European Journal of Operational Research 214 (2011) 501–511

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

The multiple container loading cost minimization problem

Chan Hou Che [a], Weili Huang [a], Andrew Lim [a], Wenbin Zhu [b,c,*]

[a] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave, Kowloon Tong, Hong Kong
[b] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[c] Department of Logistics and Maritime Studies, Faculty of Business, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 31 March 2010
Accepted 18 April 2011
Available online 27 April 2011

Keywords:
Packing
Heuristics
Container loading
Integer programming
Design of experiments

ABSTRACT

In the shipping and transportation industry, there are several types of standard containers with different dimensions and different associated costs. In this paper, we examine the multiple container loading cost minimization problem (MCLCMP), where the objective is to load products of various types into containers of various sizes so as to minimize the total cost. We transform the MCLCMP into an extended set cover problem that is formulated using linear integer programming and solve it with a heuristic to generate columns. Experiments on standard bin-packing instances show our approach is superior to prior approaches. Additionally, since the optimal solutions for existing test data is unknown, we propose a technique to generate test data with known optimal solutions for MCLCMP.

1. Introduction

Our team was contracted by a buying agent for a large multinational retailer to investigate better ways to formulate packing plans for the loading of goods into multiple containers. The agent's procurement department is responsible for ordering products from manufacturers located in Asia, inspecting the products to ensure quality, and finally shipping the products to retail stores spread across Europe. As a result, one of the procurement department's tasks is to arrange the products for shipping. Typically, the goods are loaded into containers of various standard sizes with different costs. The task is to select a set of containers that can contain all items while minimizing the total shipping cost.

This problem is a variant of a class of problems known as the multiple container loading problem (MCLP), also called the multiple container packing problem. The MCLP describes the scenario where, given a set of 3-D rectangular boxes of different types and a set of 3-D rectangular containers of different types, the objective is to store the boxes into the containers as efficiently as possible. There are many variants to the MCLP; the particular variant our paper addresses is referred to as the multiple container loading cost minimization problem (MCLCMP).

The MCLCMP is defined as follows. We are given a number of rectangular containers of M types, represented by C_1, $C_2,\ldots,C_M$, that differ in terms of dimensions and cost. The cost for each type is given by $c_1, c_2,\ldots,c_M$, and there are an unlimited number of each container available. We are also given a number of rectangular boxes of N types with different dimensions, represented by B_1, $B_2,\ldots,B_N$; there are n_j, $1 \leqslant j \leqslant N$ available boxes of type j. The objective of the MCLCMP is to produce a set of packing plans such that all boxes are orthogonally packed into the containers and the total cost of containers used is minimized. Our technique can also handle a limited number of containers (i.e., there are m_t, $1 \leqslant t \leqslant M$ available containers of type t) with minor modifications (as detailed in Section 8). We do not consider supporting area constraints, although our approach can be modified to do so.

Our approach is an extension of the integer linear programming formulation proposed by Eley (2003) for the MCLP. We add a new parameter to this formulation that controls the percentage of boxes to be packed, and perform a binary search on this variable. In addition, we devise three different strategies to quickly generate packing patterns for single containers along with a subroutine that augments the set of packing patterns while searching for solutions for the IP-problem. We tested this technique using the standard set of 47 bin-packing test instances proposed by Ivancic et al. (1989), and found that it produced superior results in comparison with other existing approaches.

However, the optimal solutions for these test instances are unknown, and so we cannot evaluate the objective performance of our approach based on this test set. Hence, we also devise a technique for generating test instances for the MCLCMP with verifiably optimal known solutions.

The remainder of this paper is organized as follows. We first provide an overview of the existing literature related to our study

* Corresponding author at: Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Tel.: +852 64067667.
E-mail address: i@zhuwb.com (W. Zhu).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.04.017

in Section 2. Our new parameterized linear integer programming formulation is presented in Section 3; we perform binary search on the parameter based on a number of generated packing patterns. Section 4 describes our three strategies for generating packing patterns, which are used by the binary search algorithm given in Section 5. In order to obtain a more objective evaluation of the strength of our approach, we describe a new technique in Section 6 for generating test instances for this problem with known optimal solutions. We test our approach on both the bin-packing instances used in current research and on our new instances, and present the results in Section 7. Our paper concludes in Section 8, where we suggest ways to further extend our technique to handle other variants of the MCLP.

2. Related work

Following the improved typography for cutting and packing problems introduced by Wäscher et al. (2007), the MCLCMP can be considered a variant of either the multiple stock-size cutting stock problem (MSSCSP) or the multiple bin-size bin packing problem (MBSBPP) depending on the heterogeneity of the boxes, where the objective is to minimize the cost of the containers.

There is little existing literature that examines the MCLP and its variants directly. Takahara and Miyamoto (27) studied a similar problem where the objective was to minimize wasted volume, which is equivalent to minimizing the total capacity of containers used. This may be regarded as a variant of the MCLCMP where the containers of various types have equal unit costs. An evolutionary approach was proposed, where a pair of sequences (one for boxes and one for containers) is used as the genotype, and a heuristic was used to determine the loading plan given the sequence of boxes and containers. The MCLCMP and some possible variants were also studied by Eley (2003) who proposed a bottleneck assignment approach. The author used a set cover formulation for linear integer programming using pre-generated packing patterns, which were found using a tree search based heuristic. We adapted this set cover formulation in our work; details are given in Section 3. Set-cover-based heuristic has also been proposed by Monaci and Toth (2006) for both 1D and 2D bin packing problems.

Although the 3-D bin packing problem (3D-BPP) is reasonably well-studied (Verweij, 1996; Lodi et al., 2002; Crainic et al., 2008; Fekete and van der Veen, 2007; Crainic et al., 2009; Faroe et al., 2003; Parreño et al., 2008; Martello et al., 2000), this problem differs from multiple container loading in two crucial ways. Firstly, 3D-BPP approaches generally assume that all items cannot be rotated; this is an unrealistic assumption since most items can at least be rotated by 90°. Secondly, 3D-BPP assumes that all bins are of the same size. However, containers come in multiple standard sizes, and there is a trade-off between the size of the container and its cost. Both of these factors must be addressed for the MCLCMP.

Earlier literature often recommended ways to adapt procedures for the well-analyzed single container loading problem (SCLP) for multiple containers. Possible strategies include the sequential strategy, where single containers are filled in turn using SCLP approaches (Ivancic et al., 1989; Bischoff and Ratcliff, 1995a,b; Eley, 2003; Lim and Zhang, 2005); the pre-assignment strategy, which first assigns boxes to containers before loading (Terno et al., 2000); and the simultaneous strategy, which considers multiple containers during the loading of boxes (Eley, 2002).

Our technique makes use of SCLP heuristics as sub-routines. Dyckhoff and Finke (1992) provided an excellent review of the SCLP, and Christensen and Rousøe (2009) also provided a brief introduction of some approaches in their technical report.

3. Linear integer programming formulation for the MCLCMP

Our approach is an extension of the set partitioning formulation proposed by Eley (2003) for the multiple container loading problem. However, the term *set partitioning formulation* used in the original publication is a misnomer since the formulation is in fact a set *cover*; we will henceforth refer to the formulation as a set cover formulation. We assume that there is a set P of single container packing patterns indexed by i, and the box types are indexed by j. Each packing pattern p_i fills a container with associated cost c_i with b_{ij} boxes of type j. Let x_i be the integer decision variable for the ith column that represents the number of packing patterns p_i used. A feasible solution to MCLCMP can be obtained by solving the following optimization model:

$$SC: z = \min \sum_{p_i \in P} c_i x_i \tag{1}$$

$$\text{s.t.} \sum_{p_i \in P} x_i b_{ij} \geqslant n_j, \quad \forall j \tag{2}$$

$$x_i \geqslant 0 \quad \text{and integer}, \quad i = 1, \ldots, |P|, \tag{3}$$

where the objective function (1) computes the total cost of all selected packing patterns, and the inequality (2) ensures that the selected packing patterns have sufficient space to pack the boxes of each type, where n_j is the number of boxes to be packed for box type j.

It is clear that an optimal solution of the basic model SC is a feasible solution for the MCLCMP instance. However the solution is often suboptimal due to two reasons. Firstly, the set of packing patterns P must encompass the set of all valid packing patterns, and this is computationally prohibitive except for very small problem instances. As a result, a sample of valid packing patterns must be taken.

Secondly, the selected containers may offer excessive capacity for some box types, i.e., for some j the inequality (2) is strictly greater. The presence of excessive capacity offers an opportunity to reduce the cost by selecting a set of containers with smaller capacity. Hence, we introduced a loading factor a to inequality (2), and parameterized the model as follows:

$$SC(a): z = \min \sum_{p_i \in P} c_i x_i, \tag{4}$$

$$\text{s.t.} \sum_{p_i \in P} x_i b_{ij} \geqslant a \cdot n_j, \quad \forall j, \tag{5}$$

$$x_i \geqslant 0 \quad \text{and integer}, \quad i = 1, \ldots, |P|. \tag{6}$$

The loading factor a is the percentage of boxes that are guaranteed to be loaded into selected containers. For example, the solution found by SC(0.97) will guarantee that at least 97% of each type of box can be loaded into the containers selected, while the remaining 3% has no guarantee. It may be possible to load the unguaranteed 3% into the containers selected by SC(0.97), whereupon we would obtain a feasible solution of MCLCMP. Note that the cost of the optimal solution of SC(a) is a non-decreasing function of a because if $a < b$, then any feasible solution to SC(b) is also a feasible solution to SC(a) (i.e., an optimal solution of SC(a) is at least as good as an optimal solution for SC(b)).

We make use of this parameterized formulation as follows. First, we generate a set P of packing patterns, which are solutions to the SCLP, using three strategies that employ fast heuristics for the SCLP. We then perform binary search on the range $0.01 \leqslant a \leqslant 1.0$ for *max_iter* iterations, employing a commercial IP solver for each iteration. If there are boxes that are not loaded in a given iteration, we invoke a routine that attempts to load these boxes without increasing the number of containers. This routine has the side-effect of possibly increasing the number of packing patterns (i.e., increasing the size of P).

4. Strategies for generating packing patterns

We developed three fast heuristic strategies to generate a reasonably large number of diversified packing patterns, namely the *G4-heuristic Bin-packing (G4BP)* strategy; the *Sequential Application of GRASP (S-GRASP)* Strategy; and the *Application of GRASP on Combinations (C-GRASP)* Strategy. The aim of all three strategies is to generate diverse and utilization-efficient packing patterns for different combinations of container and boxes by finding solutions for SCLP instances, for use when solving the above IP-problem.

4.1. The G4-heuristic bin-packing strategy

In the G4-heuristic Bin-packing (G4BP) strategy, we extend the G4-heuristic by Scheithauer and Terno (1996) for the 2-D pallet loading problem to 3-D to solve the SCLP with homogenous boxes. This is an application of the n-tet graph approach devised by Lins et al. (2002).

Let the dimensions of a box be arbitrarily assigned as its length, width, and height and denoted by l, w, and h, respectively. A box can be placed in six possible ways, of which there are two each with the l-side placed vertically, and similarly for the w- and h-sides (resulting in 3 *vertical orientations*). We assume that a container of dimensions $L \times W \times H$ is placed such that the L-side is parallel to the x-axis, the W-side is parallel to the z-axis, and the H-side is parallel to the y-axis. We fill the container layer by layer from the bottom up along the vertical direction with boxes of the same type. For each layer, we impose the restriction that all boxes must be in the same vertical orientation, and we search for a packing that maximizes the number of boxes. Boxes in different layers can have different vertical orientations.

Consider the vertical orientation for a box where the h-side is vertical. We can create a layer of height h by packing as many boxes as possible in this vertical orientation. This problem is identical to placing as many rectangles of dimensions $l \times w$ into a rectangle of dimensions $L \times W$ as possible, which is an instance of the 2-D pallet loading problem.

We make use of the G4-heuristic (Scheithauer and Terno, 1996), which recursively divides the containing rectangle into 4 subrectangles as shown in Fig. 1. Assuming the sides of a rectangle are integral, then the number of ways in which it can be divided in this way is at most $L^2 \times W^2$. This algorithm produces high quality solutions for the 2-D pallet loading problem in pseudo-polynomial time. Let N_h be the number of boxes in the solution found by the G4-heuristic. Similarly, let N_l and N_w be the number of boxes in the solutions for the l- and w-side vertical orientations, respectively.

If we treat each layer as an item, then the SCLP becomes a one dimensional bin-packing problem with three types of items of sizes N_l, N_w, and N_h. Let I_l,I_w, and I_h be 0–1 constants that indicate whether boxes are allowed to be placed in the l-side, w-side, and h-side vertical orientations respectively, where 1 indicates that the vertical orientation is allowed and 0 otherwise; these values are dependent on the problem instance and are part of the input. Let x_l, x_w, and x_h be integer variables that denote the number of layers in which boxes are placed in the l-side, w-side, and h-side vertical orientations respectively. We can model this problem using linear integer programming:

$$\textbf{G4BP}: z = \max N_l x_l I_l + N_w x_w I_w + N_h x_h I_h, \tag{7}$$

$$\text{s.t. } N_l x_l I_l + N_w x_w I_w + N_h x_h I_h \leqslant H, \tag{8}$$

$$x_l, x_w, x_h \geqslant 0 \quad \text{and integer}, \tag{9}$$

$$I_l, I_w, I_h \in \{0, 1\}, \tag{10}$$

The objective (7) is to maximize the total number of boxes in all layers in a container, subject to the constraint (8) that the sum of the heights of all layers does not exceed the height of the container.

For each container type C_i and each box type B_j, we assume an unlimited supply of boxes to form an instance of the single container loading problem with homogeneous boxes. We use the G4BP strategy to find solutions for each instance, and add the resulting packing patterns into our set of packing patterns P. This strategy produces packing patterns that only employ one type of box, which may be particularly useful in the scenario where there are proportionally many boxes of one type, and therefore the optimal solution is likely to consist of some containers that are loaded entirely with one type of box. Furthermore, it is desirable in practice to load a container with only one type of product during shipping for ease of inventory management. This scenario is common in the sea freight industry, where forwarders often ship large quantities of a particular product.

In the above discussion, we assumed that the layers are built along the height direction from the bottom up. In fact, layers can also be built on the (W,H) face along the length direction, or on the (L,H) face along the width direction. For a given box type and container type, the G4BP strategy therefore solves the above linear program three times, once for each direction, and returns the best result. Hence, the linear program is solved $3MN$ times in total, resulting in MN packing patterns.

4.2. Sequential Application of GRASP Strategy

Given M types of containers and n boxes of N types, the Sequential Application of GRASP Strategy (S-GRASP) makes use of the Greedy Randomized Adaptive Search Procedure (GRASP) approach (Moura and Oliveira, 2005), which produces solutions for the SCLP. It repeatedly invokes GRASP first on a problem instance with one container and n boxes, then on reduced problem instances with the boxes that were used in previous solutions removed, until all boxes have been loaded. This is done for each of the M different types of containers.

The original GRASP algorithm for the SCLP made use of the wall building constructive heuristic GRMOD (George and Robinson, 1980), which built vertical layers corresponding to the (W,H) face of the container (called *walls*) that consist of $n_1 \times n_2$ homogeneous boxes in the same orientation. The walls are placed side by side along the length dimension (which simulates the loading of goods via a rear door situated on the (W,H) face of the container). It is performed in two phases. In the construction phase, an initial solution is generated: the algorithm repeatedly picks a region of free space (called *residual space*), then uniformly randomly chooses one out of the top δ% of all possible walls in terms of volume utilization and loads it into the residual space, subject to the availability of the boxes. The value of δ is dynamically adjusted every 500 iterations based on information gathered from previous iterations (details can be found in Moura and Oliveira (2005)). In the improvement phase, a part of the solution is undone, and then redone by greedily choosing the best wall to load in terms of volume utilization. In our implementation, if the construction phase runs for k steps, then we perform the undo–redo operation for all steps k down to 2.

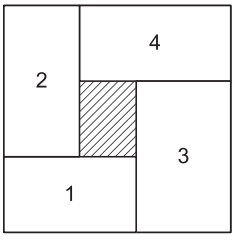

Fig. 1. G4 Heuristic.

We made two modifications to the GRMOD heuristic in our implementation. Firstly, we used a different way to generate residual spaces. In the original GRMOD heuristic, after a wall is built, three residual spaces are generated. The left diagram in Fig. 2 illustrates the three residual spaces $c1$, $c2$, and $c3$ generated by the GRMOD heuristic after wall b is built. However, there are in fact 6 different ways to partition the residual space, including the alternative illustrated in the right diagram. Both ways shown generate a residual space $c1$ that is fully supported by the placed wall, which allows the checking of supporting area constraints if desired.

If a residual space is too small to hold the smallest box remaining, then it cannot be used at all and is called *rejected space*, otherwise it is called *useful space*. It would be desirable to minimize the total volume of the rejected spaces (thereby maximizing the amount of useful space) so that more boxes can potentially be loaded. We would also like to minimize the amount of fragmentation of the residual space, i.e., we prefer to have large useful spaces. Therefore, we calculate the fitness of the above two ways of generating residual space and select the one with the higher fitness value:

$$\text{fitness} = \text{vol. of largest useful space} - \text{total vol. of rejected space}. \quad (11)$$

Secondly, we modified the evaluation function that ranks a particular loading configuration by taking into account the amount of rejected space generated. The original GRMOD heuristic only considered the volume utilization, which is equivalent to greedily selecting the locally optimal arrangement. However, there are arrangements with high utilization that produces a large amount of rejected space, which negatively affects the overall utilization, resulting in a solution that is globally suboptimal. Therefore, we used the following evaluation function:

$$\text{eval} = \text{utilization} - \text{volume lost}, \quad (12)$$

where *volume lost* is the total volume of rejected space in percentage terms. For example, for the arrangement illustrated in the left diagram of Fig. 2, if $c1$ is rejected space, then the evaluation for that arrangement would be:

$$\text{eval} = b/(b + c1 + c2 + c3) - c1/(b + c1 + c2 + c3).$$

For each container type C_i, we create a sequence of SCLP instances and repeatedly apply the above GRASP algorithm to solve the instances. As boxes are loaded, they are removed from subsequent problems in the sequence. Each of the generated packing patterns is added into the set P for use in the solution of the parameterized set cover IP formulation.

This technique takes advantage of the non-deterministic nature of GRASP to produce a variety of packing patterns. These generated patterns can potentially make use of any combination of boxes, unlike the patterns generated by the G4BP strategy that consist of only one type of box.

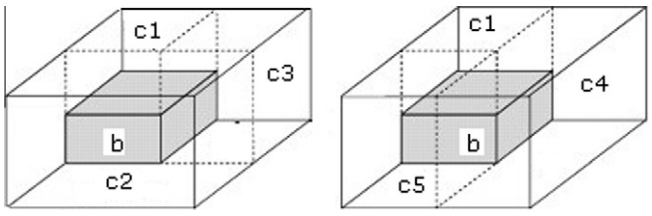

Fig. 2. Two ways of generating residual spaces.

4.3. Application of GRASP on Combinations Strategy

In the Application of GRASP on Combinations (C-GRASP) Strategy, we wish to generate packing patterns that load exactly r, $2 \leqslant r \leqslant N$ types of boxes for each type of container using the above GRASP algorithm, which serves to diversify our set of packing patterns to include different numbers of types of boxes. Since it is not feasible to generate packing plans for all $\binom{N}{r}$ combinations for large N, we only generate up to $R = 5$ packing patterns using a probabilistic sampling strategy for each container C_i and each value of r, $2 \leqslant r \leqslant N$.

We illustrate our procedure using the following MCLCMP instance. There are two container types C_1 and C_2 with corresponding volumes $V_1 = 100$ and $V_2 = 50$. There are three types of boxes B_1, B_2, and B_3, with $n_1 = 15$, $n_2 = 20$, and $n_3 = 25$ boxes of each type available. The volume of each box type is $v_1 = 10$, $v_2 = 15$, and $v_3 = 20$. For container C_1 and $r = 2$:

1. Enumerate all possible combinations of r box types and place them into *B_Set*. In our example, *B_Set* = {(B_1,B_2), (B_2,B_3), (B_1,B_3)}.
2. For each combination in *B_Set*, assign a probability value in proportion to the total volume of all combinations (see Table 1).
3. Choose a combination in *B_Set* according to the probabilities, and remove it from *B_Set*. Assume in our example that (B_1,B_3) is chosen.
4. Create an instance of SCLP with only boxes in the chosen combination, where the number of boxes of each type is proportional to the total volume of the selected box types, rounded down. In our example, the number of boxes n'_1 of type B_1 is $\left\lfloor \frac{V_1}{v_1} \cdot (v_1 \cdot n_1) \Big/ (v_1 \cdot n_1 + v_3 \cdot n_3) \right\rfloor = \lfloor \frac{100}{10} \times 150/(150 + 500) \rfloor = 1$. Similarly, the number of boxes n'_3 of type B_3 is $\lfloor \frac{100}{20} \times 500/(150 + 500) \rfloor = 3$. If the rounded-down value for a box type becomes 0, we set it to 1.
5. Generate a packing pattern for the SCLP instance using the GRASP algorithm and add this pattern to P. Assume that the GRASP algorithm successfully finds a packing pattern that loads 1 box of type B_1 and 4 boxes of type B_3 into the container.
6. Subtract the boxes used by the packing pattern from the original problem. In our example, we set $n_1 = 14$ and $n_3 = 21$.
7. Go back to Step 2, until $R = 5$ packing patterns are generated or all combinations are exhausted.

For the special case of $r = N$ (i.e., all box types are considered), we always generate R packing patterns. For each container type, we generate up to R packing patterns for each value of $r = 2, 3, \ldots, N$. Since there are M types of containers, the above procedure generates up to $M \times (N - 1) \times R$ packing patterns. For our implementation, we set $R = 5$.

By devising the SCLP instances such that the number of boxes of each type is proportional to the total volume of such remaining boxes, the C-GRASP strategy generates packing patterns that uses a number of each box type that follows this proportion. Furthermore, the total volume of all boxes in the instance is likely to be close to volume of the container. The premise behind this approach is, should a good packing pattern be found, then it can be repeatedly used to load most of the boxes of the chosen types.

Table 1
Probability of combination selection.

Combination	Total box volume ($\sum v_i \cdot n_i$)	Probability
(B_1,B_2)	$10 \times 15 + 15 \times 20 = 450$	0.281
(B_2,B_3)	$15 \times 20 + 20 \times 25 = 500$	0.313
(B_1,B_3)	$10 \times 15 + 20 \times 25 = 650$	0.406

5. Binary search on loading factor

The idea behind our technique is to solve SC(a) in order to provide an estimate of the number of containers needed to guarantee successful loading of a percentage of the input boxes. We then attempt to insert the remaining boxes into the unused capacity of selected containers using the subroutine *Insert*. If we are able to successfully load all boxes, then we have found a solution to the original MCLCMP instance and we reduce the loading factor a; otherwise, we increase a. Since the cost of the optimal solution of SC(a) is a non-decreasing function of a, we employ a binary search procedure that terminates after *max_iter* iterations to find the lowest possible value of a efficiently (for our implementation, we set *max_iter*=8). See Algorithm 1.

Algorithm 1. SCBS

1. Generate the set of packing patterns P
2. $a_l \leftarrow 0.01, a_h \leftarrow 1.0, a \leftarrow a_l$, $iter \leftarrow 0$
3. **while** $iter < max\_iter$ **do**
4. $B \leftarrow$ set of boxes to be loaded
5. Solve SC (a); let S be the list of selected packing patterns sorted by space utilization in descending order
6. **for all** packing patterns $s \in S$ **do**
7. Load a container with boxes based on s
8. Remove loaded boxes from B
9. **end for**
10. $C_m \leftarrow$ containers loaded with multiple box types, sorted by space utilization in descending order
11. $C_s \leftarrow$ containers loaded with only one box type, sorted by space utilization in descending order
12. If B is not empty, then invoke *Insert*(C_m, P, B)
13. If B is not empty, then invoke *Insert*(C_s, P, B)
14. If B is not empty, then $a_l \leftarrow a$; else $a_h \leftarrow a$ and record the solution
15. $a \leftarrow (a_h + a_l)/2, iter \leftarrow i + 1$
16. **end while**

A set of packing patterns P is first generated using the three strategies described in Section 4. We then perform binary search on the range $[a_l, a_h]$ for the loading factor a, with initial values $a_l = 0.01$ and $a_h = 1.0$.

Next, we solve the model using a standard IP solver. The solution has a corresponding list of selected packing patterns S, which we sort by space utilization in descending order. We then load containers with boxes according to the patterns in S. At the end of this process, there may be boxes left over. If so, we invoke the *Insert* subroutine, first on containers with multiple box types, then on containers containing only one type of box. Finally, we update the binary search range accordingly.

Algorithm 2. Insert (C,P,B)

1. **for** each loaded container $c \in C$ **do**
2. Let B_c be the set of boxes loaded into c
3. Form an instance of SCLP with container c and boxes $B \cup B_c$
4. Invoke GRASP to solve this instance; let p be the resulting packing pattern, and let B_p be the set of boxes in p
5. Reload the container c according to p
6. $B \leftarrow B - B_p$
7. $P \leftarrow P \cup p$
8. If B is empty, then break
9. **end for**

The aim of the *Insert* subroutine is to attempt to load the remaining boxes into the selected containers; the pseudocode is given in Algorithm 2. In effect, given the set of leftover boxes, the subroutine unloads each container in turn (chosen in descending order of space utilization) and uses the GRASP algorithm to reload the container inclusive of the leftover boxes. Over the course of this process, new packing patterns might be discovered, which we add to our set of packing patterns P.

The *Insert* subroutine potentially helps to improve solution quality in two ways. Firstly, the unloading-and-reloading process might result in a solution that loads all boxes into the containers, which corresponds to a feasible loading plan with potentially higher utilization than the original set cover formulation. Secondly, the side effect of introducing new packing patterns into P could improve the solutions generated in later iterations.

6. Test instance generation

Ivancic et al. (1989) proposed 64 instances for the multiple container loading problem: 47 instances use only 1 type of container, and 17 instances use 2–3 types of containers. Currently, the 47 instances with a single container type are commonly used to compare the performance of MCLP approaches. Unfortunately, we were unable to locate the 17 instances with multiple container types despite searching online repositories and contacting the authors.

One drawback of these instances is that the optimal solutions are not known. Although the authors provided straightforward lower bounds on the solutions for these instances, the tightness of the bounds are unknown. In this section, we propose a method to generate instances for the MCLCMP with known optimal solutions. The same method can also be used to generate test data with known optimal solutions for the single container loading problem (SCLP).

Our technique begins by partitioning the containers into rectangular sub-regions; each of these sub-regions can be viewed as a separate container that contains only boxes of one type. We then make use of the G4BP strategy to find packing patterns for these sub-regions that use only one type of box and with very high utilization (>99%). Finally, we recombine the patterns for each sub-region to form packing patterns for the original containers, which can be used to form a provably optimal solution to an MCLCMP instance. We ensure the optimality of this solution by solving a linear integer program (given by Eqs. (13)–(15) and explained later in this section), which verifies that no other solution has a better utilization.

Algorithm 3. GenPattern (c,T)

1: Initialize list of packing patterns $Q \leftarrow \emptyset$
2: **for** each box type $B = (l,w,h)$, l in $[l_l, l_h]$, w in $[w_l, w_h]$, h in $[h_l, h_h]$ **do**
3: Skip if $max(l,w,h)/min(l,w,h) > T$
4: Form an instance of SCLP with container c and $\lceil V_c/(l \cdot w \cdot h) \rceil$ boxes of type B
5: Invoke G4BP on this instance; let p be the resulting packing pattern
6: **if** volume utilization of p is greater than 99% **then**
7: $Q \leftarrow Q \cup p$
8: **end if**
9: **end for**
10: Sort Q by volume of box type; discard the last 50% of Q
11: Randomly return $p' \in Q$

We first describe our *GenPattern* algorithm for finding packing patterns for SCLP instances with homogeneous boxes that have a utilization greater than 99%, given in Algorithm 3. For a given sub-region denoted by container c with volume V_c, we invoke the G4BP strategy on SCLP instances with boxes of integer dimensions $l \times w \times h$, where the ranges for l,w, and h are $[l_l,l_h]$, $[w_l,w_h]$, and $[h_l,h_h]$, respectively. However, we discard the box types that are long and thin, i.e., the ratio between the largest and smallest dimensions of the box is greater than a threshold T (line 3). We retain the packing patterns that have utilization greater than 99%. At the end of the algorithm, we discard the half of the patterns that are composed of boxes with smaller volume. This is because the existence of several small boxes allows small residual spaces to be easily filled, thereby reducing the difficulty of the problem. Finally, we randomly select one of the remaining patterns as the output to this algorithm.

We set the ranges for l, w, and h of the box types to be [27,98], [19,170], and [17,67], respectively. These values were obtained by inspecting the dimensions of 16 types of household appliances that our client commonly handles, e.g., television sets, air-conditioning units and refrigerators. We set the ratio threshold T to be 7.5, which approximately corresponds to the largest dimension ratio of the 16 considered appliances (a DVD player).

Three types of container are commonly used in the sea freight industry, called the *20-foot* (20') container, the *40-foot* (40') container, and the *40-foot high cube* (40'h) container. We use these three types of containers to create instances of the MCLCMP. Table 2 lists the specifications for each container. The dimensions are given in centimetres for better granularity.

Note that although larger containers are more expensive, they offer a comparatively lower cost per volume. Therefore, we generated the following three sets of test cases. Although the problem specification for all instances allow the use of all three types of container, Type 1 instances have known optimal solutions that utilize only 40' high cube containers, Type 2 instances use 40' and 40' high cube containers, and Type 3 instances require all three types of containers for the known optimal solution. For each test set, we generate seven divisions of instances with 3–9 types of boxes, respectively.

Type 1 instances (40'h): Our generation process is as follows. For a test instance with d box types, we arbitrarily partition the 40'h container into d sub-regions. For each partition π_i, $i = 1, 2, \ldots, d$, we invoke the *GenPattern* subroutine to find a packing pattern for that partition using n_i boxes of type B_i with utilization greater than 99%. The combination of all d packing plans is therefore a valid packing plan for the 40'h container with total utilization greater than 99%. We then randomly select an integer $K \in [3,11]$ and multiply the number of boxes of each type by K. As a result, we have now generated a MCLCMP instance with three container types and $K \times n_i$ boxes of type B_i with a known solution that uses K 40'h containers. We now show that the known solution is optimal if $K \leqslant 11$.

Assume an optimal solution S for this instance uses a_1 20-foot containers, a_2 40-foot containers, and a_3 40-foot high cube containers. We first show that the cost of S is not lower than the optimal solution of the following minimization problem:

Table 2
Container types.

Container type	Volume	Length	Width	Height	Cost (USD)	Cost/cm^3
20'	V_1	590	235	239	900	2.71E−5
40'	V_2	1203	235	239	1700	2.51E−5
40' high cube	V_3	1203	235	270	1800	2.36E−5

$$\mathbf{OP}(\boldsymbol{K}): \; z = \min 900x_1 + 1700x_2 + 1800x_3, \tag{13}$$

$$\text{s.t. } V_1x_1 + V_2x_2 + V_3x_3 \geqslant 0.99KV_3, \tag{14}$$

$$x_1, x_2, x_3 \geqslant 0 \quad \text{and integer}. \tag{15}$$

Let the total volume of boxes loaded into 20', 40', and 40'h containers in the solution S be A, B, and C respectively, and the total volume of all boxes in the problem instance be V. Since S is a feasible solution of the instance, the sum of the volume of boxes in all containers is the same as the total volume of all boxes (Eq. 16), and the total volume of containers of each type is large enough to accommodate the boxes inside them (inequalities (17)).

$$A + B + C = V, \tag{16}$$

$$V_1a_1 \geqslant A, V_2a_2 \geqslant B, V_3a_3 \geqslant C. \tag{17}$$

Let u be the utilization of a 40'h container in our known solution. The total volume of boxes in it is therefore V_3u, and the total volume of all boxes in K containers is $K\,V_3u$. Since the known solution is a feasible solution, the total volume of all boxes in K containers is the same as the total volume of all boxes (Eq. (18)).

$$KV_3u = V. \tag{18}$$

With Eqs. (16)–(18), we have:

$$V_1a_1 + V_2a_2 + V_3a_3 \geqslant A + B + C = V = KV_3u \geqslant 0.99KV_3. \tag{19}$$

Hence, $x_1 = a_1$, $x_2 = a_2$, and $x_3 = a_3$ is a feasible solution to the model OP (K), which implies that the cost of optimal solution S is not lower than the cost of the optimal solution of the model OP (K).

By using a commercial IP solver to solve OP(K), it can be verified that the optimal solution of the model is 1800K for $K \leqslant 11$, which means that the cost of any optimal solution to the instance must be at least 1800 K. Since the cost of the known solution that uses K 40'h containers is 1800K, it is optimal. We generated 15 test instances for each value of $d \in [3,9]$ for a total of 105 Type 1 test instances.

Note that while we constructed the partitions of our containers by hand in these test instances, the partitions can be completely arbitrary as long as there are d different types of sub-regions, and a pattern with greater than 99% volume utilization can be found for each sub-region. Furthermore, utilization values other than 99% can be used, and the same process can be used to find the valid combinations for the number of each container type that ensures the optimality of the solution. However, if the utilization value is too small, then there may not exist a value of K that ensures optimality. Note that it would also be undesirable to set the utilization value to 100% (a perfect packing) because such instances are known to be easier to solve: for every 2D perfect packing, there is a permutation of the boxes (rectangles) that yields that perfect packing using the bottom-left heuristic Lesh et al. (2004, Theorem 1, pp. 9). The same argument applies to 3D perfect packings, so in the worst case we can enumerate the $O(n!)$ solutions to find the optimal. In contrast, the best known solution encoding scheme for general 2D packing utilizes a sequence-pair Murata et al., 1996, which implies that the enumeration of $O((n!)^2)$ solutions may be required to find the optimal in the worst case.

Type 2 instances (40' + 40'h): We use a similar technique as for Type 1 instances, but we use both 40' and 40'h containers. Note that the partitions for each box type need not be a single rectangular parallelepiped. It can instead be composed of multiple such rectangular sub-regions as long as the total utilization for the entire partition exceeds 99%. For example, to generate an instance with three types of boxes, we created three partitions for both the 40' and 40'h containers, as shown in Fig. 3. Each partition can consist of multiple identical rectangular sub-regions. By invoking the *GenPattern* subroutine on single sub-regions, which returns a packing pattern with greater than 99% utilization, we ensure that

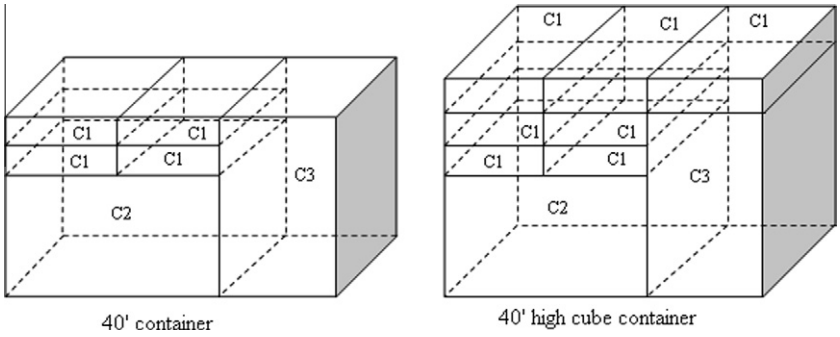

Fig. 3. Partitions for two containers, $d = 3$.

Table 3
Valid combinations of $K_{40'}$ and $K_{40'h}$ to ensure known solution optimality.

$K_{40'}$	Range of $K_{40'h}$
4	1, 2,...,7
3	1, 2,...,8
2	1, 2,...,9
1	1, 2,...,10

Table 4
Valid combinations of $K_{20'}$; $K_{40'}$; and $K_{40'h}$ to ensure known solution optimality.

$K_{20'}$	$K_{40'}$	range of $K_{40'h}$
1	3	1, 2,...,5
1	2	1, 2,...,9
1	1	1, 2,...,10

the known solution obtained has utilization greater than 99% for all partitions.

Let $K_{40'}$ and $K_{40'h}$ be the number of 40' and 40'h containers that we wish to include in the known optimal solution, respectively. Using an argument similar to the Type 1 test instances, we can verify that any of the values for $K_{40'}$ and $K_{40'h}$ given in each row of Table 3 ensures that the known solution with the corresponding number of containers is optimal. For each value of $d \in [3,9]$, we generated 20 test cases by randomly selecting a valid combination of values for $K_{40'}$ and $K_{40'h}$, for a total of 140 Type 2 instances.

Type 3 instances (20' + 40' + 40'h): This test set includes all three container types in the known optimal solutions, generated in a similar manner. Table 4 gives the combinations of values for $K_{20'}$,$K_{40'}$, and $K_{40'h}$ that ensures the optimality of the known solution. We generated 15 test cases for each value of $d \in [3,9]$ for a total of 105 Type 3 test instances.

We have therefore generated 50 test instances for each value of $d \in [3,9]$ for a total of 350 test instances. They are grouped into 7 test sets of 50 instances each, labeled *mtc3, mtc4,...,mtc9*, where the digit corresponds to the value of *d*. The new instances, the known optimal solutions, and detailed experiment results are located at http://www.zhuwb.com/mclcmp, along with the partitions we used for each test case.

7. Results and analysis

Our SCBS was implemented as a sequential algorithm in Java, and no multi-threading was explicitly utilized. It was executed on a HP server equipped with one Intel Xeon E5430 CPU clocked at 2.66 GHz, 8 GB RAM and running CentOS 5.4. Sun Microsystem's JDK 1.6.0 update 22 and ILog CPlex 11.0 were installed. Both Java and CPlex are 32-bit editions.

A set of experiments were conducted to find the best parameters for our SCBS (Appendix B in the online supplement); the results reported in this section used the following settings. All three strategies for generating packing patterns G4BP, S-GRASP and C-GRASP were used. Both S-GRASP and C-GRASP use GRASP as the underlying solver to solve the SCLP, and we fixed the random seed so that our version of GRASP is essentially deterministic. For C-GRASP, the parameter *R* was fixed to 5.

7.1. Results on Bin-Packing Instances

Ivancic et al. (1989) proposed 47 instances of the bin-packing problem, which has since become a standard benchmark test set for multiple container loading problems. Only one container type is used, and the objective is to minimize the number of containers used to pack all the boxes. This can be considered a special case of MCLCMP with the cost for the single container type $c_1 = 1$. We compared our SCBS approach with the following algorithms:

- **IMM**: an integer programming based approach by Ivancic et al. (1989); the result is extracted from Bischoff and Ratcliff (1995a).
- **BO**: the approach by Bortfeldt (2000).
- **seq**: the sequential version of the approach by Eley (2002).
- **sim**: the simultaneous version of the approach by Eley (2002).
- **bot**: the set cover based bottleneck approach of Eley (2003).
- **IC**: the Iterated Construction approach by Lim and Zhang (2005).
- **SA**: the meta-heuristic (local search,SA) approach by Takahara et al. (2006).
- **MS**: the multi-start local search approach by Takahara et al. (26).

For each of the 47 instances, we executed our SCBS five times using random seed 3, 5, 7, 9, and 11, respectively. The results corresponding to various seeds are very similar (the total number of bins for 47 instances are 693, 693, 691, 691 and 692 respectively), we only report the detailed results corresponding to seed 11 in Table 5 due to space limitations. The detailed results corresponding to other seeds can be downloaded from http://www.zhuwb.com/mclcmp.

In Table 5, for each instance, we provide the number of box types (column *bt*) and the total number of boxes (column *boxes*, the number of boxes of each type is given in Table A.9 in the online supplement), as well as the lower bound (column *lb*) given by Ivancic et al. (1989). The values stated for each technique is the number of containers required for the solution generated. Techniques marked with an asterisk (*) handle the supporting area constraint, and therefore will tend to produce poorer results than techniques that do not handle this constraint. The last column *Time(s)* reports

Table 5
Results on bin-packing problems IMM01–47, SCBS with Seed 11.

No.	bt	Boxes	lb	IMM	BO	seq*	sim*	bot*	IC	SA	MS	SCBS	Time (s)
1	2	70	19	26	25	27	26	25	25	25	25	25	12
2	2	70	7	11	10	11	10	10	10	10	10	10	14
3	4	180	19	20	20	21	22	20	19	20	20	19	98
4	4	180	26	27	28	29	30	26	26	26	26	26	63
5	4	180	46	65	51	55	51	51	51	51	51	51	143
6	3	103	10	10	10	10	10	10	10	10	10	10	21
7	3	103	16	16	16	16	16	16	16	16	16	16	17
8	3	103	4	5	4	4	4	4	4	4	4	4	22
9	2	110	18	19	19	19	19	19	19	19	19	19	19
10	2	110	47	55	55	55	55	55	55	55	55	55	24
11	2	110	16	18	18	17	18	17	16	16	16	16	30
12	3	95	45	55	53	53	53	53	53	53	53	53	33
13	3	95	22	27	25	25	25	25	25	25	25	25	42
14	3	95	28	28	28	27	27	27	27	27	27	27	61
15	3	95	11	11	11	12	12	11	11	11	11	11	21
16	3	95	21	34	26	28	26	26	26	26	26	26	29
17	3	95	7	8	7	8	7	7	7	7	7	7	29
18	3	47	2	3	2	2	2	2	2	2	2	2	6
19	3	47	3	3	3	3	3	3	3	3	3	3	14
20	3	47	4	5	5	5	5	5	5	5	5	5	29
21	5	95	17	24	21	24	26	20	20	20	20	20	119
22	5	95	8	10	9	9	9	8	9	9	9	8	107
23	5	95	17	21	20	21	21	20	20	20	20	**19**	96
24	4	72	5	6	6	6	6	6	5	5	5	5	52
25	4	72	4	6	5	6	5	5	5	5	5	5	41
26	4	72	3	3	3	3	3	3	3	3	3	3	20
27	3	95	4	5	5	5	5	5	5	5	5	5	49
28	3	95	9	10	10	11	10	10	9	10	10	**10**	51
29	4	118	15	18	17	18	18	17	17	17	17	17	86
30	4	118	18	24	22	22	23	22	22	22	22	22	100
31	4	118	11	13	13	13	14	13	12	13	13	12	70
32	3	90	4	5	4	4	4	4	4	4	4	4	28
33	3	90	4	5	5	5	5	5	4	5	5	4	16
34	3	90	7	9	8	8	9	8	8	8	8	8	32
35	2	84	3	3	2	2	2	2	2	2	2	2	11
36	2	84	11	18	14	18	14	14	14	14	14	14	21
37	3	102	12	26	23	26	23	23	23	23	23	23	31
38	3	102	26	50	45	46	45	45	45	45	45	45	33
39	3	102	12	16	15	15	15	15	15	15	15	15	35
40	4	85	7	9	9	9	9	8	9	9	9	8	62
41	4	85	14	16	15	16	15	15	15	15	15	15	65
42	3	90	4	4	4	4	4	4	4	4	4	4	28
43	3	90	3	3	3	3	3	3	3	3	3	3	21
44	3	90	3	4	3	4	4	4	3	3	3	3	29
45	4	99	2	3	3	3	3	3	3	3	3	3	105
46	4	99	2	2	2	2	2	2	2	2	2	2	70
47	4	99	3	4	3	3	3	3	3	3	3	3	54
Total			599	763	705	733	721	699	694	698	698	**692**	
Avg. time (s)				–	–	–	–	30	6.43	2	1		45.94

the running time in seconds of our SCBS algorithm for each instance. The last row gives the average running time in seconds reported by the corresponding publications (a dash '-' indicates that it was not reported).

For 45 out of the 47 instances, our result is the same as the best results in literature. For instance No. 23, SCBS improves the best known result by 1, while it is worse than the best known result by 1 for instance 28. Over all test instances, the total number of bins required by SCBS (692) is the least out of all the approaches tested. In particular, it uses 7 fewer bins than the original set cover based bottleneck approach given by column *bot*, which is an indication of the effectiveness of our modifications to the IP formulation and packing pattern generation strategies.

Although the SCBS algorithm was developed for the MCLCMP rather than the specific bin-packing problem variant, it is able to slightly improve on existing techniques for the bin-packing problem. Importantly, it is unclear how these existing algorithms can be adapted to the MCLCMP, where the costs for multiple container types are different.

7.2. Results on Generated MCLCMP Instances

Since our SCBS has a stochastic component, we executed SCBS three times for each instance of MCLCMP, and we fixed the initial seed used by the pseudorandom number generator to 3, 11 and 5 for the three executions.

Table 6 provides the results for the SCBS approach on the 350 newly generated MCLCMP instances. The columns under the heading *Instance Characteristics* provides the information about the instances. The columns under the headings *seed = 3, seed = 11* and *seed = 5* present the results for SCBS when the pseudorandom number generator is initialized with seed 3, 11 and 5, respectively. The new instances are divided into seven sets *mtc3- mtc9* of 50 instances each (*mtc* stands for "multiple-container test case" and the suffix gives the number of box types in each instance in that set). The *#box* column gives the average number of boxes in the problem set. The *optimal* column shows the average optimal cost of the 50 instances. The *opt.util* column provides the average volume utilization of the containers in the optimal solutions. The *gap*

Table 6
Results of SCBS on new generated MCLCMP instances.

Instance characteristics				Seed = 3			Seed = 11			Seed = 5		
Set	#Box	Optimal	opt.util (%)	gap (%)	vol.util (%)	t (s)	Gap (%)	vol.util (%)	t (s)	Gap (%)	vol.util (%)	t (s)
mtc3	1828	11,704	99.30	5.45	95.24	727	5.42	95.28	717	5.47	95.35	729
mtc4	1871	12,234	99.26	5.43	94.50	1282	5.30	94.56	1272	5.53%	94.35	1307
mtc5	1890	12,548	99.29	6.78	93.65	2040	6.76	93.63	1976	6.82	93.61	1993
mtc6	1930	12,282	99.28	7.66	92.78	3139	7.65	92.90	3060	7.67	93.08	3065
mtc7	2136	13,496	99.33	6.94	93.17	4303	7.08	93.18	4337	7.01	93.08	4381
mtc8	1734	11,540	99.31	8.81	91.81	5126	9.13	91.72	5149	8.91	91.70	5128
mtc9	1886	12,572	99.34	9.00	91.59	6782	9.06	91.62	6871	9.42	91.49	6696
Average			99.30	7.15	93.25	3343	7.20	93.27	3340	7.26	93.24	3328

columns give the average percentage difference between the SCBS solution and the optimal solution, computed as (*cost* − *optimal cost*)/ *optimal cost*. The *vol. util* columns provide the average volume utilization of the containers in the solutions found by SCBS. The columns *t (s)* give the average time taken by SCBS in CPU seconds. Finally, the last row of the table provides the average over 350 instances.

The results show that even though our SCBS has a stochastic component, the overall performance on any of the 7 sets of instances is stable regardless of the random seed used. Hence, we base the following analysis on the results obtained using seed 3. The results show that the average gap between the SCBS approach and the known optimal solution over all 7 sets of instances is 7.15%, and this gap increases as the number of box types increases. This is not a small difference, which shows that although SCBS outperforms existing algorithms for the standard bin-packing test problems, it is only moderately successful when solving MCLCMP instances. Also note that the average volume utilization of the SCBS solutions is about 93.25% for test instances with 8 or 9 box types even though the optimal solution has a volume utilization of over 99.30%. It is reasonable to say that the MCLCMP is a difficult problem that deserves further research.

Table 7
Effectiveness of loading factor.

Set	Seed = 3				Seed = 11				Seed = 5			
	#	%	impr (%)	impr2 (%)	#	%	impr (%)	impr2 (%)	#	%	impr (%)	impr2 (%)
mtc3	30	60.00	1.19	2.21	31	62.00	1.18	2.04	30	60.00	1.13	1.88
mtc4	30	60.00	2.15	2.82	33	66.00	2.19	2.67	31	62.00	1.98	2.67
mtc5	36	72.00	2.11	2.45	31	62.00	2.06	2.52	33	66.00	2.12	2.52
mtc6	41	82.00	2.10	2.63	41	82.00	2.06	2.34	45	90.00	2.06	2.40
mtc7	42	84.00	2.36	2.94	42	84.00	2.21	2.77	42	84.00	2.34	2.72
mtc8	44	88.00	2.40	3.33	45	90.00	2.23	2.86	47	94.00	2.48	3.18
mtc9	48	96.00	2.08	2.74	46	92.00	2.01	2.52	46	92.00	1.80	2.37
Avg.	38.7	77.43	2.06	2.73	38.4	76.86	1.99	2.53	39.1	78.29	1.99	2.53

Table 8
Average frequency of various strategies used by SCBS.

S	Set	Tot t (s)	G4BP			S-GRASP			C-GRASP		
			t (%)	# pat (%)	t (s)	t (%)	# pat (%)	t (s)	t (%)	# pat (%)	
3	mtc3	727.0	89.6	12.32	9.0	122.0	16.78	13.9	285.9	39.33	41.3
	mtc4	1282.0	139.4	10.87	12.0	176.9	13.80	18.6	610.4	47.61	60.9
	mtc5	2040.0	130.5	6.39	15.0	217.9	10.68	23.0	1055.1	51.72	79.7
	mtc6	3139.0	191.7	6.11	18.0	284.1	9.05	25.1	1530.4	48.75	93.2
	mtc7	4303.0	316.1	7.35	21.0	373.9	8.69	28.8	2157.7	50.14	111.7
	mtc8	5126.0	237.5	4.63	24.0	407.2	7.94	26.8	2409.3	47.00	120.5
	mtc9	6782.0	232.3	3.42	27.0	523.8	7.72	29.6	3179.1	46.88	138.7
	Avg	3342.7	191.0	7.30	18.0	300.8	10.67	23.7	1604.0	47.35	92.3
11	mtc3	717.0	89.4	12.45	9.0	121.9	17.31	13.9	285.3	40.58	41.3
	mtc4	1272.0	139.3	11.03	12.0	176.8	13.95	18.6	608.1	48.18	60.9
	mtc5	1976.0	130.4	6.56	15.0	217.9	10.90	23.0	1048.8	53.27	79.7
	mtc6	3060.0	191.6	6.24	18.0	283.6	9.19	25.1	1535.1	50.00	93.5
	mtc7	4337.0	316.1	7.14	21.0	374.2	8.70	28.8	2150.0	50.00	111.1
	mtc8	5149.0	237.4	4.51	24.0	407.7	7.93	26.8	2405.2	47.00	120.9
	mtc9	6871.0	232.3	3.37	27.0	524.6	7.76	29.6	3189.4	46.94	138.8
	avg	3340.3	190.9	7.33	18.0	300.9	10.82	23.7	1603.1	47.99	92.3
5	mtc3	729.0	89.4	12.33	9.0	121.6	16.91	13.9	284.8	40.02	41.3
	mtc4	1307.0	139.3	10.59	12.0	177.0	13.71	18.6	610.6	47.53	61.0
	mtc5	1993.0	130.5	6.44	15.0	217.7	10.86	23.0	1055.7	53.39	79.9
	mtc6	3065.0	191.7	6.20	18.0	284.5	9.26	25.1	1540.3	50.17	93.4
	mtc7	4381.0	316.1	7.05	21.0	374.8	8.71	28.8	2169.2	50.17	112.0
	mtc8	5128.0	237.5	4.53	24.0	408.2	7.97	26.8	2408.6	47.47	121.0
	mtc9	6696.0	232.2	3.48	27.0	524.8	7.90	29.6	3194.3	47.94	138.6
	avg	3328.4	191.0	7.23	18.0	301.2	10.76	23.7	1609.1	48.10	92.4

Table 7 examines the effect of parameterizing the set cover formulation. If SCBS stops with $a < 1$, we can conclude that the introduction of the loading factor has helped to find a better solution. The number of instances improved by binary search is given in columns #. We can see that over 60% of instances are improved by binary search (column%), regardless of the random seed used. For each instance, the improvement due to binary search is calculated as the gap to optimal before binary search less the gap to optimal after binary search. For a set of instances, the average improvement is given in column *impr*. If we only consider instances that are actually improved by binary search, the average improvement to the quality of the solutions is given in column *impr2* for each test set.

For all seven sets of instances except mtc3, the introduction of binary search has reduced the average gap to optimal by about 2% (column *impr*) regardless of the random seed used. This reduction is a significant contribution to the performance of SCBS since without the binary search phase the average gap to optimal would increase from 7% to 9%.

Table 8 reports the frequency of the various pattern generation strategies employed by our SCBS algorithm. All figures reported are averaged over 50 instances in the test set. Since we executed SCBS three times with random seed set to 3, 11 and 5, the results are organized into three groups of rows by random seed. The column *S* indicates the random seed used. The column *tot t (s)* gives the total time in CPU seconds used by SCBS. The columns under the headings *G4BP*, *S-GRASP*, and *C-GRASP* provide the statistics for the G4-heuristic bin-packing strategy (Section 4.1), the Sequential Application of GRASP Strategy (Section 4.2) and the Application of GRASP on Combinations Strategy (Section 4.3), respectively. For each strategy, the column *t (s)* reports the total CPU seconds spent on the strategy over the entire execution of SCBS, while the column *t (%)* gives this value as a percentage of total CPU time spent. The column *# pat* provides the number of patterns generated by the strategy.

8. Conclusion

This paper studies the multiple container loading cost minimization problem, which has the practical application of modeling the process whereby the buying agent for a multi-national retailer redistributes products after inspection. We added a loading factor parameter to the set cover formulation by Eley (2003) to exploit the excess capacity of the chosen containers inherent in the formulation. Combined with three new strategies to find efficient packing patterns, a binary search on the loading factor produces a solution that is superior to the original model.

Our SCBS approach is designed to handle multiple containers, and makes use of SCLP algorithms to do so. As new and better SCLP approaches are developed, they can be incorporated into the SCBS approach in order to find good packing patterns. The type of constraints that can be added to the basic MCLCMP is dependent on the constraints that the SCLP sub-components can handle. For example, although the S-GRASP and C-GRASP strategy can handle the supporting area constraint, the G4BP strategy is unable to do so. If we replace the G4BP strategy with a technique that can handle the supporting area constraint (or remove it entirely), then SCBS will be able to handle this constraint for the MCLCMP.

Our approach can also handle the case where there are m_t containers of type t. Let $C(i)$ be the container filled using packing plan i. We simply add the following equation to the linear integer program SC (a):

$$\sum_{C(i)=t} x_i \leqslant m_t, \quad \forall t. \tag{20}$$

This modification also allows us to handle the case where the price offered by different carriers is different for containers of the same size: containers with the same dimensions but with different prices are considered to be of different types. Note that when there are a limited number of containers, then feasibility could become an issue should the total capacity of all containers be close to the total volume of all items. However, we have found that this scenario does not occur in the practical problem instances faced by our client, and in fact our assumption of unlimited containers is a reasonable approximation of the real situation.

We also proposed a technique to generate instances of MCLCMP with known optimal solutions, and used this technique to generate a 350-instance test suite with 3–9 box types using three standard shipping container sizes. The box dimensions are based on the dimensions of common household appliances. The existence of a known optimal solution for these sets of test data allows a more accurate measure of the performance of various approaches than using a theoretical lower bound. This technique can also be used to generate test data for other variants of container loading problems.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2011.04.017.

References

Bischoff, E.E., Ratcliff, M.S.W., 1995a. Issues in the development of approaches to container loading. OMEGA the International Journal of Management Science 23, 377–390.

Bischoff, E.E., Ratcliff, M.S.W., 1995b. Loading multiple pallets. Journal of the Operational Research Society 46, 1322–1336.

Bortfeldt, A., 2000. A heuristic for multiple container loading problems. OR Spectrum 22, 239–261.

Christensen, S.G., Rousøe, D.M., 2009. Container loading with multi-drop constraints. International Transactions in Operational Research 16, 727–743.

Crainic, T.G., Perboli, G., Tadei, R., 2008. Extreme point-based heuristics for three-dimensional bin packing. INFORMS Journal on Computing 20, 368–384.

Crainic, T.G., Perboli, G., Tadei, R., 2009. TS^{2}PACK: A two-level tabu search for the three-dimensional bin packing problem. European Journal of Operational Research 195, 744–760.

Dyckhoff, H., Finke, U., 1992. Cutting and Packing in Production and Distribution: A Typology and Bibliography. Contributions to Management Science, first ed. Physica-Verlag, Heidelberg, German.

Eley, M., 2002. Solving container loading problems by block arrangement. European Journal of Operational Research 141, 393–409.

Eley, M., 2003. A bottleneck assignment approach to the multiple container loading problem. OR Spectrum 25, 45–60.

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

Fekete, S.P., van der Veen, J.C., 2007. PackLib^2: An integrated library of multi-dimensional packing problems. European Journal of Operational Research 183, 1131–1135.

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

Ivancic, N., Mathur, K., Mohanty, B.B., 1989. An integer programming based heuristic approach to the three-dimensional packing problem. Journal of Manufacturing and Operations Management 2, 268–298.

Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M., 2004. Exhaustive approaches to 2d rectangular perfect packings. Information Processing Letters 90, 7–14.

Lim, A., Zhang, X., 2005. The container loading problem. In: SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing. ACM, New York, NY, USA.

Lins, L., Lins, S., Morabito, R., 2002. An n-tet graph approach for non-guillotine packings of n-dimensional boxes into an n-container. European Journal of Operational Research 141, 421–439.

Lodi, A., Martello, S., Vigo, D., 2002. Heuristic algorithms for the three-dimensional bin packing problem. European Journal of Operational Research 141, 410–420.

Martello, S., Pisinger, D., Vigo, D., 2000. The three-dimensional bin packing problem. Operations Research 48, 256–267.

Monaci, M., Toth, P., 2006. A set-covering-based heuristic approach for bin-packing problems. INFORMS Journal on Computing 18, 71–85.

Moura, A., Oliveira, J.F., 2005. A grasp approach to the container-loading problem. IEEE Intelligent Systems 20, 50–57.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996. Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 1518–1524.

Parreño, F., Alvarez-Valdes, R., Oliveira, J.F., Tamarit, J.M., 2008. A hybrid GRASP/VND algorithm for two-and-three-dimensional bin packing. Annals of Operations Research 179, 203–220.

Scheithauer, G., Terno, J., 1996. The G4-heuristic for the pallet loading problem. Journal of the Operational Research Society 47, 511–522.

Takahara, S., 2006. A simple meta-heuristic approach for the multiple container loading problem, in: SMC '06: IEEE International Conference on Systems, Man and Cybernetics, vol. 3, 2006, pp. 2328–2333.

Takahara, S., 2008. A multi-start local search approach to the multiple container loading problem, in: W. Bednorz (Ed.), Advances in Greedy Algorithms, I-TECH Education and Publishing, Zieglergasse 14, 1070 Vienna Austria, European Union: IN-TECH, pp. 55–68.

Takahara, S., Miyamoto, S., 2005. An evolutionary approach for the multiple container loading problem, in: HIS '05: Proceedings of the Fifth International Conference on Hybrid Intelligent Systems, pp. 227–232.

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

Verweij, B., 1996. Multiple Destination Bin Packing, Technical Report CS-1996-39, Department of Information and Computing Sciences, Faculty of Science, Utrecht University 3508 TC Utrecht, Netherlands.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. European Journal of Operational Research 183, 1109–1130.