is called an iteration of the heuristic. In each iteration, a block is selected and packed into the container. Additional iterations are carried out until either all items have been packed or no more items can be packed into the container.

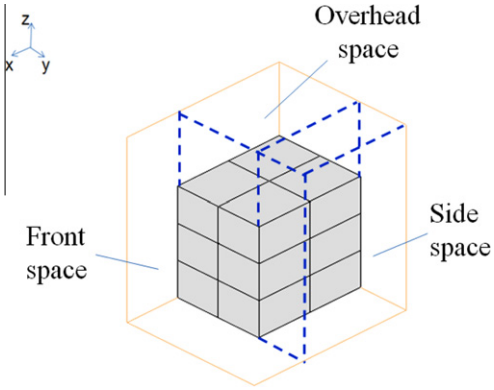


Fig. 5. Split space.

3.1.4. Space splitting and merging

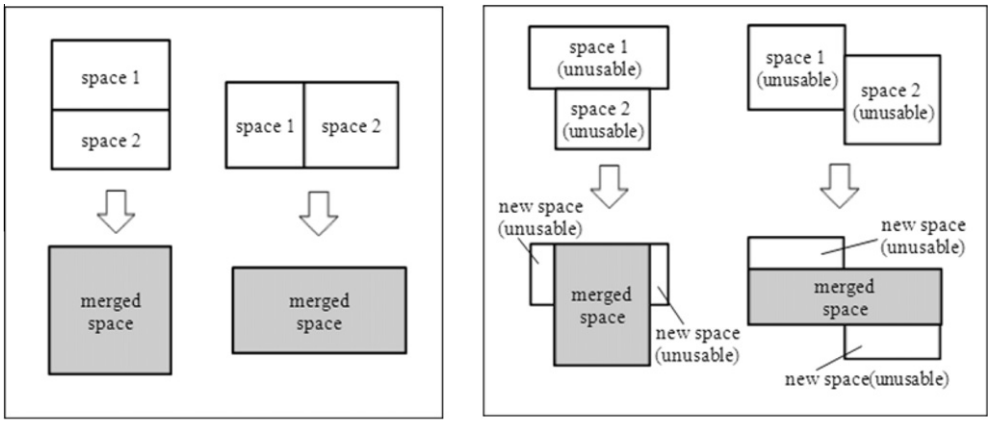
The residual space of *S* can be split into three sub-spaces (i.e., the front space, the side space and the overhead space shown in Fig. 5). For each of the three newly generated spaces, if it is too small to contain any of the residual items, mark it as "unusable". Then add the three newly generated spaces into *sList* and delete *S* from *sList*.

Any two adjoining spaces in the space list that have the same z-coordinate (for their lower back left corner) can be merged if one of the following conditions is satisfied:

(i) The two spaces have a common edge along the x-axis or y-axis as shown in Fig. 6 (a).

(ii) The two spaces do not have a common edge along the x-axis or y-axis, but both are marked as "unusable" as shown in Fig. 6 (b).

In both of these conditions, the merged space replaces the two original spaces in the space list. If the merged space is too small to contain any of the residual items, it is marked as "unusable". Furthermore, in condition (ii), the contiguous edges of two spaces do not completely coincide with each other. Therefore, other new spaces are generated. They are marked as "unusable" and added into the space list. The advantage of condition (ii) is that after being merged, the unusable spaces can be reused as much as possible. The method of merging spaces has also been proposed by other authors (e.g., Bortfeldt and Gehring, 2001; Eley, 2002). However, Eley only considered the merging in condition (i), and Bortfeldt and Gehring only considered the merging of the front space and side space, which are sub-spaces that are generated from the same space.



(a) Two spaces have a common edge. (b) Two spaces are both unusable.

Fig. 6. Merge spaces (perspective: top view).

3.2. Tree search for the general CLP

To improve the solution generated by the GH5E, a tree search called TRS_GH5E (Tree Search based on the GH5E) is implemented, which allows the selection of different blocks for each space.

As mentioned in Section 3.1.2, when using evaluations E1–E5 in the sequential manner, only one block is selected in each iteration of the GH5E. However, in the tree search, the five evaluations are applied in the simultaneous manner. Usually multiple feasible blocks are selected, and these blocks constitute the branches of the search tree.

The root node represents an empty container, and each node of the search tree represents a partially filled container (i.e., a partial solution). Each node is branched into some sub-nodes according to all of the feasible blocks that achieve the highest value for one of the five evaluations. For example, as shown in Fig. 7, the first three sub-nodes are generated according to the blocks B1, B2 and B3 that are selected using evaluation E1, and other sub-nodes are generated according to other blocks that are selected using E2, E3, E4 or E5. The set of all partial solutions at the same depth in the search tree is called a partial solution list.

Considering the exponentially increasing number of nodes, only a fixed number (parameter *breadth*) of nodes are retained from each depth. Similar to Eley (2002), a best search strategy is applied, which selects *breadth* nodes that obtained the highest ranking from an evaluation function. This function should not only consider the volume utilization obtained thus far, but it should also evaluate the potential for filling the residual spaces with the residual items. The function is a lower bound (for the volume utilization of the container) derived by filling the residual spaces of the corresponding partial solution by applying the greedy heuristic (GH5E). Furthermore, to avoid one good solution replacing all other good solutions, we use the following criterion: if multiple nodes at the same depth in the tree achieve the same evaluation function value, only the nodes among them with the largest volume utilization obtained thus far (not including the items packed using the greedy heuristic) are retained, and the remaining nodes are deleted.

The tree search is described by the pseudo-code shown in Fig. 8.

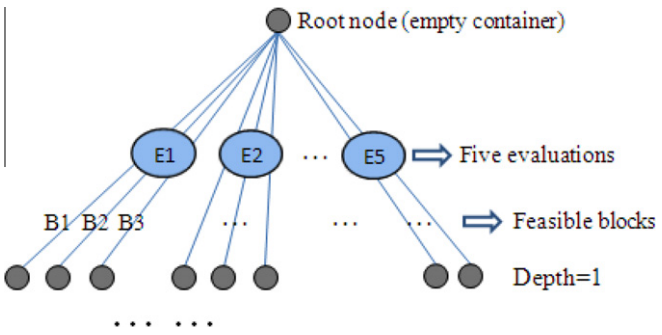


Fig. 7. Tree search for the CLP (TRS_GH5E).

3.3. Generalizing algorithm for the CLPSP

The TRS_GH5E algorithm can be generalized and adapted to solve the CLPSP when the constraint of shipment priority is considered. The generalized tree search algorithm is called TRS_GH5ESP (TRS_GH5E considering shipment priority).

Five alternative evaluations (E1–E5) are defined in the GH5E for block selection. Similarly, the following two criteria are proposed for the proper allocation of the high- and low-priority items:

(P1): Select a block composed of high-priority items.
(P2): Select a block composed of low-priority items.

These two criteria can be combined with evaluations E1–E5 to constitute ten evaluations for block selection: {P1E1, P1E2, P1E3, P1E4, P1E5, P2E1, P2E2, P2E3, P2E4, P2E5}. For example,

$$\mathrm{P1E2} = \begin{cases} bw \times bh, & \text{if the item is high priority} \\ -\infty, & \text{if the item is low priority} \end{cases}$$

$$\mathrm{P2E2} = \begin{cases} -\infty, & \text{if the item is high priority} \\ bw \times bh, & \text{if the item is low priority} \end{cases}$$

Similar to the TRS_GH5E algorithm, the blocks that achieve the highest values for one of the ten evaluations are called feasible blocks, and these blocks constitute the branches of the search tree (Fig. 9).

Considering the characteristics of the CLPSP, any infeasible solution (or partial solution) that violates the priority constraint must be eliminated. In detail, the process can be described as follows:

(i) For each solution, if high-priority items are left behind, the solution is eliminated.
(ii) For each partial solution, the total volume of the residual high-priority items and the total volume of the usable spaces are calculated. If the former exceeds the latter, then the partial solution is eliminated.

The procedure of deleting the infeasible partial solutions, as mentioned in (ii), is explained in terms of pseudo-code (Fig. 10).

```
best_solution: = empty container;
partial_solution_list: = {empty container};
while (partial_solution_list is not empty) do
  for all partial solution in partial_solution_list do
    for all feasible blocks that achieve the largest value for one of the five evaluations (E1~ E5) do
      add current feasible block to current partial solution to generate a new partial solution;
      add the new partial solution to partial_solution_list;
    endfor
    if no item could be packed into the container and the volume utilization
    of current partial solution is higher than that of best_solution then
      best_solution: = current partial solution;
    endif
    delete current partial solution from partial_solutions_list;
  endfor
  delete the partial solutions that have identical evaluation function value;
  select breadth best partial solutions from partial_solution_list;
endwhile
```

Fig. 8. Pseudo-code of the TRS_GH5E.

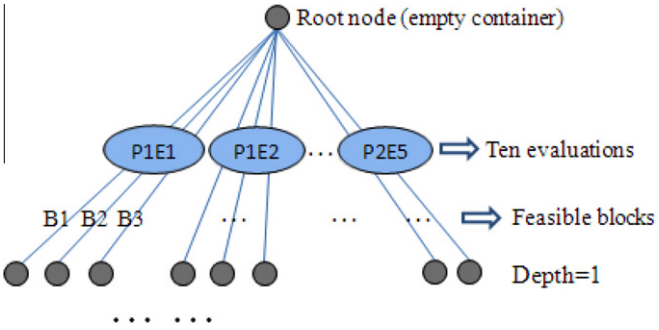


Fig. 9. Tree search for the CLPSP (TRS_GH5ESP).

The pseudo-code of the TRS_GH5ESP algorithm is shown in Fig. 11.

Similar to the TRS_GH5E algorithm, a lower bound for the volume utilization is derived by filling the residual spaces of the partial solution using the greedy heuristic, which serves as an evaluation function for selecting *breadth* best partial solutions from the partial solution list. Considering the constraint of shipment priority, the greedy heuristic must be modified as follows. In each iteration of the heuristic, if a block satisfies the following two conditions, it cannot be selected for packing:

(i) It is made up of low-priority items.
(ii) Its packing will result in the total volume of the residual high-priority items exceeding the total volume of the usable spaces.

```
for all partial solution in partial_solution_list do
  high_priority_volume: = 0;
  usable_volume: = 0;
  for all item types do
    if current item type is high priority then
      high_priority_volume: = high_priority_volume + residual volume of current item type;
    endif
  endfor
  for all space in space list do
    if current space is not marked as "unusable" then
      usable_volume: = usable_volume + volume of current space;
    endif
  endfor
  if high_priority_volume > usable_volume then
    delete current partial solution from partial_solution_list;
  endif
endfor
```

Fig. 10. Pseudo-code of "delete infeasible partial solutions".

```
best_solution: = empty container;
partial_solution_list: = {empty container};
while (partial_solution_list is not empty) do
  for all partial solution in partial_solution_list do
    for all feasible blocks that achieve the largest value for one of the ten evaluations (P1E1 ~ P2E5) do
      add current feasible block to current partial solution to generate a new partial solution;
      add the new partial solution to partial_solution_list;
    endfor
    if no item could be packed into the container and no high-priority items are left behind and
    the volume utilization of current partial solution is higher than that of best_solution then
      best_solution: = current partial solution;
    endif
    delete infeasible partial solutions;
    delete current partial solution from partial_solutions_list;
  endfor
  delete the partial solutions that have identical evaluation function value;
  select breadth best partial solutions from partial_solution_list;
endwhile
```

Fig. 11. Pseudo-code of the TRS_GH5ESP.

4. Computational results

The proposed algorithm was implemented in Visual C++ 2003 under Windows XP. All tests were performed on an Intel Core 2 U9300 PC (1.2 GHz, 2 GB RAM). Two well-known reference problem sets from literature are used for benchmarking purposes. These are the 15 test problems from Loh and Nee (1992) (LN problems) and the 700 test problems from Bischoff and Ratcliff (1995) (BR problems). For the LN problems, the number of item types ranges from

Table 1
Results from the LN problems. Bold values denotes the best result.

Problem	H_BR (1995)	TS_BG (1998)	T_E (2002)	GA_TT (2004)	G_MO (2005)	H_L (2007)	T_W (2008)	TRS_GH5E	Running time (s)
LN01	62.5	62.5	62.5	62.5	62.5	62.5	62.5	62.5	7
LN02	90.0(35)	96.7(28)	90.8(53)	91.3(31)	92.6(19)	89.7[a]	90.7(35)	**97.9(25)**	332
LN03	53.4	53.4	53.4	53.4	53.4	53.4	53.4	53.4	18
LN04	55.0	55.0	55.0	55.0	55.0	55.0	55.0	55.0	5
LN05	77.2	77.2	77.2	77.2	77.2	77.2	77.2	77.2	32
LN06	83.1(77)	96.2(32)	87.9(44)	90.5(52)	91.7(28)	91.4[a]	92.9(37)	**96.3(21)**	167
LN07	78.7(18)	84.7	84.7	84.7	84.7	84.6[a]	84.7	84.7	29
LN08	59.4	59.4	59.4	59.4	59.4	59.4	59.4	59.4	6
LN09	61.9	61.9	61.9	61.9	61.9	61.9	61.9	61.9	14
LN10	67.3	67.3	67.3	67.3	67.3	67.3	67.3	67.3	24
LN11	62.2	62.2	62.2	62.2	62.2	62.2	62.2	62.2	3
LN12	78.5	78.5	78.5	78.5	78.5	78.5	78.5	78.5	43
LN13	78.1(20)	85.6	85.6	85.6	85.6	85.6	85.6	85.6	61
LN14	62.8	62.8	62.8	62.8	62.8	62.8	62.8	62.8	10
LN15	59.5	59.5	59.5	59.5	59.5	59.5	59.5	59.5	19
Ave.	68.60	70.90	69.90	70.10	70.30	70.00	70.20	**70.93**	51

Note: the number of remaining items is given in parentheses.
[a] Denotes the occurrence of remaining items, but the number of items is not provided.

Table 2
Results from the LN problems considering priority. Bold values denotes the average value.

Problem	Original problem	High priority	Low priority	Volume utilization (%)	Running time (s)
LN01& 02	LN01, LN02	LN01	LN02	99.30	624
LN02_1	LN02	1, 2, 3, 4	5, 6, 7, 8	97.27	544
LN02_2	LN02	5, 6, 7, 8	1, 2, 3, 4	96.94	474
LN02_3	LN02	1, 2, 3, 4, 5	6, 7, 8	97.27	426
LN02_4	LN02	4, 5	1, 2, 3, 6, 7, 8	97.14	460
LN02_5	LN02	1, 3, 5, 7	2, 4, 6, 8	96.39	516
LN06_1	LN06	1, 2, 3, 4	5, 6, 7, 8	95.97	325
LN06_2	LN06	5, 6, 7, 8	1, 2, 3, 4	94.37	261
LN06_3	LN06	7, 8	1, 2, 3, 4, 5, 6	95.74	300
LN06_4	LN06	1, 2, 3, 4, 5	6, 7, 8	95.31	300
LN06_5	LN06	1, 3, 5, 7	2, 4, 6, 8	95.45	271
Ave.				**96.47**	**409**

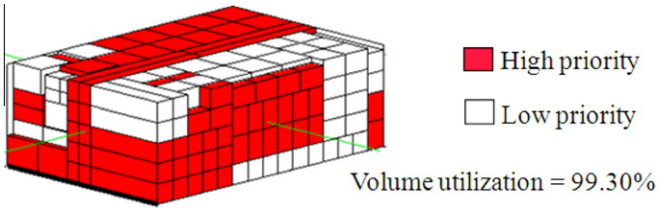


Fig. 12. Illustrative result of LN01& 02.

6 to 10, and the number of available items ranges from 100 to 250. The BR problems are divided into seven groups (BR1–BR7) each containing 100 problems. Each group is distinguished by the number of different item types which increases from 3 types of BR1 to 20 types of BR7. The average number of items in each type is 50.2 for BR1, but decreases continuously and is only 6.5 for BR7. The container used in the BR problems is the standard ISO 20FT container with three dimensions of 587 centimeters $\times$ 233 centimeters $\times$ 220 centimeters.

Due to the lack of universally acknowledged test data for the CLPSP, the test problems were generated from the above problems by adding shipment priorities to the different item types.

To achieve high volume utilization in reasonable computing time, the value of the *breadth* parameter was set to 100 for the CLP and 200 for the CLPSP.

4.1. Computational results for the LN problems

4.1.1. Results for CLP

The proposed TRS_GH5E algorithm has been compared with the following seven approaches:

- H_BR: the heuristic approach of Bischoff and Ratcliff (1995).
- TS_BG: the TS of Bortfeldt and Gehring (1998).
- T_E: the tree search approach of Eley (2002).
- GA_TT: the GA of Techanitisawad and Tangwiwatwong (2004).
- G_MO: the GRASP approach of Moura and Oliveira (2005).
- H_L: the heuristic approach of Liang et al. (2007).
- T_W: the tree search approach of Wang et al. (2008).

Table 1 shows the volume utilization (%) obtained from the eight algorithms, including the proposed TRS_GH5E algorithm. We found that in most problems other than LN02 and LN06, all items could be packed into one container by most algorithms. The best results for LN02 and LN06 were obtained with the proposed TRS_GH5E algorithm. Therefore, the proposed TS_GH5E algorithm obtained the highest average volume utilization for all 15 problems. These results show that the proposed TRS_GH5E algorithm has high validity for the general CLP in comparison with other algorithms.

The running time of the TRS_GH5E algorithm is shown in the last column of Table 1.

4.1.2. Results for CLPSP

The test problems were generated from the LN problems by randomly setting the shipment priorities of the item types. As shown in Table 2, for example, the problem "LN01& 02" was generated by combining the problems LN01 and LN02. The items in LN01 were all set to high priority, and the items in LN02 were all set to low priority. The problem "LN02_1" was generated from LN02 by setting the first four item types to high priority and the last four item types to low priority.

As shown in Table 2, even though some problems were generated from the same problem (LN02 or LN06), their volume utilizations differed depending on the assigned priorities. The results show that volume utilization depends not only on the item types, but also on the priorities of the items. The average of the volume utilizations for LN02_1–LN02_5 is 97.00%, which is higher than the results of all the algorithms except for the proposed TRS_GH5E algorithm in Table 1 for LN02. The average of the volume utilizations for LN06_1–LN06_5 is 95.37%, which is higher than the results of all the algorithms except for the TS_BG approach and the proposed TRS_GH5E algorithm in Table 1 for LN06. The results show that, despite the addition of a new constraint (shipment priority), the proposed algorithm obtains better results than most algorithms that do not consider the shipment priority in Table 1.

An illustrative result of LN01&02 is shown in Fig. 12.

4.2. Computational results for the BR problems

4.2.1. Results for CLP

The proposed TRS_GH5E algorithm has been compared with the following seven approaches:

- TS_BG: the TS of Bortfeldt and Gehring (1998).
- GA_GB: the parallel GA of Gehring and Bortfeldt (2002).
- T_E: the tree search approach of Eley (2002).
- LS_M: the parallel hybrid local search of Mack et al. (2004).
- H_B: the heuristic approach of Bischoff (2006).
- G_P: the GRASP approach of Parreño et al. (2008).
- T_FB: the tree search approach of Fanslau and Bortfeldt (2010).

The average volume utilizations (%) of each problem are shown in Table 3. The proposed TRS_GH5E algorithm competed well with the other approaches. Only the T_FB approach achieved higher average volume utilization for the 700 problems. The highest average volume utilization for BR5 was obtained with the proposed TRS_GH5E algorithm.

The average running time of the TRS_GH5E algorithm for each problem is shown in the last column of Table 3.

Table 4 shows the frequencies of the evaluations E1–E5 in the experiment on the BR problems. For example, in BR1, 1088 blocks were packed into 100 containers, out of which 978 blocks achieved the highest value with E1 and 570 blocks with E2. The evaluation E1 is used more often than other evaluations (90.6%). It should be noted that in some cases, a block achieves the highest value

Table 3
Results from the BR problems. Bold values denotes the best result.

Problem	TS_BG (1998)	GA_GB (2002)	T_E (2002)	LS_M (2004)	H_B (2006)	G_P (2008)	T_FB (2010)	TRS_GH5E	Running time (s)
BR1	92.63	88.10	88.00	93.70	89.39	93.85	**94.51**	93.90	114
BR2	92.70	89.56	88.55	94.30	90.26	94.22	**94.73**	94.54	145
BR3	92.31	90.77	89.50	94.54	91.08	94.25	**94.74**	94.35	213
BR4	91.62	91.03	89.30	94.27	90.90	94.09	**94.41**	94.08	308
BR5	90.86	91.23	89.00	93.83	91.05	93.87	94.13	**94.17**	413
BR6	90.04	91.28	89.20	93.34	90.70	93.52	**93.85**	93.48	500
BR7	88.63	91.04	88.00	92.50	90.44	92.94	**93.20**	92.82	663
Ave.	91.26	90.43	88.79	93.78	90.55	93.82	**94.22**	93.91	337

Table 4
Frequencies of the five evaluations.

Problem	E1 (%)	E2 (%)	E3 (%)	E4 (%)	E5 (%)	Total
BR1	978(89.9)	570(52.4)	536(49.3)	563(51.7)	529(48.6)	1088
BR2	1397(89.6)	856(54.9)	776(49.8)	798(51.2)	833(53.4)	1559
BR3	1823(87.9)	1152(55.5)	945(45.5)	949(45.7)	1096(52.8)	2075
BR4	2132(90.3)	1283(54.3)	1056(44.7)	1068(45.2)	1235(52.3)	2361
BR5	2538(92.7)	1444(52.7)	1208(44.1)	1309(47.8)	1389(50.7)	2738
BR6	2751(91.1)	1624(53.8)	1352(44.8)	1429(47.3)	1619(53.6)	3019
BR7	3264(91.0)	1932(53.9)	1518(42.3)	1656(46.2)	1888(52.6)	3586
Total	14883(90.6)	8861(53.9)	7391(45.0)	7772(47.3)	8589(52.3)	16426

Table 5
Results from the BR problems considering priority.

Problem	High priority	Low priority	Volume utilization (%)	Running time (s)
BR1_1	1	2, 3	93.45	318
BR1_2	3	1, 2	93.48	370
BR4_1	1, 2,..., 5	6, 7,...,10	92.99	636
BR4_2	6, 7,..., 10	1, 2,..., 5	93.14	597
BR7_1	1, 2,..., 10	11, 12,..., 20	91.75	1213
BR7_2	11, 12,..., 20	1, 2,..., 10	91.74	1310
Ave.			92.76	741

with multiple evaluations; therefore, the sum of the frequencies of E1–E5 is larger than the total number of the packed blocks.

4.2.2. Results for CLPSP

As shown in Table 5, the test problems of CLPSP were generated from problems BR1, BR4 and BR7 by adding different priorities in the following manner: BR1_1, BR4_1 and BR7_1 were generated by setting the first $[n/2]$ (n is the number of item types in each problem) item types to high priority and the remaining item types to low priority, and BR1_2, BR4_2 and BR7_2 were generated by setting the last $[n/2]$ item types to high priority and the remaining item types to low priority. Despite the addition of the shipment priority constraint, high volume utilizations were obtained.

The average running time of the TRS_GH5ESP algorithm for each problem is shown in the last column of Table 5.

5. Conclusions

A tree search method based on the greedy heuristic with five evaluations (i.e., TRS_GH5E) has been proposed for solving the CLP. Five alternative evaluations for the utilization of one, two and three dimensions of the container space have been defined for block selection. The different blocks selected by each evaluation constitute the branches of the search tree. A method of space splitting and merging has also been embedded in the algorithm for making efficient use of the container space. The 15 test problems generated by Loh and Nee (1992) and the 700 test problems generated by Bischoff and Ratcliff (1995) have been used to examine the validity of the TS_GH5E algorithm. The computational results show that the TRS_GH5E algorithm obtains the highest average volume utilization for the LN problems and the second highest average volume utilization for the BR problems compared to the other algorithms.

A tree search considering the shipment priority (TRS_GH5ESP) generalized from the TRS_GH5E has been proposed to solve the CLPSP. The computational results show that the volume utilization depends not only on the item type but also on the priorities of the items. High volume utilization has been obtained despite the addition of the shipment priority constraint. Thus, the proposed TRS_GH5ESP algorithm is considered to be useful for solving practical problems with shipment priority.

One shortcoming of this algorithm is that it does not consider the position of items in a container with different destinations (i.e., multi-drop situation), which may increase the effort involved in unloading and reloading. Further research is required to find a balance between high volume utilization and reduced loading and unloading effort. Reduction of the computational effort is also an issue for future research.

Acknowledgement

The authors wish to express their appreciation to Beijing UNITY Information Technology Co., Ltd. The computational experiments have been executed using the CLS (Container Loading System) developed by the company.

References

Bischoff, E.E., 2006. Three dimensional packing of items with limited load bearing strength. European Journal of Operational Research 168 (3), 952–966.
Bischoff, E.E., Ratcliff, M.S.W., 1995. Issues in the development of approaches to container loading. Omega – International Journal of Management Science 23 (4), 377–390.
Bortfeldt, A., Gehring, H., 1998. Ein tabu search-verfahren für containerbeladeprobleme mit schwach heterogenem kistenvorrat. OR Spektrum 20 (4), 237–250.
Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research 131 (1), 143–161.
Christensen, S.G., Rousøe, D.M., 2009. Container loading with multi-drop constraints. International Transactions in Operational Research 16 (6), 727–743.
Davies, A.P., Bischoff, E.E., 1999. Weight distribution considerations in container loading. European Journal of Operational Research 114 (3), 509–527.
Eley, M., 2002. Solving container loading problems by block arrangement. European Journal of Operational Research 141 (2), 393–409.
Fanslau, T., Bortfeldt, A., 2010. A tree search method for solving the container loading problem. Informs Journal on Computing 22 (2), 222–235.
Gehring, H., Bortfeldt, A., 1997. A genetic algorithm for solving the container loading problem. International Transactions in Operational Research 4 (5/6), 401–418.
Gehring, H., Bortfeldt, A., 2002. A parallel genetic algorithm for solving the container loading problem. International Transactions in Operational Research 9 (4), 497–511.
George, J.A., Robinson, D.F., 1980. A heuristic for packing boxes into a container. Computer and Operations Research 7 (3), 147–156.
Jin, Z., Ohno, K., Du, J., 2004. An efficient approach for the three-dimensional container packing problem with practical constraints. Asia-Pacific Journal of Operational Research 21 (3), 279–295.
Liang, S., Lee, C., Huang, S., 2007. A hybrid meta-heuristic for the container loading problem. Communications of the IIMA 7 (4), 73–84.

Loh, T.H., Nee, A.Y.C., 1992. A packing algorithm for hexahedral boxes. In: Proceedings of the Conference of Industrial Automation, Singapore, pp. 115–126.
Mack, D., Bortfeldt, A., Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. International transactions in operational research 11 (5), 511–533.
Moura, A., Oliveira, J.F., 2005. A GRASP approach to the container-loading problem. IEEE Intelligent Systems 20 (4), 50–57.
Parreño, F., Alvarez-Valdes, R., Tamarit, J.M., Oliveira, J.F., 2008. A maximal-space algorithm for the container loading problem. Informs Journal on Computing 20 (3), 412–422.
Pisinger, D., 1998. A tree search heuristic for the container loading problem. Ricerca Operativa 28 (87), 31–48.
Pisinger, D., 2002. Heuristics for the container loading problem. European Journal of Operational Research 141 (2), 382–392.
Techanitisawad, A., Tangwiwatwong, P., 2004. A GA-based Heuristic for the interrelated container selection and loading problems. Industrial Engineering and Management System 3 (1), 22–37.
Wang, Z.J., Li, K.W., Levy, J.K., 2008. A heuristic for the container loading problem: a tertiary-tree-based dynamic space decomposition approach. European Journal of Operational Research 191 (1), 86–99.