The Container Loading Problem

Andrew Lim
Dept of IEEM, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
iealim@ust.hk

Xingwen Zhang
Dept of IEEM, Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
xwzhang@ust.hk

ABSTRACT
This paper addresses single and multiple container loading problems. We propose to use dynamic prioritization to handle awkward box types. The box type with a higher priority will be packed onto lower surfaces of the container, or packed in earlier containers. The solution found in one iteration of the algorithm is analyzed, and the priorities are updated to be used in the next iteration. Our algorithm outperformed all previous methods using standard benchmark data sets. We found the existing test data for the multiple container loading problem to be deficient and supplemented them by generating new test data consisting of 2800 test cases. The results from our algorithm using this data set are excellent.

Categories and Subject Descriptors
I.2.8 [ARTIFICIAL INTELLIGENCE]: Problem Solving, Control Methods, and Search—Heuristic methods

General Terms
Algorithms

Keywords
Container Loading, Heuristic

1. INTRODUCTION
In the container loading problem, a consignment consists of \( n \) different types of small three-dimensional rectangular boxes that are required to be loaded into a three-dimensional rectangular container. The boxes must be packed orthogonally into the container, i.e., the edges of the boxes must be parallel to one edge of the container. Each box must be completely packed in the container, and cannot overlap with any other box. Boxes with the same dimensions and same characteristics, like weight, Stock Keeping Unit (SKU), are grouped into the same box type. For each box type, a given amount of boxes are available. The boxes are allowed to be stowed in all or some of the six possible orientations.

There are two main container loading problems according to Dyckhoff [9]. The first problem is to load the entire or part of the consignment into a single container. The objective is to maximize volume utilization or to minimize the unused container volume. The second problem is to load the entire consignment into one or more containers. The objective is to minimize the number of containers used. The container loading problem contains the well-known Knapsack problem which is NP-hard. Therefore, it is also NP-hard.

We adopt the following notation for the rest of the paper: the container has length \( L \), width \( W \) and height \( H \); the \( n \) types of boxes are denoted as Box 1, Box 2, ..., Box \( n \), while each box type \( i \) has dimensions \( l_i, w_i, h_i \) and the given quantity \( m_i \). The bottom-left-back corner of the container is the origin of the coordinate system; the length, width and height of the container lie on the positive \( x \), \( y \) and \( z \) axes.

2. LITERATURE REVIEW
Based on the number of different box types, the container loading problem can be classified into three categories, namely, the homogeneous problem with only one type of box, the weakly heterogeneous problem with relatively few box types and the strongly heterogeneous problem with a large number of different box types.

The container loading problem with homogeneous boxes is reduced to the two-dimensional “Pallet Loading Problem” (PLP) by assuming that the boxes are only allowed to be placed upright. Nelißen [17] provided an excellent survey of the PLP. Our algorithms for the single and multiple container loading problems use the G4-heuristic [20, 19] for the PLP as a sub-routine to pack a certain amount of boxes of a single type onto a given surface. For the heterogeneous container loading problem, more than one type of box are to be loaded in the container. Due to the problem's complexity most results on this topic are based on heuristics and meta-heuristics. There are two basic approaches used in the heuristics to solve the single container loading problem with heterogeneous boxes. These methods are based on wall-building and layer-building. The wall-building approach constructs vertical walls across the container length or width, while the layer-building approach builds the loading plan layer by layer from the floor of the container upwards. [12], [7], [21], [5], [2], [10], [18] and [3] represent the most recent work on the single container loading problem. In contrast to the large number of heuristics proposed for the single container loading problem, the multiple container loading problem has received less attention. [21] and [10] represent the most recent work on this problem.
3. ALGORITHM FOR THE SINGLE CONTAINER LOADING PROBLEM

In the previous approaches for single container loading problem, the existence of “trouble-making” box types is widely observed. They are those box types that are relatively large in relation to the dimensions of the container and are believed to be awkward to pack. In most of the previous approaches, ranking rules are applied to accommodate boxes of large sizes in earlier stages. For example, in [13], the approach gives the box type with larger size in its smallest dimension a higher ranking. In [10], the boxes are sorted by volume with larger boxes being chosen first. The ranking rules are always predefined and applied to all problem instances throughout the approaches, which is not flexible and may not be accurate for all instances.

The existence of “trouble-making” elements suggests that the use of dynamic prioritization in the algorithm would possibly improve the solution quality, as inspired by the GRASP [11] and SWO [15, 16] heuristics. In our algorithm for single container loading problems, each box type is assigned a priority factor. The priority factor (blame) of each box type is initially one and is dynamically updated from iteration to iteration during the execution of the algorithm.

During an iteration of the algorithm, a greedy heuristic integrated with a tree-search procedure is applied to load the container. The best solution found by the greedy heuristic is analyzed, and the priority factors are updated with knowledge learned from the analysis. Those box types that are considered as “trouble-makers” are assigned more blame, i.e., their priority factors are increased more. Then, the greedy heuristic is applied again and the above procedures are repeated. This Construct/Analyze/Prioritize [15] cycle continues until the maximal number of iterations has been reached or the time limit has been exceeded.

In the greedy heuristic, box types with higher priority factors would be handled in the earlier stages of the heuristic, that is, they would be packed onto lower surfaces. Compared to the priority factor concept in [4], the analysis of solutions constructed by the greedy heuristic and the prioritization guided by the analysis enable the algorithm to learn from previous iterations to update the priority factors, and the newly updated priority factors in turn determine the next execution of the greedy heuristic.

In the greedy heuristic, the evaluation value of a complete or partial solution is the sum of the evaluation value of each packed box. The evaluation value of each packed box is its volume for the single container loading problem. Therefore, for single container loading problems, the evaluation value is the same as the volume utilization.

3.1 The Greedy Heuristic

The greedy algorithm constructs the loading plan of a single container layer by layer from the bottom up. At the initial stage, the list of available surfaces contains only the initial surface of size $L \times W$ with its initial position at height 0. At each step, the algorithm picks the lowest usable surface and then determines the box type to be packed onto the surface, the number of the boxes and the rectangle area the boxes to be packed onto, by the procedure selectLayer. The procedure selectLayer calculates a layer of boxes of the same type with the highest evaluation value. The procedure selectLayer uses breadth-limited tree search heuristic to determine the most promising layer, where the breadth is different depending on the different depth level in the tree search. The advantage is that the number of nodes expanded is polynomial to the maximal depth of the problem, instead of exponentially growing with regard to the problem size. After packing the specified number of boxes onto the surface according to the layer arrangement, the surface is divided into up to three sub-surfaces by the procedure divideSurfaces in Section 3.1.1. Then, the original surface is deleted from the list of available surfaces and the newly generated sub-surfaces are inserted into the list. Then, the algorithm selects the new lowest usable surface and repeats the above procedures until no surface is available or all the boxes have been packed into the container. The algorithm follows a similar basic framework as that described in [2]. The pseudo-code of the greedy algorithm is given by the greedyHeuristic procedure.

procedure greedyHeuristic()
list of surface := initial surface of $L \times W$ at height 0
list of box type := all box types
while (there exist usable surfaces) and (not all boxes are packed) do
  select the lowest usable surface as current surface
  set depth := 0
  set bestLayer := selectLayer(list of surface, list of box type, depth)
  pack bestLayer on current surface
  reduce the number of the packed box type by the packed amount
  set a list of new surfaces := divideSurfaces(current surface, bestLayer, list of box type)
  delete current surface from the list of surfaces
  insert each surface in list of new surfaces into list of surfaces
end while

3.1.1 The Procedure to Divide the Loading Surface into Sub-Surfaces

Given a layer of boxes of the same type arranged by the G4-heuristic, the layer is always packed in the bottom-left corner of the loading surface. As illustrated in Figure 1, up to three sub-surfaces are to be created from the original loading surface by the procedure divideSurfaces, including the top surface, which is above the layer just packed, and the possible spaces that might be left at the sides. If $l = L$ or $w = W$, the original surface is simply divided into one or two sub-surfaces, the top surface and a possible side surface. Otherwise, two possible division variants exist, i.e., to divide into the top surface, the surface $(B, C, E, F)$ and the surface $(F, G, H, I)$, or to divide into the top surface, the surface $(B, C, D, I)$ and the surface $(D, E, G, H)$. The divisions are done according to the following criteria, which are similar to those in [2] and [5].

The primary criterion is to minimize the total unusable area of the division variant. If none of the remaining boxes can be packed onto a sub-surface, the area of the sub-surface is unusable. The secondary criterion is to avoid the creation of long narrow strips. The underlying rationale is that narrow areas might be difficult to fill subsequently. More specifically, if $L - l \geq W - w$, the loading surface is divided into the top surface, the surface $(B, C, E, F)$ and the surface $(F, G, H, I)$. Otherwise, it is divided into the top surface, the surface $(B, C, D, I)$ and the surface $(D, E, G, H)$.
3.2 The Analyzer and Prioritizer

In the analyze/prioritize step, those box types that are considered as “awkward” are assigned more blame, which causes their priority factors to increase more. The priority factors will then be used to guide the greedy heuristic in the construction step. A number of possible ways exist to analyze the solution and update the priority factors. For example, a possible way is to increase the priority factor of each box type by the amount proportional to the number of boxes that are not loaded. In view of the situation that the portion of consignment that is not loaded is relatively small for single container loading problems, the unloaded part of the consignment is considered to be loaded in an assumptive second container, and the priority factor of each box type is increased by an amount that is equal to \((1 - \text{the average volume utilization of the box type})\). The volume utilization for each single box is defined to be the volume utilization of the container in which it is packed, while the volume utilization of a container is the ratio of the sum of the volume of the packed boxes to the total container volume. The average utilization volume for each box type is the sum of the volume utilization of each box of the box type divided by the total number of the box type. The underlying rationale for the prioritization scheme is that the unpacked boxes should be handled in earlier stages of the construction heuristic, that is, they should be packed onto lower surfaces of the container.

4. ALGORITHM FOR MULTIPLE CONTAINER LOADING PROBLEMS

Our greedy heuristic construction for single container loading problems is modified and extended to address the multiple container loading problem by using the sequential approach. That is, the containers are filled one after another. The approach attempts to fill a single container with the highest evaluation value by the greedy heuristic for single container loading problems. Then, those boxes already packed in the container are removed from the consignment. The construction heuristic goes on with the reduced consignment and a new empty container until the whole consignment is loaded. The reasons for using the sequential approach, instead of the simultaneous one, include the use of prioritization, the superiority of the sequential approach over the simultaneous approach for layer-building algorithms [4] and the ease of implementation.

Similar to the situation in single container loading problems, the existence of “trouble-making” box types is also observed in multiple container loading problems [4]. The existence of trouble spots means that the idea of prioritization for the single container loading problem would possibly be useful for the multiple container loading problems. In the original sequential approach, as the containers are packed one after another, a possible drawback is that boxes used up during the packing of early containers might be better used to pack later containers. On the other hand, boxes that should be packed in earlier containers are left to later containers, which causes the later containers to have poor volume utilizations. This drawback can be overcome by the prioritization, at least partially. With the prioritization, when the containers are packed sequentially, the boxes with high priorities are handled in earlier stages, i.e., they are packed in the earlier containers. Those boxes with low priorities will be left over to later stages, i.e., they are packed in the later containers. As those boxes with lower priorities are more easily packed, the volume utilizations of the later containers will increase. Thus, the approach would possibly reduce the number of containers used. The excellent integration of the prioritization with the sequential approach is also verified by good experimental results in the following section.

When the greedy heuristic is applied to pack a single container in multiple container loading problems, the evaluation value of a complete or partial packing plan for a container is the sum of the evaluation values of each packed box. The evaluation value of each packed box is its volume multiplied by the priority factor of the box type. Therefore, for multiple container loading problems, the evaluation value is not the same as the volume utilization, such that the effect of prioritization is not compromised.

During an iteration of the algorithm for multiple container loading problems, the greedy construction heuristic loads the whole consignment into a number of containers sequentially. The solution found by the greedy heuristic is analyzed and the priority factors are updated by the same analyzer and prioritizer used in the single container loading problem. Then, the greedy heuristic is applied again with new priority factors and the above procedures are repeated. This cycle continues until the maximal number of iterations has been reached or the time limit has been exceeded.

5. EXPERIMENTAL RESULTS

The algorithms for container loading problems, including single heterogeneous cases and multiple heterogeneous cases, were implemented in Java. The experiments were carried out on a 2.40 GHz Pentium 4 PC with 128 MB memory limit. We refer to our algorithm as the IC-Algorithm (Iterated Construction Algorithm) in the following sections.

5.1 Experimental Results for Single Heterogeneous Container Loading Problems

Our approach for single heterogeneous container loading problems was tested on standard benchmark instances of 700 instances of Bischoff and Ratcliff [3]. These benchmark instances are available online in the OR-Library [4]. The 700 Bischoff/Ratcliff test cases are divided into seven test classes, namely BR1 to B7, where each test class consists of 100 test cases. Our approach was compared with the approaches of Bischoff, Janetz and Ratcliff, 1995 (B/J/R)[2], of Bischoff and Ratcliff, 1995a (B/R)[3], of Gehring and Bortfeldt, 1997 (G/B)[12], of Bortfeldt and Gehring, 1998 (B/G)[6], of Davies and Bischoff, 1999 (D/B)[8], of Terno,
Table 1: Volume utilization (%) for BR1 to BR7 with the IC-Algorithm

<table>
<thead>
<tr>
<th>Problem</th>
<th>Minimum</th>
<th>Average</th>
<th>Maximum</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR1(3)</td>
<td>91.18</td>
<td>91.60</td>
<td>91.97</td>
<td>0.49</td>
</tr>
<tr>
<td>BR3(5)</td>
<td>86.61</td>
<td>91.99</td>
<td>96.26</td>
<td>1.93</td>
</tr>
<tr>
<td>BR3(8)</td>
<td>87.84</td>
<td>92.30</td>
<td>94.67</td>
<td>1.19</td>
</tr>
<tr>
<td>BR4(10)</td>
<td>89.70</td>
<td>92.36</td>
<td>94.79</td>
<td>1.05</td>
</tr>
<tr>
<td>BR5(12)</td>
<td>88.88</td>
<td>91.90</td>
<td>94.33</td>
<td>0.93</td>
</tr>
<tr>
<td>BR6(15)</td>
<td>89.18</td>
<td>91.51</td>
<td>93.76</td>
<td>0.86</td>
</tr>
<tr>
<td>BR7(20)</td>
<td>88.85</td>
<td>91.01</td>
<td>93.11</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Table 2: Average volume utilization for BR1 to BR7

<table>
<thead>
<tr>
<th>Approach</th>
<th>Volume Utilization(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B/J/R(1995a)</td>
<td>81.97</td>
</tr>
<tr>
<td>B/R(1995a)</td>
<td>83.45</td>
</tr>
<tr>
<td>B/R-comb</td>
<td>84.94</td>
</tr>
<tr>
<td>G/B(1997)</td>
<td>88.30</td>
</tr>
<tr>
<td>H/G(1998)</td>
<td>91.26</td>
</tr>
<tr>
<td>D/B(1999)</td>
<td>84.19</td>
</tr>
<tr>
<td>B/G(2001)</td>
<td>90.06</td>
</tr>
<tr>
<td>E(2002)</td>
<td>88.75</td>
</tr>
<tr>
<td>IC(2003)</td>
<td>91.81</td>
</tr>
</tbody>
</table>

The experimental results for the 700 Bischoff/Ratcliff test cases are summarized in Table 1. The minimal, average and maximal volume utilizations of 100 test instances for each test class are presented, together with the standard deviations of the instances. For all test classes, the average volume utilization remains above 91%.

In Table 2, the overall average volume utilization for the 700 test cases is given and compared with the utilization achieved by other methods. The volume utilizations of other methods are taken from the literature and updated with the most recent publications. Among the 10 approaches, our approach achieved the highest volume utilization, about 0.55% higher than the second highest one. The average running time for the 700 test cases is 706.59 seconds, while the minimal and maximal times are 0.89 seconds and 4664.30 seconds respectively.

5.2 Experimental Results for Multiple Heterogeneous Container Loading Problems

Our approach for multiple heterogeneous container loading problems was tested on standard benchmark instances from [14] and a new set of test classes that extends the BR set of test classes, BR1 to BR7, for multiple cases.

In [14], 47 test cases, denoted as IMM01 to IMM47, were used for multiple container loading problems. For the IMM test class, our approach was compared with the approach by Ivancic, Mathur and Mohanty 1989 (I/M/M)[14], the sequential approach of Bischoff and Ratcliff, 1995a (B/R)[3], the sequential approach of Bischoff and Ratcliff, 1995b (B/R)[4], the sequential approach of Eley, 2002 (E-seq)[10], and the simultaneous approach of Eley 2002 (E-sim)[10].

For the sake of completeness of the experiments and in order to obtain statistically reliable results and to provide potential benchmarks for further research on multiple container loading problems, a new set of test classes was generated from BR1 to BR7. The generator used a framework similar to that described in [4]. The only input parameter of the generator is the target ratio, $R$, which is defined as the desired sum of the volume of the boxes in the cargo divided by the volume of a single container. The total volume of the consignment will be larger than or equal to $T_v = R \times L \times W \times H$ and less than $(T_v + \text{the volume of the largest box})$.

The values of the target ratio, $R$, are chosen to be 5.00, 5.25, 5.50 and 5.75, which leads to four sets of test classes (BR-5.00-1 to BR-5.00-7, BR-5.25-1 to BR-5.25-7, BR-5.50-1 to BR-5.50-7, and BR-5.75-1 to BR-5.75-7). Therefore, altogether 4 x 700 = 2800 test cases are generated and experimented with. With these target ratios, as in [4], at least six containers are needed to load the whole consignment. The reason for using the above four values for the target ratio, $R$, instead of using only 5.00 as in [10], is to provide more challenging and representative test instances. When $R = 5.00$, experiments have shown that six containers are enough to load the whole consignment for most of the test cases. From the point of view of volume utilization, loading a consignment with the total volume just over five times the container volume in six containers results in an average volume utilization of 83.3%, which is relatively undemanding.

The experimental results for the 47 IMM test cases, together with those in the literature and the volume lower bounds ($v_{lb}$), are given in Table 3. Among all the approaches, our algorithm achieved the best overall result. The total number of containers required is reduced to 694, and it is the first approach that is able to solve all 47 instances by using fewer than 700 containers. Meanwhile, 21 of the 47 test cases are solved optimally by our approach. Detailed comparisons of different results for each test case show that our approach dominates all other approaches, as our approach generated at least an equivalent solution for every test case, if not better. In Table 3, our approach generated better solutions than any other methods for 12 test cases. For IMM21, the approach produced a solution with 20 containers, which led to a significant improvement of four containers. The average running time for the 47 test cases is 6.43 seconds, while the maximal time is 107.70 seconds.

The experimental results for the 2800 newly generated benchmark test cases are summarized in Table 4. For each test class, BR-R-K ($R$=5.00 to 5.75, $K$=1 to 7), which consists of 100 test cases, the total number of containers required is given. As mentioned previously, at least six containers are required to load the whole consignment for each test case, so the lower bound of the total number of containers required for each test class is 600. Table 4 shows that for most test cases of the test classes with $R$=5.00, 5.25 and 5.50, our approach is able to load the whole consignment using only six containers. For the test class with $R$=5.75, when most of the test cases require 7 containers, 7.29% (51 out of 700) require only 6 containers. The results are satisfactory, as loading a consignment with a total volume over 5.75 times the container volume in 6 containers results in an average volume utilization of more than 95.83%. Among all 2800 test cases, only three require eight containers and none requires more than eight.

Eley [10] applied his sequential and simultaneous approaches on a set of test cases randomly generated by a similar procedure with the target ratio $R=5.0$. Out of 100 test cases with three or five types of boxes, eight of them require an
6. CONCLUSION

In the paper, a new algorithm for the single container loading problem is developed. The algorithm uses dynamic prioritization to handle trouble-making box types. The box type with a higher priority is packed onto lower surfaces of the container. The solution found by the construction heuristic is analyzed and the priorities are updated and used in the next execution of the construction heuristic. The algorithm is modified and extended to address multiple container loading problems by using the sequential approach. With dynamic prioritization and the sequential approach, the trouble-making box types with high priorities are packed in the earlier containers.

7. REFERENCES


---

**Table 3: Results for test cases IMM01 to IMM47**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value of Target Ratio $R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.90</td>
</tr>
<tr>
<td>BR-R-1(5)</td>
<td>601</td>
</tr>
<tr>
<td>BR-R-2(6)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-3(6)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-4(10)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-5(12)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-6(15)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-7(20)</td>
<td>600</td>
</tr>
</tbody>
</table>

---

**Table 4: Results for test classes BR-R-1 to BR-R-7**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Value of Target Ratio $R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.90</td>
</tr>
<tr>
<td>BR-R-1(5)</td>
<td>601</td>
</tr>
<tr>
<td>BR-R-2(6)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-3(6)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-4(10)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-5(12)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-6(15)</td>
<td>600</td>
</tr>
<tr>
<td>BR-R-7(20)</td>
<td>600</td>
</tr>
</tbody>
</table>

---


---

**European Journal of Operational Research, 131:143–161, 2001.**

---

**European Journal of Operational Research, 114:509–527, 1999.**

---

**European Journal of Operational Research, 44:145–159, 1990.**

---

**European Journal of Operational Research, 141:393–409, 2002.**

---

**Journal of Global Optimization, 6:109–133, 1995.**

---

**International Transactions of Operational Research, 4:401–418, 1997.**

---

**Computer and Operations Research, 7:147–156, 1980.**

---


---


---

**Journal of Artificial Intelligence Research, 10:353–373, 1999.**

---

**Schriften zur Informatik und Angewandten Mathematik 155, July 1993.**

---

**European Journal of Operational Research, 6:382–392, 2002.**

---


---

**European Journal of Operational Research, 123:372–381, 2000.**