Container loading with multi-drop constraints

Søren Gram Christensen* David Magid Rousøe
Informatics and Mathematical Modelling, Technical University of Denmark
Kgs. Lyngby, Denmark

July 1, 2007

Abstract

In this paper an algorithm for the container loading problem (CLP) with multi-drop constraints is presented. When adding multi-drop constraints we demand, that the relevant boxes must be available, without rearranging others, when each drop-off point is reached. To make the solutions feasible in the real world, it is further demanded that all boxes are placed in a feasible manner with respect to load bearing strength and with proper support from below. This makes it possible to load consignments originating from builder merchants.

A heuristic based on a tree search framework is proposed. It uses greedy solutions to evaluate each choice taken. To make the framework more generic, a dynamic breadth is proposed. Based on problem characteristics and the time limit imposed, it will choose the breadth of the tree, making sure that the time is utilised most profitable.

The algorithm is tested on new real world data from a Danish company distributing construction products. For the solutions to these problems to be feasible in a real world setting both multi-drop- and load bearing strength constraints are essential. The obtained results show, that the proposed model and algorithm are able to solve the new real world problems in fractions of a second. Furthermore, results obtained on benchmark problems indicate that the algorithm performs comparable with other more specialised methods.

Keywords Packing; Container loading; Multi-drop; Load bearing strength; Tree-search

1 Introduction

The container loading problem (CLP) is the problem of loading a subset of rectangular boxes into a rectangular container of fixed dimensions such that the volume of the packed boxes is maximized. If boxes belonging to different customers are loaded in the same container, a natural extension to CLP is multi-drop constraints, which demand that boxes should be available without rearranging other boxes when a customer is reached. In recent literature the problem of combined vehicle routing and container loading is considered. Gendreau et al. (2006a) propose an algorithm to solve the combined problem with three dimensional

*Tel.: +45 23661882
E-mail address: soren@gramchristensen.dk

packing constraints. This problem is denoted the three dimensional loading capacitated vehicle routing problem (3L-CVRP). Other approaches have reduced the loading constraints, so that only one or two dimensional loads are considered. The two dimensional loading capacitated vehicle routing problem (2L-CVRP) are dealt with by e.g. Gendreau et al. (2006b) and Iori et al. (2007). The one dimensional variant, where multi-drop constraints are applied, is considered in Doerner et al. (2007). Common to the approaches is that the two problems are solved separately. A vehicle routing algorithm suggest routes which are checked for feasibility by a packing algorithm. For this setup to work in a practical setting, solutions to the CLP must be obtained fast, as typical many routes need to be checked. Furthermore, when the container loading problem is combined with vehicle routing the multi-drop constraints on CLP become essential.

Many different properties make up the characteristics of container loading problems. Both the typologies of cutting and packing problems by Dyckhoff (1990) and Wäscher et al. (2006) are good starting points for investigations in the area of cutting and packing. When cutting and packing problems are classified, it is important to describe the dimensionality, the assortment of the small items and large objects and the shape of the small items. In this paper a heuristic for the three dimensional rectangular single container loading problem is presented. Using the typology of Wäscher et al. (2006) this problem is classified *SLOPP*, as a weakly heterogeneous assortment of boxes are considered. In the typology of Dyckhoff (1990) the problem is classified *3/B/O/F* and *3/B/O/R*, as also problems with few boxes are considered in this paper. We further demand that boxes are only rotated in feasible directions, meaning that it should be respected if a box cannot be rotated in some given ways.

Bischoff and Ratcliff (1995) give an excellent overview of what is in the typologies denoted *problem extensions*. These include both multi-drop- and load bearing strength constraints and rotation restrictions. The load bearing strength constraints ensure that boxes placed on top of each other does not damage each other. This type of constraints becomes very important if the problem is to load construction products, as these often have very different characteristics.

The CLP is known to be an NP-hard optimisation problem, as it contains the well described knapsack problem. For this reason, when solution procedures are developed, heuristics are often used. Mixed integer programming (MIP) models have, however, been put forward. Chen et al. (1993) proposed a MIP model to the multi-container loading problem, which can easily be changed to also model the single container loading problem. Scheithauer (1998) proposed a different MIP model, which he uses to produce dual bounds to CLP.

When heuristics are used a general solution strategy is to divide the container into smaller pieces, and then packing each of these separately. The point often being to reduce the solution space from three dimensions to one or two. Pisinger (2002) gives an excellent overview of these strategies. The most common dividing procedure is *wall* building, introduced by George and Robinson (1980). A wall is constructed by making a vertical slice through the container. The depth of a slice is defined by the depth of the first box placed in the wall. As the slice is filled, the boxes will create a wall-like formation. See e.g. Bortfeldt and Gehring (2001) who combines wall building with a genetic algorithm. Moura and Oliviera (2005) uses walls in combination with a GRASP approach, and Pisinger (2002) uses a tree search to decide the depth of each wall. Davies

and Bischoff (1999) propose a method that combines several walls into a block, to increase stability. Methods where the container is split by horizontal slices have also been developed. This is often called *layers*. Compared to the wall approaches, layer approaches are said to produce more stable loads. Examples can be seen in Bischoff and Ratcliff (1995), Ratcliff and Bischoff (1998) and Lim and Zhang (2005). In *stack* building approaches (see e.g. Gehring and Bortfeldt (1997) and Gilmore and Gomory (1965)) box stacks are constructed, leaving only a two dimensional problem of arranging the stacks on the container floor. Other methods put together boxes in *blocks* before they are placed in the container. A block typically consists of one or two types of boxes. Therefore the advantage is, that each block can be tightly packed. This procedure is used by Bortfeldt et al. (2003) in combination with a tabu search, by Mack et al. (2004) with a hybrid parallel tabu search and simulated annealing, and by Eley (2002) in combination with a tree search algorithm.

A common term used, in connection with wall and layer building algorithms, but also by itself is *guillotine* cuts. The term is taken from cutting theory, and it describes a cut going straight through an object. The cut will continue until another guillotine cut is met or the object is cut through. When cutting glass, it is often demanded that the cutting patterns are guillotine cuttable. In relation to container loading, guillotine cuts can be used to split the container into smaller pieces, which is utilised by Morabito and Arenales (1994).

Only few papers addresses the problem of container loading with multi-drop constraints. Bischoff and Ratcliff (1995) propose a method, which load a single customer at a time. Another approach is mentioned in Gendreau et al. (2006a), where the combined routing and packing problem is solved. They use a tabu search to find a solution to a strip packing problem which is good enough to fit in the real container. The shared idea in the two approaches is to load the boxes in the opposite sequence as they should be dropped off (LIFO). The latter approach furthermore introduces the idea to check the feasibility of an insertion, before a box is placed. This was also done in Gendreau et al. (2006b) for a two dimensional packing problem.

Another rarely mentioned extension, is the load bearing strength constraint. How to incorporate this constraint into the algorithms is discussed in Ratcliff and Bischoff (1998) and Bischoff (2004). The first approach uses the concept of empty spaces to model the container whereas the latter uses two matrices to represent occupied room in the container and remaining load bearing strength. The latter approach can be seen as a simplification of the ideas in Ngoi et al. (1994) to represent the container room with matrices.

The problem, which is treated in this paper, is highly restricted compared to most problems considered in the literature. We demand that boxes are placed multi-drop- and load bearing strength feasible, only rotated in some pre-described directions and given proper support from below.

In the next section the mathematical model handling the imposed constraints is described. In Section 3 the proposed tree search algorithm and the new dynamic breadth are introduced. In Section 4 the algorithm is tested on both new real world data and on benchmark data from the OR-Library. On the new data the introduced constraints are essential to make the loads practically feasible. Finally some concluding remarks are given in Section 5.

2 Mathematical model

The container is placed in a coordinate system with the origin in the back-left-lower corner. The length L of the container is placed along the x-axis, the width W along the y-axis and the height H along the z-axis. Every box $i \in B$ has dimensions l_i, w_i and h_i and weight m_i. Dependent on the chosen rotation of the box the different sides of the box will be placed parallel to the sides of the container. To every box a set of feasible rotations are given. To a box there can be either two, four or six feasible rotations. When a box i is placed it is placed in the point $(x, y, z) = (a_i, b_i, c_i)$, which will also be denoted the minimum of the box $\underline{i} = (\underline{a}_i, \underline{b}_i, \underline{c}_i)$. Furthermore when a box is placed the rotation of the box is chosen. Then the maximum of the box is given by $\overline{i} = (\overline{a}_i, \overline{b}_i, \overline{c}_i)$. We will be working with two distinct set of boxes. The set of already placed boxes B_{placed} and the set of not placed boxes $B_{nPlaced}$.

To manage the space inside the container different strategies are presented in the literature. Ngoi et al. (1994) use a matrix representation to describe where and how the boxes are placed inside a container. Another approach was pointed out by George and Robinson (1980) based on empty spaces describing where boxes can be placed. In the proposed algorithm we use empty spaces that are allowed to overlap each other. This choice complicates the procedures needed to update the empty spaces, as more spaces can be affected by the insertion of a single box. On the other hand, this approach does not limit the spaces in size unnecessarily. Every space $s \in S$ is given by its minimum $\underline{s} = (\underline{x}_s, \underline{y}_s, \underline{z}_s)$ and maximum $\overline{s} = (\overline{x}_s, \overline{y}_s, \overline{z}_s)$. A valid empty space is guaranteed to be empty, therefore we need to check all spaces after a box is inserted, and update these if the space is no longer empty.

To reflect the reality met when for instance packing construction products for distribution, it is necessary to allow the spaces to overhang each other. This means that there is no guarantee of full support for the boxes. The amount of support is controlled by the non-negative parameter γ, which describes how large a part of a space that does not need support. If $\gamma = 1$, the supported and the not supported parts of the space have the same size. If $\gamma = 0$, no overhang is allowed. Additionally two variables α and β are used to describe where support from below exists in the x- and y-dimensions, respectively.

2.1 Updating the empty spaces

When a solution to the CLP is constructed, many boxes are placed in the container. The boxes are always placed in empty spaces. This ensures that all boxes will have proper support from below. Furthermore we will always place the box in the minimum of a space. This follows the concept of a normalised packing. It can be shown that to any packing, an equally good normalised packing exists. This was done by Martello et al. (2000), when packing a single bin in the three dimensional bin packing problem and by Herz (1972) for the two dimensional stock cutting problem. After a box is placed it will occupy some of the space in the container, therefore the empty spaces need to be updated to reflect this new situation.

Because we work with overlapping empty spaces many new spaces can be generated when a box is inserted. Spaces on all six sides of the inserted box can appear. In Figure 1 an example where three new spaces are generated is shown.

On the left picture the container is shown with the old invalid space. On the right picture the new empty space are drawn on the container floor.

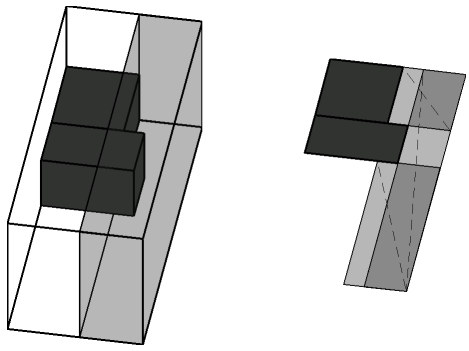

Figure 1: Three new spaces generated after a box is inserted. On the left picture the container is shown in three dimensions with the old invalid empty space. On the right picture the container floor is shown. The dark grey area illustrate where the new spaces, indicated by dashed lines from their minimum to maximum, overlap each other.

As mentioned, in general six new spaces can be generated after a box is inserted. If we place box i in the container and it invalidates space s the potential new empty spaces are given in Table 1. Here, space $t = 1$ is situated behind

Table 1: The spaces made in other spaces when inserting a box and reduced support is allowed.

	min			**Support**	
t	$\underline{x}_t$	$\underline{y}_t$	$\underline{z}_t$	α_t	β_t
1	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$	$\min(\alpha_s, \underline{a}_i)$	β_s
2	$\overline{a}_i$	$\underline{y}_s$	$\underline{z}_s$	α_s	β_s
3	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$	α_s	$\min(\beta_s, \underline{b}_i)$
4	$\underline{x}_s$	$\overline{b}_i$	$\underline{z}_s$	α_s	β_s
5	$\underline{x}_s$	$\underline{y}_s$	$\underline{z}_s$	α_s	β_s
6	$\underline{x}_s$	$\underline{y}_s$	$\overline{c}_i$	$\min(\overline{a}_i, \alpha_s)$	$\min(\overline{b}_i, \beta_s)$

	max		
t	$\overline{x}_t$	$\overline{y}_t$	$\overline{z}_t$
1	$\underline{a}_i$	$\overline{y}_s$	$\overline{z}_s$
2	$\min(\alpha_s \cdot (1+\gamma) - \overline{a}_i \cdot \gamma, \overline{x}_s)$	$\overline{y}_s$	$\overline{z}_s$
3	$\overline{x}_s$	$\underline{b}_i$	$\overline{z}_s$
4	$\overline{x}_s$	$\min(\beta_s \cdot (1+\gamma) - \overline{b}_i \cdot \gamma, \overline{y}_s)$	$\overline{z}_s$
5	$\overline{x}_s$	$\overline{y}_s$	$\underline{c}_i$
6	$\min(\overline{a}_i \cdot (1+\gamma) - \underline{a}_i \cdot \gamma, \overline{x}_s)$	$\min(\overline{b}_i \cdot (1+\gamma) - \underline{b}_I \cdot \gamma, \overline{y}_s)$	$\overline{z}_s$

the box, space $t = 2$ is in front of, space $t = 3$ left of, $t = 4$ right of, $t = 5$ is situated below the box and $t = 6$ is placed on top of the box. The spaces are only generated if they are needed. In general only a subset of the six spaces are generated from a single space. The space above is only generated when $\underline{s} = \underline{i}$, i.e. if the the space has the same minimum as the box. This is necessary because many spaces can have the same minimum but different support from below.

When updating the empty spaces a number of crucial operations are performed to keep the spaces as large and few as possible. First of all spaces which

have the same $\underline{z}$ and abut on each other are amalgamated. This is done based on the supported parts of the spaces. Furthermore, it is demanded that spaces which are subsets of other spaces are deleted. Finally, spaces which are too small to hold any box will be deleted. This is only done if there is no chance that the space will ever be amalgamated with another space. The details in these operations are described more thoroughly in Christensen and Rousøe (2007).

2.2 A valid box placement

The way the spaces are constructed makes sure that boxes will have proper support from below when they are placed in the container. This was one of the extensions introduced on the problem at hand. Two more concerns have to be taken into account. First of all the multi-drop constraints and secondly that the boxes only have a limited load bearing strength. The validity of both constraints are ensured before a box is placed in the container.

A solution to CLP is *multi-drop feasible* if boxes can be unloaded without rearranging other boxes. In practice we demand that there exist a *passage* from the box to an entrance of the container. We are working with whole surfaces as entrances to the container and define the four parameters X^+, X^-, Y^+ and Y^- to represent, if it is possible to unload boxes from the surfaces given by $(L,0,0)-(L,W,Z)$, $(0,0,0)-(0,W,Z)$, $(0,W,0)-(L,W,Z)$ and $(0,0,0)-(L,0,Z)$, respectively. X^+ for instance represents whether the front of the container is a valid unloading surface.

For each unloading direction a passage is defined as follows. There exist a passage for box i in the X^- direction if:

$$\forall j \in B_{placed} \setminus i \mid a_i \leq a_j \vee d_i \geq d_j \vee \neg(\underline{b}_i < \overline{b}_j \wedge \underline{b}_j < \overline{b}_i).$$

There exist a passage for box i in the X^+ direction if:

$$\forall j \in B_{placed} \setminus i \mid a_i \geq a_j \vee d_i \geq d_j \vee \neg(\underline{b}_i < \overline{b}_j \wedge \underline{b}_j < \overline{b}_i).$$

There exist a passage for box i in the Y^- direction if:

$$\forall j \in B_{placed} \setminus i \mid b_i \leq b_j \vee d_i \geq d_j \vee \neg(\underline{a}_i < \overline{a}_j \wedge \underline{a}_j < \overline{a}_i).$$

There exist a passage for box i in the Y^+ direction if:

$$\forall j \in B_{placed} \setminus i \mid b_i \geq b_j \vee d_i \geq d_j \vee \neg(\underline{a}_i < \overline{a}_j \wedge \underline{a}_j < \overline{a}_i).$$

Furthermore we demand that no boxes placed on top of box i has lower sequence number. Two boxes i and j are placed on top of each other if they overlap in the x- and y-dimensions, i.e. $\underline{a}_i < \overline{a}_j \wedge \underline{a}_j < \overline{a}_i \wedge \underline{b}_i < \overline{b}_j \wedge \underline{b}_j < \overline{b}_i$. If this happens we demand that the lowest placed box has the higher sequence number. Without loss of generality lets assume that box i is placed lower than box j and the two boxes are placed on top of each other. Then we demand that the sequence number of box i is at least as high as the sequence number of box j, i.e. $d_i \geq d_j$.

Now we can define when a load is multi-drop feasible. A solution to CLP respects the multi-drop constraints if: For all boxes i, there exist a passage in a *possible* unloading direction, and no boxes j placed on top of box i, have higher sequence number. This situation is controlled every time a box is inserted in the solution. Thereby all partial loads are also multi-drop feasible.

The concept of *load bearing strength* (LBS) was first suggested by Bischoff and Ratcliff (1995). The basic idea is that all boxes are fragile to some extent. Hence, we need to guarantee that no boxes are put under more pressure from other boxes than allowed. Load bearing strength is a measure for how much weight a given box can carry without being damaged.

To model this feature we need to know, for every box $i \in B$ what its load bearing strength is. This is denoted LBS_i. In our model there exist one LBS-value for every box, no matter how the box is rotated. This differs from the work of Ratcliff and Bischoff (1998) who use three LBS, one for each dimension. We will assume that the mass of the products are homogeneous distributed in the products. When dealing with construction products this is in most situations the case. However, if boxes are overhanging the moment of force is neglected. Of course only boxes which support each other should be strained with each others weight. Furthermore boxes which does not have full support should strain other boxes with more weight proportional to the not supported area.

To handle this the concept of *box stacks* is introduced. A box stack $bs \in BS$, with reference point, $r = (r_x, r_y, 0)$, is composed of all boxes where

$$\underline{a}_i < r_x < \overline{a}_i \ \wedge \ \underline{b}_i < r_y < \overline{b}_i \ \wedge \ \left(\underline{c}_i = 0 \vee (\exists j \in bs, j \neq i | \underline{c}_i = \overline{c}_j)\right)$$

is true. This means that boxes are only members of a box stack, if they have support in the reference point by either another box in the box stack or the container floor.

When a box is overhanging it means that the supported area is smaller than the base area of the box. The supported area of box j is denoted S_j^{area}. When we try to insert a box the following check should be made for all boxes in the different box stacks wherein the inserted box is a member:

$$LBS_i \geq \sum_{\substack{j \in bs \\ \underline{c}_i < \underline{c}_j}} \frac{m_j}{S_j^{area}}, \quad \forall i \in bs.$$

3 Tree search

To manage in which sequence the different boxes in a CLP are placed, a tree search heuristic is proposed. It places one box in each node of the search tree. Thereby each node constitutes a *partial solution* PS. Like Eley (2002) the space choice is made greedily and each insertion is evaluated by a corresponding greedy solution. All different combinations of box types and associated feasible rotations are tried in each node. Because of the tremendous amount of different possible choices, it is not possible to inspect them all. Therefore the breadth λ is introduced to restrict the number of solutions kept from each depth. Often a limited time is available when solving problems. If this is the case a single value for the breadth is not desirable. If λ is chosen too low the search will finish prematurely, if it is chosen too high the search will never reach the bottom of the tree. Therefore a dynamic breadth is proposed. This breadth changes its value based on estimates of how much time it will take to finish the search. Different weighting strategies are used to control in which depth of the tree more time should be spent. Another improvement to the tree search was proposed in Christensen and Rousøe (2007), where a part of the greedy solution is kept in the partial solution. This will be denoted accelerated tree search and proved successful when there is only one customer in the problems.

3.1 The greedy algorithm

As both the space choice and the evaluation of each insertion depend on the greedy procedure, this procedure becomes fundamental to the overall tree search. The generic framework of the method is shown in Algorithm 1. The deciding point in the algorithm is how line 4 is performed.

Algorithm 1: GREEDY

```
1: repeat
2:    Choose subset of B_nPlaced
3:    while More boxes can be placed do
4:       Find feasible box/rotation/space combination
5:       Insert found box in the found space
6:       Update empty spaces and boxes
7:    end while
8: until stop
```

Before entering the while loop, the option to work on a limited number of customers at a time, is given. This can help control the packing sequence more thoroughly. If it is chosen to load boxes from one customer at a time - starting with the customer visited last and ending with the one visited first - this will indeed be a LIFO (Last-In-First-Out) packing. This follows the ideas introduced by Bischoff and Ratcliff (1995) and Gendreau et al. (2006a) to accommodate for multi-drop constraints. The minimum number of boxes present in the subset of $B_{nPlaced}$ used when running the inner loop of the greedy algorithms is called κ. The point of splitting the batch in smaller subsets should be not to mix up boxes from different customers. Loading one customer at a time, however, could be too strict.

The algorithm stops if: 1) all boxes are packed, 2) no boxes in the current subset of $B_{nPlaced}$ fit in the remaining empty spaces or 3) there are no more empty spaces.

The greedy method has a very simple strategy when choosing the next box to place. It depends on that the boxes are sorted with descending order by sequence and volume, sequence and biggest surface, or sequence and largest dimension. Also the sequence in which the different feasible box rotations are tried is important. Six different strategies are tested. The first favours the biggest dimension placed along the x-dimension, the middle dimension along the y-dimension and the smallest dimension along the z-dimension. This is denoted XYZ. The second favours the biggest dimension along the x-dimension, the middle one along the z-dimension and the smallest along y-dimension, which is denoted XZY. The same principle make up the last four rotation sequences, which are denoted YXZ, YZX, ZXY and ZYX. Finally also the empty spaces are sorted. This is done according to their position in the container, starting with the spaces with low $\underline{x}, \underline{y}$ and $\underline{z}$ in that sequence.

The procedure chooses the first space and then tries to find a box that can be placed in it, starting with the most promising box according to the chosen box sort rule. When a feasible combination have been found it is chosen and the box is inserted. This makes this greedy methods extremely fast, compared to evaluating all boxes with all possible rotations in all possible spaces. The possibility to fix the box and rotation is given. Thereby a greedily chosen space

to the combination is found.

3.2 Dynamic breadth

As mentioned, the choice of λ is crucial for the performance of the tree search. A reasonable choice depends on both the problem at hand and the time available to solve it. Partly because of this and partly because it could be a good idea to change the breadth throughout the search, a dynamic breadth is introduced.

The dynamic breadth is calculated on the fly, based on the remaining time, Δt, and the estimated time necessary to get to the bottom of the search tree through one branch $t_{branch,PS}$. This leads to the following value of the breadth in depth p:

$$\lambda_p = \frac{\Delta t}{t_{branch,PS}} \cdot \mathcal{W} \tag{1}$$

where $\mathcal{W}$ is a weight which will be explained in details later in this section. The time used in one branch of the tree is not trivially estimated, as it depends on many factors varying independently. The most important of these are the number of different boxes possible to place, the possible rotations of these, the number of spaces in the container and the depth of the branch. All these factors cannot be taken into account and instead a simplified estimate is used.

The time t_{PS}, it takes to expand one partial solution PS in the tree is estimated to be linear dependent on the number of not yet loaded boxes $|B_{nPlaced,PS}|$, and given by:

$$t_{PS} = K \cdot |B_{nPlaced,PS}|. \tag{2}$$

To expand a partial solution all feasible combinations of box types and rotations are evaluated by the greedy method. t_{PS} and $|B_{nPlaced,PS}|$ are estimated based on the average time used to develop all the possible solutions and the average number of boxes not loaded in each node. The expansion time varies between the different partial solutions, mainly because different boxes are available. Based on t_{PS} and $|B_{nPlaced,PS}|$, K can be found. To make a full solution, many boxes need to be placed. The best estimate on how many boxes should be placed, is the number of boxes placed in the best found solution so far. The number of not placed boxes in the best know solution is denoted $|B^*_{nPlaced}|$. The total time $t_{branch,PS}$ needed to get from a partial solution PS to the leafs in the tree, can now be found as the integral:

$$t_{branch,PS} = \int_{|B^*_{nPlaced}|}^{|B_{nPlaced,PS}|} K \cdot x \, dx \tag{3}$$

Thereby the breadth λ_p in each depth is estimated as:

$$\lambda_p = \left\lfloor \frac{2\Delta t \cdot |B_{nPlaced,PS}|}{t_{PS} \cdot \max\left(|B_{nPlaced,PS}|^2 - |B^*_{nPlaced}|^2, 3\right)} \cdot \mathcal{W} \right\rfloor \tag{4}$$

If more or the same number of boxes are placed in the partial solution than in the best solution found so far, the difference in the denominator in Equation (4) becomes non-positive. This is not desirable. The difference "describes" how many boxes we need to place before we are at the leaf of the branch. If we have not found a solution where all boxes are placed this is at least one more. Therefore we take the larger number of 3 and the difference. The new breadth

λ_p is sensitive to changes in the time measure t_{PS} and fluctuations in this time could change the breadth unwanted. Hence, it is chosen that λ_p can change with maximum 20% compared to λ_{p-1}. If the breadth is too small to change at all (for instance $\lambda_{p-1} = 2$) then it is allowed to change the breadth with ± 1. Of course $\lambda_p \geq 1$ most hold at all times.

When building the search tree, it is not necessarily best to have the same breadth in the start and the end. When only a few boxes are placed in the container, it is difficult to estimate which choice of placement for the next box is best. Therefore a tree that narrows in during the execution could lead to better solutions than an equally weighted tree. To test this, a weight $\mathcal{W}$ is introduced in Equation (1). It is determined as:

$$\mathcal{W} = \max\left(\left(2 - \frac{|B_{placed,PS}|}{|B^*_{placed}|}\right)^{\psi}, 1\right) \tag{5}$$

where $\psi \in \mathbb{R}$. If $\psi = 1$ the weight $\mathcal{W}$ will in the beginning be close to 2 and the chosen breadth is thereby twice as wide as the estimated breadth. In the end the weight is close to 1 and not influencing the estimated breadth. By changing ψ the impact of the weight can be adjusted. If more boxes are placed in the partial solution we are looking at than in the best found solution, the expression $\left(2 - \frac{|B_{placed,PS}|}{|B^*_{placed}|}\right)$ will evaluate to a value smaller than 1. If this happens we change the breadth unwanted, therefore the smallest allowed $\mathcal{W}$ is set to 1.

The dynamic breadth makes the tree search more generic. Without user intervention and business knowledge it is possible to solve problems of very different nature with the same setting for the tree search. At the same time all time available is used and we are able to get to the bottom of the search tree. This makes the dynamic breadth well suited for problems where a limited (and strict) time is available. The concept of the dynamic breadth does not only apply to container loading problems but to tree search algorithms of any kind, where a single value for the breadth of the tree is not appropriate.

4 Numerical experiments

To test the proposed algorithm different time horizons are used. When solving benchmark problems from the OR-Library 60 seconds are used as a time limit. On the new real life problems only 10 seconds are used as the time limit. These relative short times should reflect the need for fast solutions when for instance the problems are solved in a vehicle routing context. The best settings for the algorithm is using rotation sequence strategy YXZ, which favours the largest dimension of the box along the y-dimension, the middle dimension along the x-dimension and the smallest dimension in the z-dimension. The boxes should furthermore be sorted by their sequence and volume. Additionally when more customers are considered, the greedy procedure should only consider boxes from a single customers at the time, i.e. $\kappa = 1$. Finally, based on preliminary experiments, the breadth weight factor ψ is set to 0. Further details can be found in Christensen and Rousøe (2007). The algorithm is implemented in C++ and all tests are performed on Linux machines with 2,4GHz 64 bit AMD processors and 2GB RAM.

4.1 OR-Library problems

A number of CLP instances are found in the OR-Library. These problems were first used by Bischoff and Ratcliff (1995). There exists 7 distinct files, which we will denote *data groups*, each containing 100 CLP instances. Every data group is distinguished by the number of different box types present in the CLP instances. In the problems from the first data group (BR1), there are three different box types. This increases to 20 different box types in the last data group (BR7). These problems are used to test the performance of the algorithm and the effect of the multi-drop constraint. The latter is accomplished by randomly assigning a customer number to each box. The number of customers in a problem is denoted nC. When solving these problems we demand that all placed boxes have full support from below, i.e. $\gamma = 0$. Furthermore, in instances with more than one customers the only way out of the container is given by the direction X^+.

In Figure 2 the results of the test are shown. On the horizontal axis the number of customers in the problems, are shown. Each line represents results from distinct data groups. A clear connection between the number of customers

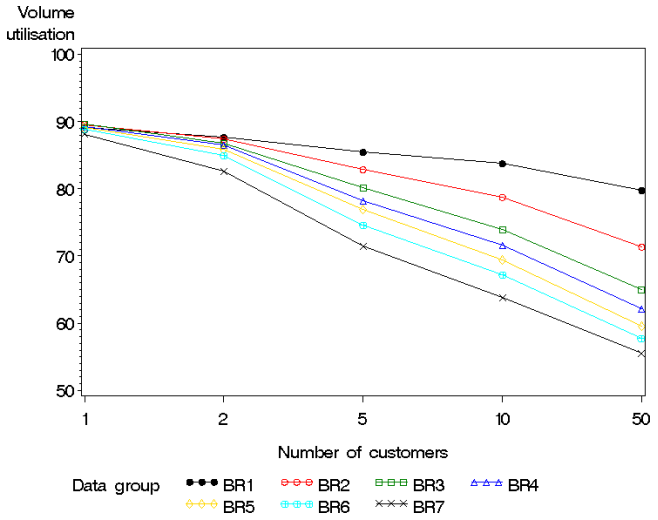

Figure 2: The results dependencies on the number of customers and the number of box types.

and the solution quality can be seen. The figure, furthermore, reveals that the solution quality depends not only on this, but also on the combination of the number of box types and the number of customers in the problem. When only 3 box types exist (BR1) the solution quality only drops 5% going from 1 to 50 customers, when 20 box types exist (BR7) the difference is approximately 30%.

It is not surprising, that the difficult problems both have many customers and box types. When the load is only weakly heterogeneous the multi-drop constraint will have little effect, as several boxes of each box type are available to all customers, leaving much freedom when choosing each box. Moreover, also the effect of full support from below becomes more serious when we are forced to load boxes in a certain sequence given by the customer numbers of the boxes.

The algorithm proposed in this paper have not been designed to solve the problems from the OR-Library without customers. A tuning to make a more fair comparison with other approaches have, however, been carried out. It was found that better performance is obtained if using an accelerated tree search in combination with a greedy method based on Bischoff and Ratcliff (1995) method for multi-drop problems. Also the breadth weight factor is changed. Now the best value is $\psi = 2$, which indicates that more time should be used in the top of the tree.

The greedy method proposed by Bischoff and Ratcliff (1995) evaluate all possible insertions against each other, choosing the one which utilise a stack in a space the best way. The accelerated search keeps a part of the greedy solution, which is tightly packed. In this way this method is able to get faster down through the search tree. This approach is elaborated in Christensen and Rousøe (2007). The results obtained with this setting is reported in Table 2. The test shows an average volume utilisation of 90,5%, looking across all data

Table 2: Test of OR-Library problems with one customer.

Data group	Min.	Avg.	Max.	SD.
BR1	83,38	90,48	96,81	2,70
BR2	85,31	91,16	94,93	1,89
BR3	87,77	91,33	94,97	1,53
BR4	87,91	90,98	94,09	1,34
BR5	86,41	90,83	93,25	1,32
BR6	87,03	90,19	93,12	1,26
BR7	84,81	88,36	91,48	1,27
All	83,38	90,48	96,81	1,93

groups. The best results obtained are for BR3, with 8 different box types, where the average is 91,3%. The worst results are obtained for BR7 (20 box types) with 88,4%. The data group that vary most in solution quality is BR1, with the highest standard deviation and also containing both the overall best and worst results.

The overall average of 90,5% is indeed better than the results reported in Figure 2, where the average for one customer is 89,1%. As seen in Table 3, this

Table 3: Average volume utilisations and difference for the two methods, **One** and **More**

nC	Method		Diff.
	One	More	
1	90,48	89,07	1,41
2	83,33	85,96	-2,63
5	72,10	78,52	-6,42
10	65,25	72,64	-7,39
50	56,16	64,45	-8,29
All	73,46	78,13	-4,67

however, is only in the case with one customer. The table shows the results obtained when the method is used on problems with more customers, along with

the results obtained with the best method when more customers are present. Method **One** is the method that is better when one customer exists, method **More** is the method which is better when more customers exist. In the last column, the difference between obtained results, is shown. Even though the gain for the one customer problems is notable, it is also obvious that the method behaves badly, when applied to problems with more customers. This is partly because of the accelerated methods, which does not consider the customer numbers on the boxes and partly because of the slower greedy method.

The results for one customer can be compared to work done by other authors. A selection of results obtained by others, can be seen in Table 4 together with our results. Common for all the algorithms is that full support of the boxes are required. When comparing our algorithm with others, it is important to

Table 4: Solution quality compared to other authors.

Authors	Avg.
Bischoff and Ratcliff (1995)	84,94
Bortfeldt and Gehring (2001)	90,06
Eley (2002)	88,75
Bortfeldt et al. (2003)	92,20
Moura and Oliviera (2005)	89,73
Lim and Zhang (2005)	91,81
Our method	90,48

remember that our method is not particularly created to solve problems from the OR-Library. It should be able to handle them, as well as more complex problems. The time limit used is also a consideration. We have set the time limit to 60 seconds. Except for the algorithm proposed in Bischoff and Ratcliff (1995) all other implementations use more time. The results obtained by Bischoff and Ratcliff (1995) are the combined method of two single construction method, which takes very little time to evaluate. The results by Bortfeldt, Gehring, and Mack (2003) and Eley (2002) are obtained, using a time limit of 10 minutes. In the genetic algorithm from Bortfeldt and Gehring (2001) a time limit of 500 seconds is used. Moura and Oliviera (2005) reports that average solutions times are under 64 seconds, when they perform 200 iterations of their GRASP approach. The solution times reported by Lim and Zhang (2005), spans between 1 and 4600 seconds.

Our results are not fully competitive with the best methods developed to solve the OR-Library problem instances. However, within a reasonable time limit an average volume utilisation above 90% are obtained.

4.2 Distribution of construction products

From a builders merchant eight problems have been gathered. These have all been solved by the company, indicating that it is possible to find optimal solutions to the consignments. Besides the eight problems given from the company, nine new random consignment, based on the data from the original eight problems, are constructed. This gives the possibility to test the algorithm when it is not given, that all boxes should fit in the container. In the problems, relatively few boxes are present. However, the load bearing strength of the boxes are very

different, which along with the multi-drop constraints make these problems difficult. For these trucks it is allowed to unload boxes in the directions given by X^+, Y^+ and Y^- and overhang are allowed with $\gamma = 0, 2$. The results obtained on the 17 problems are summarised in Table 5.

Table 5: Results obtained with the algorithm on problems from a builders merchant.

	Problem	No. of customers	Volume ratio	Volume utilisation	Time s	Tot. no. of boxes	No. of boxes left out
Original	1	2	16,47	16,47	0,00	3	0
	2	1	5,47	5,47	0,00	3	0
	3	2	20,17	20,17	0,00	9	0
	4	3	18,11	18,11	0,00	22	0
	5	4	15,74	15,74	0,00	8	0
	6	2	7,93	7,93	0,00	8	0
	7	1	51,72	51,72	0,00	11	0
	8	6	71,21	71,21	0,00	19	0
Generated	9	4	39,17	35,23	8.64	25	1
	10	4	83,68	61,97	3.94	31	16
	11	9	52,06	42,18	6.63	22	5
	12	5	76,11	58,37	3.43	22	4
	13	6	75,80	59,35	1.82	25	6
	14	5	31,54	31,54	1.26	20	0
	15	7	69,73	55,94	6.61	28	9
	16	7	40,51	32,11	1.48	20	4
	17	4	28,10	24,16	0.02	17	1

For the original problems most of the best solutions were found in the first node of the tree, but for two problems, the solutions were found in a depth of 2 and 4, respectively.

Figure 3 shows an optimal solution to problem 8. Two sub figures are shown.

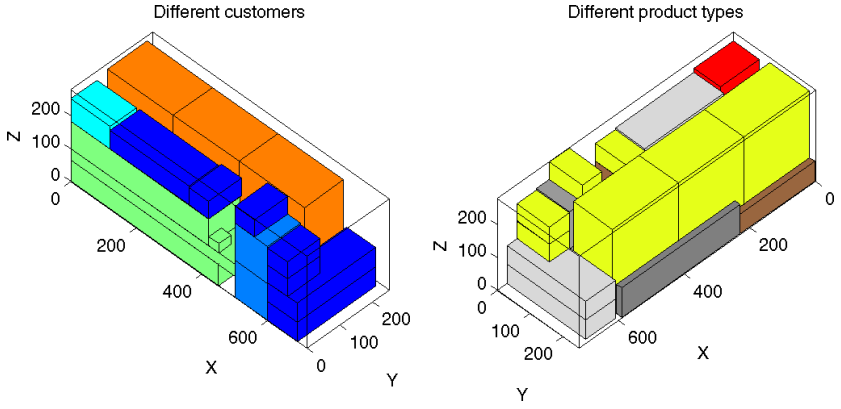

Figure 3: An optimal solution to problem 8.

The left one shows the sequence of the boxes, whereas the right one shows the different product types. The depicted solution is optimal and has a volume utilisation of 71%. There are 6 customers and 19 boxes. Information about which colour corresponds to which customer and all the products and the truck, can be seen on Table 6. The list shows that very different products are placed on the same load, both big pallets of rockwool, stacks of gypsum boards and

long tree poles. This indicate that a key factor is to be able to measure the load bearing strength of the products. A ton of stone cannot be placed on top of some tiles for the bathroom.

Table 6: List of products corresponding to the packing shown on Figure 3.

Customer	#	Dimensions *cm*	Rotations	Weight *kg*	LBS kg/cm^2	Description
-	-	$720 \times 250 \times 280$	-	9500*	-	The truck
1 (Dark blue)	3	$240 \times 90 \times 58$	$0 \times 0 \times 1$	777	0.1	Gypsum boards
1	1	$360 \times 90 \times 16$	$0 \times 1 \times 1$	193	0.02	Gypsum bars
1	4	$100 \times 60 \times 53$	$1 \times 1 \times 1$	3	0.0002	Rockwool
2 (Light blue)	2	$100 \times 100 \times 113$	$0 \times 0 \times 1$	900	0.1	Founda. blocks
3 (Turquoise)	1	$120 \times 80 \times 70$	$0 \times 0 \times 1$	250	-1	Tiles
4 (Green)	1	$420 \times 95 \times 83$	$0 \times 0 \times 1$	1500	0.1	Tree poles
4	1	$450 \times 112 \times 65$	$0 \times 0 \times 1$	1500	0.1	Tree poles
4	1	$500 \times 85 \times 35$	$0 \times 1 \times 1$	100	0.1	Wood boards
4	1	$38 \times 31 \times 29$	$1 \times 1 \times 1$	5	-1	Cardboard box
5 (Yellow)	1	$228 \times 60 \times 16$	$0 \times 1 \times 1$	40	0.03	Flooring
6 (Orange)	3	$200 \times 120 \times 265$	$0 \times 0 \times 1$	75	-1	Rockwool

* *The capacity of the truck*

The results on the original problems show, that the algorithm is able to solve the problems corresponding to the loads actually driven by a builders merchant within a very short time. This shows that the model is flexible enough to allow realistic loads to be packed. Looking on the results obtained from the generated problems, it is seen that a larger amount of the container is utilised. This is a natural consequence of the characteristics of the problems, where a larger box/container volume ratio is present. However, in only one of the generated problems all boxes are placed.

The results show, that even for the problems with only around 30% box/-container volume ratio, all boxes cannot be placed. This shows that for problems of this type, with many highly odd sized boxes, with very different characteristics, both concerning load bearing strength and customer numbers, the volume ratio alone is not a very good indication of how hard a problem is to solve.

In Figure 4 the best found solution to problem 9 is shown. The problem has 4 customers and a total of 25 boxes. The volume utilisation is 34%. In the solution, a 5,4 metre long box with wood has not been packed. It is easy to see that the remaining box, almost going from one end of the truck to the other, is hard to place without neglecting the sequence constraint.

5 Conclusion

When solving real world problems like the ones encountered by builders merchants who distribute construction products to construction sites, the need to incorporate both load bearing strength and multi-drop constraints to the ordinary CLPs are essential. Without these considerations products may get damaged during transport and inappropriate routes are driven when reallocating the products are not possible. A tree search with a dynamic breadth is proposed to solve the problems which are very different in characteristics. The dynamic breadth turnes out to be a valuable improvement to the tree search, as it makes sure that the available time is used most profitable. The results are very promising as optimal solutions to real world problems are obtained in

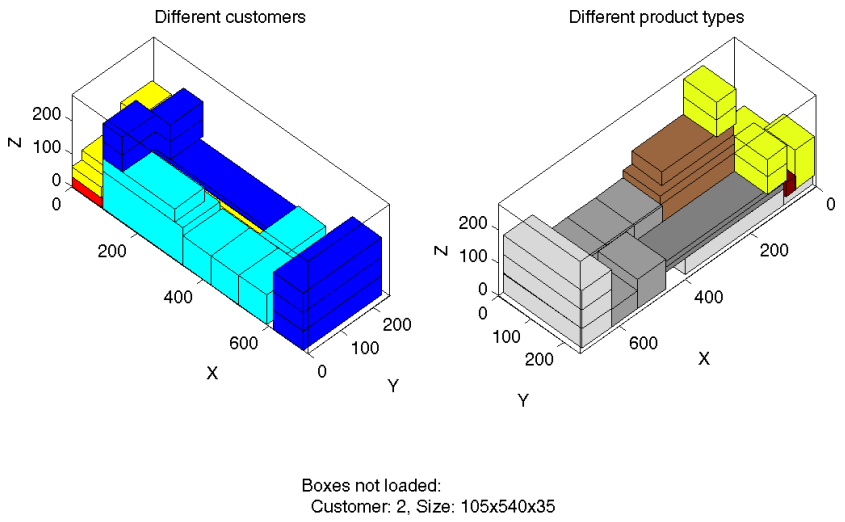

Figure 4: The best found solution to problem 9.

fractions of a second. Results obtained on benchmark problems indicate that the algorithm performs comparable with other more specialised methods.

References

E.E. Bischoff. Three-dimensional packing of items with limited load bearing strength. *European Journal of Operations Research*, 168:952–966, 2004.

E.E. Bischoff and M.S.W. Ratcliff. Issues in the delevopment of container loading problem. *Omega*, 23:277–390, 1995.

A. Bortfeldt and H. Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operations Research*, 131:143–161, 2001.

A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container loading problem. *Parallel Computing*, 29:641–662, 2003.

C.S. Chen, S.M. Lee, and Q.S. Shen. An analytical model for the container loading problem. *European Journal of Operations Research*, 80:68–76, 1993.

S. G. Christensen and D. M. Rousøe. Container loading with multi-drop constraints. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2007. http://www2.imm.dtu.dk/pubdb/p.php?5225.

A.P. Davies and E.E. Bischoff. Weight distribution considerations in container loading. *European Journal of Operations Research*, 114:509–527, 1999.

K. Doerner, G. Fuellerer, M. Gronalt, R. Hartl, and M. Iori. Metaheuristics for the vehicle routing problem with loading constraints. *Networks*, 49:294–307, 2007.

H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operations Research*, 44:145–159, 1990.

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

H. Gehring and A. Bortfeldt. A genetic algorithm for solving the container loading problem. *International Transactions in Operations Research*, 4:401–418, 1997.

M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40:342–350, 2006a.

M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, to appear, 2006b.

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

P.C. Gilmore and R.K. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

J.C. Herz. Recursive computational procedure for twodimensional stock cutting. *IBM Journal of Research and Development*, 16:462–469, 1972.

M. Iori, J.J. Salazar Gonzalez, and D. Vigo. An exact approach for the vehicle routing problem with two dimensional loading constraints. *Transportation Science*, 41:253–264, 2007.

A. Lim and X. Zhang. The container loading problem. In *ACM Symposium on Applied Computing*, 2005.

D. Mack, A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorihtm for the container loading problem. *International Transactions in Operations Research*, 11:511–533, 2004.

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

R. Morabito and M. Arenales. An AND/OR-graph approach to the container loading problem. *International Transactions in Operations Research*, 1:59–73, 1994.

A. Moura and J.O. Oliviera. A GRASP approach to the container-loading problem. *Transportation and Logistics*, pages 50–57, 2005.

B.K.A. Ngoi, M.L. Tay, and E.S. Chua. Applying spatial representation techniques to the container packing problem. *International Journal of Production Research*, 32:111–123, 1994.

D. Pisinger. Heuristics for the container loading problem. *European Journal of Operations Research*, 141:382–392, 2002.

M.S.W. Ratcliff and E.E. Bischoff. Allowing for weight considerations in container loading. *OR Spektrum*, 20:65–71, 1998.

G. Scheithauer. LP-based bounds for the container and multi-container loading problem. *International Transactions in Operations Research*, 6:199–213, 1998.

G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operations Research*, to appear, 2006.