Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

A modified wall-building-based compound approach for the knapsack container loading problem

Ampol Karoonsoontawong * **and Krongthong Heebkhoksung**

Department of Civil Engineering, Faculty of Engineering, King Mongkut's University of Technology Thonburi, 126 Pracha Uthit Road, Bang Mod, Thung Khru, Bangkok 10140, Thailand

* Corresponding author, email: ampolk@gmail.com, ampol.kar@kmutt.ac.th

Received: 27 January 2014 / Accepted: 25 March 2015 / Published: 3 April 2015

Abstract: This paper considers the knapsack container loading problem. Rectangular-shaped boxes with different sizes are packed in rectangular-shaped containers according to a specified container sequence. Each container is filled such that the total loaded volume is maximised. All boxes with the same origin-destination pair may be rotated in six orthogonal directions without load-related and positioning constraints. The proposed approach performs 36 wall-building heuristic methods based on three ranking functions, two priority rules and six orthogonal rotations of containers. Three real-world test problems from a furniture company are employed. There is not a winning heuristic that performs best on the three test problems. The typical wall-building approach does not perform well when compared with considering all six orthogonal rotations of a container. In terms of the number of containers, the proposed approach can save up to 33% on the three problems, and the highest fill percentages in the best solutions are improved by up to 70.96% when compared with the manual solutions. In an additional experiment the proposed approach yields slightly lower (up to 5.38% and 3.64%) average fill percentages, with 94.47% and 87.94% shorter CPU time than the existing tree-search heuristic for the respective weakly and strongly heterogeneous problem instances.

Keywords: heuristic algorithm, knapsack container loading problem, wall-building algorithm

INTRODUCTION

Container loading is a crucial function for an efficient supply chain [1]. Inefficient container loading may inevitably result in additional container costs as well as an unsatisfactory level of customer service. The problem considered in this paper is the knapsack container loading problem.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

The problem definition is explicitly stated as follows. Given the user-specified rectangular-shaped containers and rectangular-shaped boxes, all boxes are packed into the containers in a specified sequence. To fill a container, the subset of boxes is selected to be packed in the container such that the total volume is maximised. The cargo boxes may be rotated in any orthogonal direction without load-related and positioning constraints. It is noted that in principle, the empty spaces could be filled up with foam rubber to ensure proper support of the boxes [2]. All boxes have the same origin-destination pair. It is assumed that the cargo weights are dominated by cargo volume in container packing, so box weights are not considered in the algorithm.

In our view the contributions of this paper are threefold. First, a modified wall-building-based compound approach is proposed in this paper. The modification of the original compound approach [3] is the consideration of six orthogonal rotations of the container in addition to the three existing ranking functions and two existing priority rules for determining layer depths and strip heights. The modification results in 36 modified wall-building heuristic methods. The compound approach performs the 36 heuristic methods while recording the best solution found. Second, three real-world test problems from a furniture company in Thailand are employed in the comparison of the proposed compound method against the manual solutions and the tree-search heuristic [2] in terms of three criteria: the number of containers, the fill percentage, and the CPU time. Lastly, a performance comparison experiment of the proposed compound approach and the tree-search heuristic is conducted on the weakly heterogeneous and strongly heterogeneous standard test problem instances with a single container in terms of the fill percentage and the CPU time.

LITERATURE REVIEW

The container loading problem was first studied by Gilmore and Gomory [4]. Dyckhoff [5] and Wäscher et al. [6] proposed the general classification of cutting and packing problems. Pisinger [2] categorised the packing and loading literature into four categories based on the objective function and side constraints: strip packing, knapsack container loading, bin packing and multi-container loading. Firstly, in the strip packing problem (e.g. [3, 7]) the container has a known width and height but unlimited depth, and the problem is to pack all boxes such that the container's depth is minimised. This problem category is applicable to multi-drop situations. Secondly, in the knapsack container loading problem (e.g. [8–11]) we select a subset of boxes to be packed in a single container such that the total box volume is maximised. Thirdly, in the bin packing problem (e.g. [12]) all boxes have to be packed into a minimum number of containers with fixed dimensions. Lastly, in the multi-container loading problem (e.g. [13, 14]) all boxes are packed into a minimum number of containers, which are chosen from containers with varying dimensions such that the total shipping cost is minimised. Since the container loading problem is a nondeterministic-polynomial- hard problem [2], there does not exist an efficient algorithm for obtaining the exact solution in polynomial time. Christensen and Rousøe [15] and Bortfeldt and Wäscher [1] provide a thorough review of heuristics for the container loading problem. The heuristics for the container loading problem can be subdivided into three categories: construction algorithms [3, 4, 7, 16], tree-search algorithms [2, 17, 18], and metaheuristic algorithms [19–25]. In this study the knapsack container loading problem is considered, and the proposed approach is a construction algorithm.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

MODIFIED WALL-BUILDING-BASED COMPOUND APPROACH

The wall-building algorithm of George and Robinson [16] fills a single container by building layers (walls) across the container's depth. The layer depth is selected based on the rationale that a box with the largest size of the smallest dimensions may be difficult to accommodate later in the packing procedure. As such, the ranking rule sets the layer's depth equal to the largest size of the smallest dimensions of the unpacked boxes. Given a known layer's depth, horizontal strips are built across the container's height. To fill a horizontal strip, the algorithm inserts the box with the largest size of the smallest dimensions of an unpacked box. Bischoff and Marriott [3] proposed a compound approach that performs the wall-building algorithms with various ranking rules while recording the best solution found. In this paper we modify the compound approach by considering six orthogonal rotations of the container, Pisinger's three ranking functions and two priority rules [2], Pisinger's box pairing procedure [2], and Pisinger's dynamic programming-based algorithm for the exact solution of the 0-1 knapsack strip packing problem [26].

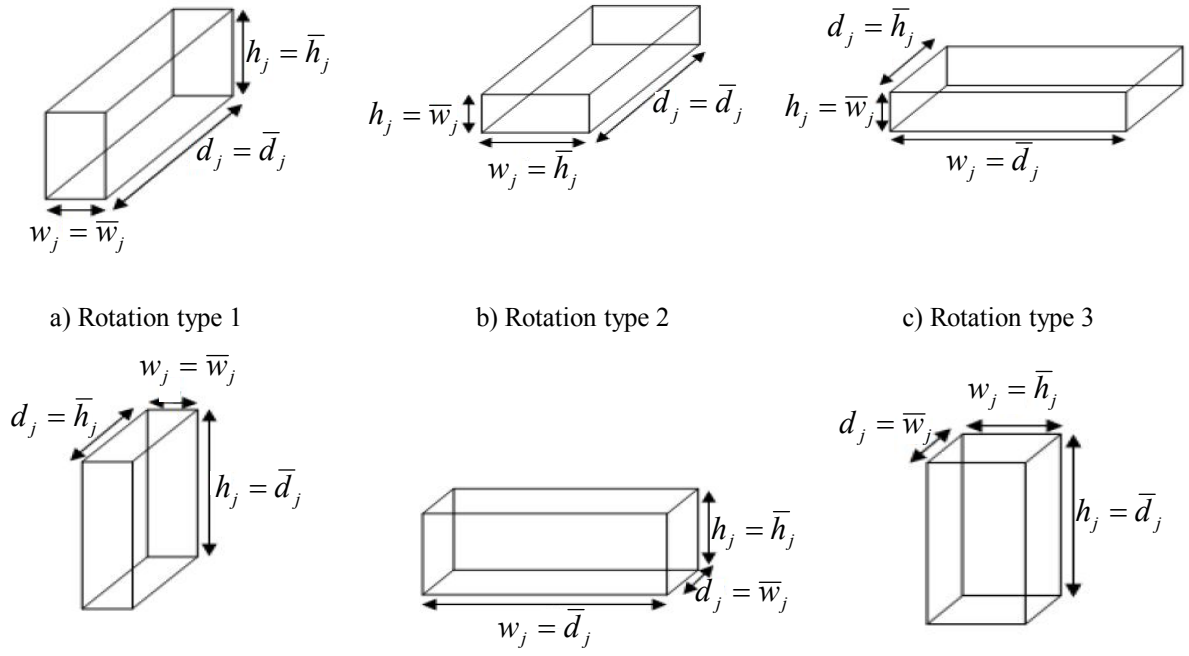
The dimensions $\overline{W}$, $\overline{H}$ and $\overline{D}$ refer to the typical width, height and depth of the container, whereas the current width, height and depth of container (W, H and D) refer to the dimensions considered in the modified wall-building algorithm along the x-, y- and z-axes respectively. In the same way the dimensions $\overline{w}_j$, $\overline{h}_j$ and $\overline{d}_j$ are the initial width, height and depth of box j, whereas the current dimensions of box j (w_j, h_j and d_j) refer to the dimensions considered in the algorithm along the three axes. All boxes can be rotated in six orthogonal directions (or rotations), as illustrated in Figure 1.

The modified wall-building-based compound approach further considers six possible container rotations in the procedure. As shown in Table 1, the container rotation types 1, 2, 5 and 6 correspond to wall-building approaches, whereas the container rotation types 3 and 4 correspond to floor-building approaches. Container rotation type 1 (i.e. typical container rotation) builds layers (walls) across the container's depth $\overline{D}$ and builds horizontal strips of length $\overline{W}$ across the container's height $\overline{H}$. For container rotation types 2–6, the procedure rotates the container in the other five orthogonal directions and performs the wall-building algorithm; these rotation types correspond to either wall or floor building and either horizontal- or vertical-strip building.

Specifically, container rotation type 2 corresponds to building layers (walls) across container's depth $\overline{D}$ and vertical strips of length $\overline{H}$ across container's width $\overline{W}$. Container rotation type 3 corresponds to building layers (floors) across container's height $\overline{H}$ and horizontal strips of length $\overline{D}$ across container's width $\overline{W}$. Container rotation type 4 corresponds to building layers (floors) across container's height $\overline{H}$ and horizontal strips of length $\overline{W}$ across container's depth $\overline{D}$. Container rotation type 5 corresponds to building layers (walls) across container's width $\overline{W}$ and horizontal strips of length $\overline{D}$ across container's height $\overline{H}$. Container rotation type 6 corresponds to building layers (walls) across container's width $\overline{W}$ and vertical strips of length $\overline{H}$ across container's depth $\overline{D}$.

The notations used in the proposed procedure, including the parameters and variables, are first given. Then the pseudo-code is described, followed by descriptions of the major components in the pseudo-code.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8



a) Rotation type 1 b) Rotation type 2 c) Rotation type 3

d) Rotation type 4 e) Rotation type 5 f) Rotation type 6

Figure 1. Six orthogonal rotations

Table 1. Container rotation types and associated descriptions

Container rotation type	W	H	D	Description
1	$\overline{W}$	$\overline{H}$	$\overline{D}$	Wall building: Layer building across depth $\overline{D}$ Horizontal strip (strip length = $\overline{W}$) building across height $\overline{H}$
2	$\overline{H}$	$\overline{W}$	$\overline{D}$	Wall building: Layer building across depth $\overline{D}$ Vertical strip (strip length = $\overline{H}$) building across width $\overline{W}$
3	$\overline{D}$	$\overline{W}$	$\overline{H}$	Floor building: Layer building across height $\overline{H}$ Horizontal strip (strip length = $\overline{D}$) building across width $\overline{W}$
4	$\overline{W}$	$\overline{D}$	$\overline{H}$	Floor building: Layer building across height $\overline{H}$ Horizontal strip (strip length = $\overline{W}$) across depth $\overline{D}$
5	$\overline{D}$	$\overline{H}$	$\overline{W}$	Wall building: Layer building across width $\overline{W}$ Horizontal strip (strip length = $\overline{D}$) across height $\overline{H}$
6	$\overline{H}$	$\overline{D}$	$\overline{W}$	Wall building: Layer building across width $\overline{W}$ Vertical strip (strip length = $\overline{H}$) building across depth $\overline{D}$

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

Notations

Parameters

OC = the user-specified ordered set of containers
$N = \{1,\ldots,n\}$ = set of boxes
$\overline{W}$, $\overline{H}$ and $\overline{D}$ = typical container width, height and depth
$\overline{w}_j$, $\overline{h}_j$ and $\overline{d}_j$ = initial width, height and depth of box j

Variables

N' = set of unloaded boxes
N'' = set of unloaded boxes that are feasible to fill the current layer
N''' = set of unloaded boxes that are feasible to fill the current strip
W = current container width in the algorithm
H = current container height in the algorithm
D = current container depth in the algorithm
w_j, h_j and d_j = current width, height and depth of box j
D_r = current residual container depth
H_r = current residual wall height
d' = current layer depth
sw' = current strip width
sh' = current strip height
strip_start_y = y-coordinate of the beginning of current strip
strip_end_y = y-coordinate of the ending of current strip
wall_start_z = z-coordinate of the beginning of current layer
wall_end_z = z-coordinate of the ending of current layer
(x_j, y_j, z_j) = (x,y,z)-coordinate of the referenced corner of loaded box j in the current solution
(dx_j, dy_j, dz_j) = (width, height, depth) of loaded box j in the current solution
(x_j^*, y_j^*, z_j^*) = (x,y,z)-coordinate of the referenced corner of loaded box j in the best solution found (dx_j^*, dy_j^*, dz_j^*) = (width, height, depth) of loaded box j in the best solution found

Pseudo-code

For each heuristic method c,u,v (i.e. container rotation type c, ranking function u and priority rule v), the following steps are performed, given the ordered set of containers OC. The variable indicating the ordinal number of container in OC is denoted by *con*.

Step 0: Set *con* = 1. Set $N' = N$.

Step 1: Initialise the following for the container *con*:

- set the current residual container depth to the current container depth: set $D_r = D$
- set the current strip width to the current container width: set $sw' = W$
- set *wall_end_z* = 0
- set $(x_j, y_j, z_j) = (0,0,0)$ for all boxes j in N'
- set $(dx_j, dy_j, dz_j) = (\overline{w}_j, \overline{h}_j, \overline{d}_j)$ for all boxes j in N'.

Step 2: Determine the current layer depth d' based on the ranking function f^u and the priority rule v and D_r. If d' can be determined:

- update the current residual container depth: set $D_r = D_r - d'$
- set the current residual wall height to the current container height: set $H_r = H$
- set *wall_start_z* = *wall_end_z*
- set *wall_end_z* = *wall_start_z* + d'
- set *strip_end_y* = 0

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

Otherwise, go to Step 7.
Step 3: Perform the box pairing procedure to obtain the set of feasible unloaded boxes (N'') to fill the current wall. Update (dx_j, dy_j, dz_j) for all rotated boxes j.
Step 4: Determine the current strip height (sh') based on the ranking function f^{α} and the priority rule v, given H_r. If sh' can be determined:

- update the current residual wall height: set $H_r = H_r - sh'$
- determine the set of feasible unloaded boxes (N''') to fill the current strip, and update (dx_j, dy_j, dz_j) for all rotated boxes j
- set *strip_start_y* = *strip_end_y*
- set *strip_end_y* = *strip_start_y* + sh'

Otherwise, go to Step 2.
Step 5: Perform the strip packing procedure to select boxes from N''' to fill the strip ($sw' \times sh' \times d'$) and update (x_j, y_j, z_j) and (dx_j, dy_j, dz_j) of each loaded box j. Update the sets of unloaded boxes N' and N''.
Step 6: If there is an unloaded box (i.e. $N' \neq \{\}$), go to Step 4. Otherwise, go to Step 8.
Step 7: Calculate the fill percentage of the current container *con*: fill percentage = volume of loaded boxes / volume of container. If *con* < |*OC*|, then set *con* = *con* + 1 and go to Step 1. Otherwise, go to Step 8.
Step 8: Calculate the fill percentage of the current container *con*.
Step 9: If the higher fill% in the current solution is higher than that in the best solution:

- set best heuristic method (c^*,u^*,v^*) = current heuristic method (c,u,v)
- set (x_j^*, y_j^*, z_j^*) = (x_j, y_j, z_j) and set (dx_j^*, dy_j^*, dz_j^*) = (dx_j, dy_j, dz_j) for all j in N.

Descriptions of Major Components

The major components of the proposed approach include the layer depth and strip height determinations, the box pairing procedure and the strip packing problem.

Layer depth and strip height determinations

In Step 2 we employ the ranking functions used by Pisinger [2], which are based on certain statistics of the dimensions of the unloaded boxes. The smallest and largest dimensions of the unloaded boxes are denoted by α and β respectively. Three different ranking functions are considered:

$$f_k^1 = \sum_{i=1}^{n} 1_{(w_i=k \vee h_i=k \vee d_i=k)} \quad \forall k = \alpha, \alpha+1, \ldots, \beta-1, \beta \tag{1.1}$$

$$f_k^2 = \sum_{i=1}^{n} 1_{(\max\{w_i,h_i,d_i\}=k)} \quad \forall k = \alpha, \alpha+1, \ldots, \beta-1, \beta \tag{1.2}$$

$$f_k^3 = \sum_{i=1}^{n} 1_{(\min\{w_i,h_i,d_i\}=k)} \quad \forall k = \alpha, \alpha+1, \ldots, \beta-1, \beta \tag{1.3}$$

Type-1 ranking function, Eq. (1.1), determines the number of occurrences of each dimension from all dimensions w_i, h_i and d_i of the remaining boxes. Type-2 ranking function, Eq. (1.2), determines the number of occurrences of each dimension from the largest dimensions of the unloaded boxes. Type-3 ranking function, Eq. (1.3), determines the number of occurrences of each dimension from the smallest dimensions of the unloaded boxes.

In Step 4 when the layer depth and the set of feasible unloaded boxes (N'') to fill the current wall have been determined, the ranking functions only consider the current width and the current

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

height of feasible unloaded boxes. The smallest dimension of the current widths and heights of the feasible unloaded boxes is denoted by α and the largest by β. Three different ranking functions are:

$$f_k^1 = \sum_{i=1}^{n} 1_{(w_i=k \vee h_i=k)} \quad \forall k = \alpha, \alpha+1, ..., \beta-1, \beta \tag{2.1}$$

$$f_k^2 = \sum_{i=1}^{n} 1_{(\max\{w_i,h_i\}=k)} \quad \forall k = \alpha, \alpha+1, ..., \beta-1, \beta \tag{2.2}$$

$$f_k^3 = \sum_{i=1}^{n} 1_{(\min\{w_i,h_i\}=k)} \quad \forall k = \alpha, \alpha+1, ..., \beta-1, \beta \tag{2.3}$$

The type-1 ranking function, Eq. (2.1), determines the number of occurrences of each dimension from all dimensions w_i and h_i of the feasible unloaded boxes. The type-2 ranking function, Eq. (2.2), determines the number of occurrences of each dimension from the largest dimensions of w_i and h_i of the feasible unloaded boxes. The type-3 ranking function, Eq. (2.3), determines the number of occurrences of each dimension from the smallest dimensions w_i and h_i of the feasible unloaded boxes.

In Steps 2 and 4 we consider two priority rules [2]. For priority rule 1, the largest dimension with positive ranking function value is selected, i.e. largest dimension k with $f_k > 0$. For priority rule 2, the most frequent dimension is selected, i.e. dimension k with the largest value of f_k. The motivation of priority rule 1 is that the largest dimension should be loaded early in the packing procedure; otherwise, it may be difficult to be packed later. The motivation of priority rule 2 is that a homogeneous layer or strip width may be tightly packed. It is noted that the procedure by George and Robinson [16] is equivalent to type-3 ranking function and priority rule 1.

Box pairing procedure

After the layer depth d' is determined, in Step 3 we employ the box pairing procedure [2] to determine the set of feasible unloaded boxes (N'') to fill the current wall. Pisinger [2] indicated that a box pairing procedure can be used to achieve an improved solution in his tree-search heuristic, and this is also adopted in our proposed algorithm. The complexity of the box pairing procedure is $O(n^2)$ and it is executed only once for each layer depth d'. The box pairing procedure is described below:

Step 3.1: Set $N''=\{\}$. If the smallest dimension of each box i in the set N' is bigger than the layer depth d', box i is not inserted in N'' and is not considered in the box pairing procedure. Otherwise, rotate each box i in the set N' such that its depth d_i is the largest dimension satisfying the constraint $d_i \le d'$. The filling ratio to pack box i in a layer with depth d' is $\mu(i)$:

$$\mu(i) = w_i h_i d_i / w_i h_i d' = d_i / d' \tag{3}$$

Step 3.2: Pair box i with another box j in the set N' where $j \neq i$. All orthogonal rotations of i and j are considered such that $d_i + d_j \le d'$, and the associated filling ratio, $\eta(i,j)$, is determined:

$$\eta(i,j) = \frac{w_i h_i d_i + w_j h_j d_j}{d' \cdot \max\{w_i, w_j\} \cdot \max\{h_i, h_j\}} \tag{4}$$

If $\eta(i,j) \le \mu(i)$ for all boxes $j \neq i$ in N' and all orthogonal rotations of boxes i and j, then box i remains alone and is inserted in N''. Otherwise, box j and the corresponding rotation with the largest value of $\eta(i,j)$ is selected to pair with box i in order to form a new box k with the following dimensions: $w_k = \max\{w_i, w_j\}$, $h_k = \max\{h_i, h_j\}$ and $d_k = d_i + d_j$. Then box k is inserted in N''.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

Strip packing problem

In Step 5 strips are filled horizontally. The strip has a width equal to the current container width $sw' = W$, a depth equal to the current layer depth d', and a height of sh'. The procedure first determines the set of feasible unloaded boxes (N''') to fill the current strip as follows. Set $N''' = \{\}$. Each box j in the set N'' is rotated in one of six directions such that w_j is minimised subject to $d_j \le d'$ and $h_j \le sh'$. If it is possible to fit box j in the current strip, box j is inserted in the set N'''. If not, then box j is not considered for the current strip packing. The strip packing problem can be formulated as a 0-1 knapsack problem, as shown below [2]:

$$\max \sum_{j \in N'''} w_j \cdot h_j \cdot d_j \cdot s_j \quad \text{subject to} \quad \sum_{j \in N'''} w_j \cdot s_j \le W \quad \text{and} \quad s_j \in \{0,1\} \quad \forall j \in N'''$$

where s_j is a binary decision variable; s_j = 1 if box j is chosen to fill the current strip, and 0 otherwise. The 0-1 knapsack problem is a nondeterministic-polynomial-hard problem [27], so there does not exist an efficient algorithm for an exact solution in polynomial time. It can be solved in pseudo-polynomial time by dynamic programming [28]. In this study we employ the effective dynamic programming-based algorithm by Pisinger [26].

COMPUTATIONAL EXPERIENCES

Thirty-six modified wall-building heuristic methods were implemented in C by modifying the callable C code: the box pairing procedure was taken from the callable C code by Pisinger [29] and the dynamic programming heuristic for a 0-1 knapsack problem was taken from the callable C code by Pisinger [30]. These heuristic trials were run on a computer with a 1.73 GHz Intel Core i7 processor and 4 GB of RAM, running under Windows 7. We used three real-world test problems from a furniture company in Thailand. The origin of the cargos was Thailand and the destinations of the cargos in the three test problems were Brunei, Vietnam and Japan. The initial dimensions of the boxes for the three test problems are shown in Tables 2a-c. These three test problems were weakly heterogeneous since the number of box types was less than 20 [2]. The standard container types were 40' HQ, 40' and 20', as shown in Table 2d. For the first and second test problems, all heuristic methods employed 40' HQ containers. For the third test problem, all methods employed a 40' container as the first container and a 20' container as the second container.

Due to space limitation, we show only the results of test problem 1 in Table 3. It is noted that the cargo weights in each container in the solutions do not exceed the allowable weight. The heuristic method *c,u,v* refers to the container rotation type *c*, ranking function f^u and priority rule *v*. There is not a winning heuristic that performs best on the three test problems. On the first test problem, there are 11 heuristic methods that yield two 40' HQ containers and 25 methods that yield three 40' HQ containers. The method *c*=6,*u*=2,*v*=2 performs best on the first test problem with two 40' HQ containers and the highest fill percentage (87.82%) of container number 1. On the second test problem there are 34 heuristic methods that yield two 40' containers and only 2 methods (*c*=3,*u*=2,*v*=1 and *c*=5,*u*=2,*v*=1) that yield three 40' containers. The method *c*=6,*u*=3,*v*=1 performs best on the second test problem with two 40'-containers and the highest fill percentage (80.75%) of container number 1. In the best solution found, the fill percentage (20.50%) of container number 2 is equal to 15,908,120 cm^3, which can fill a 20' container. Thus, the best solution found becomes a 40' container with 80.75% fill and a 20' container with 41.03% fill. On the third test problem there are 33 methods that yield two containers (40' and 20'). The three methods (*c*=1,*u*=1,*v*=2; *c*=3,*u*=1,*v*=2; *c*=4,*u*=1,*v*=2) perform best on the third test problem, yielding a single 40' container with the highest

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

fill percentage (68.42%). Interestingly, the typical wall-building algorithms that are associated with container rotation type 1 (c=1) do not perform well; thus, this reiterates the improvement by considering the six orthogonal rotations of container.

The best solutions found on the three test problems by the proposed compound approach were subsequently compared with the manual solutions by the furniture company as well as the best solutions found by the tree-search heuristic [2], as shown in Table 4. The manual solutions by the furniture company employ three 40' HQ containers for the first test problem, two 40' containers for the second test problem, and a 40' container and a 20' container for the third test problem. In terms of the number of containers, the proposed compound approach can save 33.33%, 25% and 33.33% on the three test problems respectively. The highest fill percentages in the best solutions found are

Table 2. Initial dimensions of boxes for three test problems and dimensions of standard containers

a) Initial dimensions (cm) of boxes for test problem 1 (223 boxes and 16 box types)*

Box type	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Width ($\overline{w}_j$)	100	70	96	100	85	85	85	102	80	212	80	102	102	145	70	69
Depth ($\overline{d}_j$)	100	70	70	100	190	148	195	160	155	80	170	102	102	67	68	69
Height ($\overline{h}_j$)	80	80	53	60	100	100	100	90	90	90	65	77	57	83	100	92
Number of boxes	1	3	2	3	3	4	3	3	3	4	3	6	1	16	11	157

* Total box volume = 129,402,900 cm^3

b) Initial dimensions (cm) of boxes for test problem 2 (113 boxes and 14 box types)*

Box type	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Width ($\overline{w}_j$)	90	92	110	100	80	100	110	110	110	100	100	100	100	70
Depth ($\overline{d}_j$)	100	142	110	100	95	195	110	110	110	210	210	180	180	73
Height ($\overline{h}_j$)	101	101	75	115	92	70	90	55	45	55	60	60	26	92
Number of boxes	12	1	8	4	5	3	1	1	1	1	9	1	1	65

* Total box volume = 78,578,264 cm^3

c) Initial dimensions (cm) of boxes for test problem 3 (94 boxes and 11 box types)*

Box type	1	2	3	4	5	6	7	8	9	10	11
Width ($\overline{w}_j$)	120	120	120	91	130	90	130	110	80	110	70
Depth ($\overline{d}_j$)	120	120	190	210	244	170	170	190	80	210	73
Height ($\overline{h}_j$)	70	40	70	70	80	70	70	45	70	60	92
Number of boxes	1	1	1	1	1	2	2	1	3	1	80

* Total box volume = 53,097,700 cm^3

d) Typical dimensions of standard container types

Container type	$\overline{W}$	$\overline{H}$	$\overline{D}$
40' HQ	243 cm (8 ft)	292 cm (9.6 ft)	1219 cm (40 ft)
40'	243 cm (8 ft)	262 cm (8.6 ft)	1219 cm (40 ft)
20'	243 cm (8 ft)	262 cm (8.6 ft)	609 cm (20 ft)

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

Table 3. Computational results for test problem 1 (all containers are 40' HQ with 86,495,364 cm^3)

Heuristic method *c,u,v* (CPU time) t	Container no.	Fill %
1, 1, 1 (3.88 sec)	1	77.03
	2	72.13
	3	0.45
2, 1, 1 (3.23 sec)	1	70.7
	2	77.04
	3	1.87
3, 1, 1 (0.39 sec)	1	55.39
	2	80.7
	3	13.51
4, 1, 1 (4.79 sec)	1	76.97
	2	72.64
5, 1, 1 (8.73 sec)	1	61.42
	2	53.81
	3	34.37
6, 1, 1 (1.06 sec)	1	67.5
	2	66.69
	3	15.42
1, 1, 2 (0.61 sec)	1	81.11
	2	68.49
2, 1, 2 (0.46 sec)	1	72.95
	2	73.67
	3	2.99
3, 1, 2 (0.10 sec)	1	79.04
	2	67.24
	3	3.33
4, 1, 2 (0.39 sec)	1	81.28
	2	68.33
5, 1, 2 (0.28 sec)	1	81.48
	2	64.3
	3	3.83
6, 1, 2 (1.38 sec)	1	81.9
	2	63.88
	3	3.83
1, 2, 1 (0.48 sec)	1	77.03
	2	72.13
	3	0.45
2, 2, 1 (0.55 sec)	1	71.36
	2	77.8
	3	0.45
3, 2, 1 (0.23 sec)	1	55.39
	2	80.7
	3	13.51
4, 2, 1 (6.19 sec)	1	76.97
	2	72.64
5, 2, 1 (8.64 sec)	1	61.42
	2	53.81
	3	34.37
6, 2, 1 (0.27 sec)	1	58.84
	2	70.58
	3	20.19
1, 2, 2 (0.45 sec)	1	79.51
	2	70.1
2, 2, 2 (0.37 sec)	1	79.5
	2	69.65
	3	0.45
3, 2, 2 (0.09 sec)	1	78.17
	2	52.61
	3	18.82
4, 2, 2 (0.31 sec)	1	77.99
	2	24.86
	3	46.76
5, 2, 2 (0.10 sec)	1	86.13
	2	59.94
	3	3.53
6, 2, 2 (0.22 sec)	**1**	**87.82**
	2	**61.78**
1, 3, 1 (1.58 sec)	1	78.24
	2	71.37
2, 3, 1 (0.63 sec)	1	68.99
	2	62.42
	3	18.19
3, 3, 1 (0.36 sec)	1	79.66
	2	69.95
4, 3, 1 (2.39 sec)	1	79.15
	2	70.46
5, 3, 1 (0.35 sec)	1	86.5
	2	63.11
6, 3, 1 (4.03 sec)	1	82.4
	2	67.21
1, 3, 2 (0.89 sec)	1	73.21
	2	68.15
	3	8.25
2, 3, 2 (0.60 sec)	1	80.21
	2	66.83
	3	2.56
3, 3, 2 (0.12 sec)	1	79.52
	2	66.06
	3	4.02
4, 3, 2 (0.57 sec)	1	68.75
	2	71.12
	3	9.73
5, 3, 2 (3.57 sec)	1	79.92
	2	61.28
	3	8.41
6, 3, 2 (15.85 sec)	1	78.91
	2	63.83
	3	6.87

Note: *c* = container rotation type; *u* = ranking function; *v* = priority rule
Total CPU time = 74.14 sec; the best solution found is in bold and underlined

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

increased by 70.96%, 57.32% and 27.94% on test problems 1, 2 and 3 respectively when compared with the manual solutions. The tree-search heuristic yields the same results as the proposed compound approach in terms of the number of containers. The highest fill percentages by the proposed approach are only 2.47% and 8.57% lower than those by the tree-search heuristic on problems 1 and 2 respectively. The computational times by the proposed approach are only 74.49% and 98.10% less than those by the tree-search heuristic on problems 1 and 2 respectively.

Next, an experiment was conducted to compare the performance of the proposed approach and the tree-search heuristic [2] for the knapsack container loading problem. We employed only one container in the user-specified ordered set of containers (i.e. $|OC| = 1$) so that the proposed approach

Table 4. Comparison between manual solutions, best solutions found by tree-search heuristic, and best solutions found by proposed approach on three test problems

Container no.	Manual solution		Best solution by tree-search heuristic [2]		Best solution found by modified wall-building-based compound approach	
	Container type	No. of boxes Vol. of boxes Fill %	Container type	No. of boxes Vol. of boxes Fill %	Container type	No. of boxes Vol. of boxes Fill %
Test problem 1						
1	40' HQ	82 44,440,856 cm^3 51.37%	40' HQ	111 77,884,495 cm^3 90.04%	40' HQ	162 75,962,452 cm^3 87.82%
2	40' HQ	94 41,173,128 cm^3 47.60%	40' HQ	112 51,518,405 cm^3 59.56%	40' HQ	61 53,440,448 cm^3 61.78%
3	40' HQ	47 43,788,916 cm^3 50.63%	-		-	
Total CPU time	-		290.58 sec		74.14 sec	
Test problem 2						
1	40'	39 39,838,464 cm^3 51.33%	40'	93 68,541,604 cm^3 88.32%	40'	98 62,670,144 cm^3 80.75%
2	40'	74 38,739,800 cm^3 49.92%	20'	20 10,036,660 cm^3 25.89%	20'	15 15,908,120 cm^3 41.03%
Total CPU time	-		809.80 sec		15.38 sec	
Test problem 3						
1	40'	85 41,508,600 cm^3 53.48%	40'	94 53,097,700 cm^3 68.42%	40'	94 53,097,700 cm^3 68.42%
2	20'	9 11,589,100 cm^3 29.89%	-		-	
Total CPU time	-		0.04 sec		9.65 sec	

Note: Volume of 40' HQ container = 86,495,364 cm^3
Volume of 40' container = 77,608,854 cm^3
Volume of 20' container = 38,772,594 cm^3

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

solved the knapsack container loading problem. As in Pisinger [2], data instances were randomly generated using the scheme of Hemminki [31], which reflects typical properties of industrial loading problems. The container had width, height and depth of 230, 230 and 590 cm respectively. Two types of data instances were tested: the weakly heterogeneous and the strongly heterogeneous. For weakly heterogeneous instances, 20 different box types were generated, with the widths, heights and depths randomly distributed between 25-115 cm. Then the cargo was generated by randomly choosing a box as one of the 20 box types. For strongly heterogeneous instances, boxes were generated such that they may have different dimensions. For each instance, new boxes were generated until their overall volume exceeded the target volume percentage of the container volume (T_c). A series of tests were run where boxes were generated until the overall volume of the boxes exceeded T_c = 80, 85, 90, 95, 100, 110, 120, …, 200% of the container volume. These were run on a computer with a 3.4 GHz Intel Core i7-3770 processor and 8 GB of RAM, running under Windows 7. Table 5 shows the average results of 100 problem instances at each T_c level by the tree-search heuristic and the proposed approach for the weakly and strongly heterogeneous instances. For the

Table 5. Comparison of the modified wall-building-based compound approach and the tree search heuristic for the knapsack container loading problem (average values of 100 instances)

T_c	Weakly heterogeneous instances					Strongly heterogeneous instances				
	Average total number of boxes	Tree search heuristic [2]		Proposed approach		Average total number of boxes	Tree search heuristic [2]		Proposed approach	
		Avg. % volume filled	Avg. CPU time (sec)	Avg. % volume filled (% change)	Avg. CPU time (sec) (% change)		Avg. % volume filled	Avg. CPU time (sec)	Avg. % volume filled (% change)	Avg. CPU time (sec) (% change)
90	84.76	90.74	2.06	86.89 (-4.24)	1.34 (-34.95)	82.45	90.86	0.86	88.36 (-2.75)	0.90 (4.65)
95	89.34	92.75	26.05	87.76 (-5.38)	1.44 (-94.47)	86.78	92.35	2.6	89.15 (-3.47)	0.95 (-63.46)
100	94.09	93.34	24.06	88.45 (-5.24)	1.4 (-94.18)	91.27	93.07	3.45	89.76 (-3.56)	0.98 (-71.59)
110	103.28	93.96	20.29	89.37 (-4.89)	1.53 (-92.46)	100.23	93.99	5.24	90.57 (-3.64)	1.10 (-79.01)
120	113.03	94.28	16.38	89.95 (-4.59)	1.62 (-90.11)	109.54	94.59	7.17	91.22 (-3.56)	1.20 (-83.26)
130	122.56	94.46	13.66	90.72 (-3.96)	1.81 (-86.75)	118.4	95	8.78	91.91 (-3.25)	1.27 (-85.54)
140	131.73	94.61	11.13	90.91 (-3.91)	1.86 (-83.29)	127.36	95.3	10.49	92.35 (-3.10)	1.37 (-86.94)
150	140.66	94.66	10.5	91.32 (-3.53)	1.91 (-81.81)	136.76	95.54	12.11	92.73 (-2.94)	1.46 (-87.94)
160	149.78	94.79	10.41	91.72 (-3.24)	2.05 (-80.31)	145.91	95.74	12.19	93.08 (-2.78)	1.56 (-87.20)
170	159	94.87	10.18	92.02 (-3.00)	2.17 (-78.68)	155.28	95.91	12	93.46 (-2.55)	1.65 (-86.25)
180	168.27	94.97	9.85	92.37 (-2.74)	2.27 (-76.95)	164.53	96.04	11.08	93.64 (-2.50)	1.71 (-84.57)
190	177.57	95	10.55	92.52 (-2.61)	2.39 (-77.35)	173.74	96.14	10.06	93.89 (-2.34)	1.77 (-82.41)
200	186.84	95.08	10.35	92.75 (-2.45)	2.53 (-75.56)	182.78	96.18	9.46	94.05 (-2.21)	1.88 (-80.13)

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

weakly heterogeneous problem instances, the proposed approach yields up to 5.38% less average percentage of volume filled with 94.47% shorter CPU time than the tree-search heuristic, while for the strongly heterogeneous problem instances, it yields up to 3.64% less average percentage of volume filled with 87.94% shorter CPU time. Interestingly, as T_c increases, the proposed approach yields solutions closer to those by the tree-search heuristic in terms of average percentage of volume filled, with considerably shorter CPU time.

CONCLUSIONS

The modified wall-building-based compound approach that performs 36 wall-building heuristic methods (three ranking functions, two priority rules and six orthogonal rotations of containers) is proposed. It was tested on three real-world test problems which were weakly heterogeneous with less than 20 box types from a furniture company. There was not a winning heuristic that performed best on the three test problems. The typical wall-building heuristic methods associated with container rotation type 1 did not perform well; significant improvement was achieved by considering the six orthogonal rotations of a container in the modified approach.

The best solutions found on the three test problems were compared with the manual solutions by the furniture company. In terms of the number of containers, the proposed compound approach could save up to 33% on the three test problems. The highest fill percentages in the best solutions found were improved by up to 70.96% when compared with the manual solutions. Next, the best solutions found by the proposed approach were compared with the best solutions found by the existing tree-search heuristic method. They yielded the same results in terms of the number of containers. However, the proposed approach was slightly outperformed by the tree-search heuristic method in terms of solution quality but performed much better in terms of computational time. The best fill percentages by the proposed approach were up to 8.57% less than those by the tree-search heuristic method, while the total computational times on the three test problems were up to 98.10% shorter.

Furthermore, when compared in weakly and strongly heterogeneous instances at different levels of total box volume, the proposed approach yielded up to 5.38% and 3.64% less average percentage of volume filled with 94.47% and 87.94% shorter CPU time than the tree-search heuristic respectively. Interestingly, as the generated total box volume got higher, the proposed approach yielded solutions closer to those of the existing tree-search heuristic in terms of average percentage of volume filled with considerably shorter CPU time.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from the National Research Council of Thailand and the Thailand Research Fund.

REFERENCES

1. A. Bortfeldt and G. Wäscher, “Constraints in container loading – A state-of-the-art review”, *Eur. J. Oper. Res.*, **2013**, *229*, 1-20.
2. D. Pisinger, “Heuristics for the container loading problem”, *Eur. J. Oper. Res.*, **2002**, *141*, 382-392.
3. E. E. Bischoff and M. D. Marriott, “A comparative evaluation of heuristics for container loading”, *Eur. J. Oper. Res.*, **1990**, *44*, 267-276.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

4. P. C. Gilmore and R. E. Gomory, "Multistage cutting stock problems of two and more dimensions", *Oper. Res.*, **1965**, *13*, 94-120.
5. H. Dyckhoff, "A typology of cutting and packing problems", *Eur. J. Oper. Res.*, **1990**, *44*, 145-159.
6. G. Wäscher, H. Haubner and H. Schumann, "An improved typology of cutting and packing problems", *Eur. J. Oper. Res.*, **2007**, *183*, 1109-1130.
7. E. E. Bischoff and M. S. W. Ratcliff, "Loading multiple pallets", *J. Oper. Res. Soc.*, **1995**, *46*, 1322-1336.
8. H. Gehring, K. Menschner and M. Meyer, "A computer-based heuristic for packing pooled shipment containers", *Eur. J. Oper. Res.*, **1990**, *44*, 277-288.
9. K. He and W. Huang, "A caving degree based flake arrangement approach for the container loading problem", *Comput. Ind. Eng.*, **2010**, *59*, 344-351.
10. S. Liu, W. Tan, Z. Xu and X. Liu, "A tree search algorithm for the container loading problem", *Comput. Ind. Eng.*, **2014**, *75*, 20-30.
11. N. Wang, A. Lim and W. Zhu, "A multi-round partial beam search approach for the single container loading problem with shipment priority", *Int. J. Prod. Econ.*, **2013**, *145*, 531-540.
12. S. Martello, D. Pisinger and D. Vigo, "The three-dimensional bin packing problem", *Oper. Res.*, **2000**, *48*, 256-267.
13. C. S. Chen, S. M. Lee and Q. S. Shen, "An analytical model for the container loading problem", *Eur. J. Oper. Res.*, **1995**, *80*, 68-76.
14. L. Wei, W. Zhu and A. Lim, "A goal-driven prototype column generation strategy for the multiple container loading cost minimization problem", *Eur. J. Oper. Res.*, **2015**, *241*, 39-49.
15. S. G. Christensen and D. M. Rousøe, "Container loading with multi-drop constraints", *Master Thesis*, **2007**, Technical University of Denmark, Denmark.
16. J. A. George and D. F. Robinson, "A heuristic for packing boxes into a container", *Comput. Oper. Res.*, **1980**, *7*, 147-156.
17. M. Eley, "Solving container loading problems by block arrangement", *Eur. J. Oper. Res.*, **2002**, *141*, 393-409.
18. R. Morabito and M. Arenales, "An and/or-graph approach to the container loading problem", *Int. Trans. Oper. Res.*, **1994**, *1*, 59-73.
19. H. Gehring and A. Bortfeldt, "A genetic algorithm for solving the container loading problem", *Int. Trans. Oper. Res.*, **1997**, *4*, 401-418.
20. A. Bortfeldt and H. Gehring, "A hybrid genetic algorithm for the container loading problem", *Eur. J. Oper. Res.*, **2001**, *131*, 143-161.
21. A. Moura and J. F. Oliveira, "A GRASP approach to the container-loading problem", *IEEE Intell. Syst.*, **2005**, *20*, 50-57.
22. A. Bortfeldt, H. Gehring and D. Mack, "A parallel tabu search algorithm for solving the container loading problem", *Parallel Comput.*, **2003**, *29*, 641-662.
23. D. Mack, A. Bortfeldt and H. Gehring, "A parallel hybrid local search algorithm for the container loading problem", *Int. Trans. Oper. Res.*, **2004**, *11*, 511-533.
24. P. Thapatsuwan, P. Pongcharoen, C. Hicks and W. Chainate, "Development of a stochastic optimisation tool for solving the multiple container packing problems", *Int. J. Prod. Econ.*, **2012**, *140*, 737-748.

Maejo Int. J. Sci. Technol. **2015**, *9*(01), 93-107; doi: 10.14456/mijst.2015.8

25. J. Liu, Y. Yue, Z. Dong, C. Maple and M. Keech, "A novel hybrid tabu search approach to container loading", *Comput. Oper. Res.*, **2011**, *38*, 797-807.
26. D. Pisinger, "A minimal algorithm for the 0-1 Knapsack Problem", *Oper. Res.*, **1997**, *45*, 758-767.
27. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W. H. Freeman, Los Angeles, **1979**.
28. C. H. Papadimitriou, "On the complexity of integer programming", *J. ACM*, **1981**, *28*, 765-768.
29. D. Pisinger, "A tree-search heuristic C code", **1998**, http://www.diku.dk/~pisinger/codes.html (Accession date: August 2012).
30. D. Pisinger, "The minknap algorithm callable C code", **1993**, http://www.diku.dk/~pisinger/codes.html (Accession date: August 2012).
31. J. Hemminki, "Container loading with variable strategies in each layer", Proceedings of 10th EURO Summer Institute on Combinatorial Optimization ESI-X. Groupe HEC (Ecole des Hautes Etudes Commerciales), **1994**, Jouy-en-Josas, France.