A matheuristic framework for the Three-dimensional Single Large Object Placement Problem with practical constraints

Everton Fernandes Silva[a,*], Aline Aparecida Souza Leão[a], Franklina Maria Bragion Toledo[b], Tony Wauters[a]

[a] CODeS, Department of Computer Science, KU Leuven, Gebroeders de Smetstraat, 9000, Gent, Belgium

[b] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Avenida Trabalhador São-Carlense, 13560-970, São Carlos, SP, Brazil

Abstract. The Three-dimensional Single Large Object Placement Problem consists of a set of weakly heterogeneous items that must be placed inside a single larger object without overlapping each other. Many constraints can be considered depending on the practical specifications of the problem being solved, such as orientation, stability, weight limit and positioning. Although this is a well-known problem which has received considerable academic attention, most of the research limits itself to considering only three basic constraints: non-overlap, orientation and stability of the placed items. Recent literature concerning the problem has indicated that there is a pressing need for solution methods which consider a more realistic number of sets of practical constraints given that it is very uncommon to find real-world situations where only a few of these constraints are considered together. Therefore, this paper introduced a well-performing matheuristic framework which considers multiple practical constraints: orientation, load balance, loading priorities, positioning, stability, stacking and weight limit. An extension of the developed method considering multiple containers is also discussed. Results are compared against the state of the art from the literature and demonstrate the robustness of the proposed matheuristic with respect to different combinations of constraints.

Keywords. 3D Packing, matheuristic, practical constraints

*Corresponding author.

Email address: everton.fernandesdasilva@kuleuven.be (Everton Fernandes Silva)

1 Introduction

Container Loading Problems (CLP) are a set of well-known combinatorial optimization problems that are of crucial relevance in the logistics industry. Their significance is threefold in nature: company profitability, employee safety and environmental impact. Considering all the categories proposed by Wäscher et al. (2007), this paper focuses on solving the Three-dimensional Single Large Object Placement Problem (SLOPP). This variant concerns a set of weakly heterogeneous items (boxes) and a single larger object (container). The SLOPP is an output maximization problem which aims to maximize the value, in our case the volume, of the placed boxes. These boxes must be placed completely inside the container and they cannot overlap with each other.

Bortfeldt and Wäscher (2013) have provided a very thorough review of all sets of constraints currently considered in the literature of 3D Container Loading Problems, listing which constraints can be found in over 150 papers from the literature up until the year 2012. The most significant conclusion drawn by the authors concerns the lack of papers that consider an acceptable number of practical constraints, something which would make them more comparable to the problems found in the industry. In their survey regarding all the 3D Packing Problem categories, 74.9% of the publications consider at most two sets of practical constraints, with the majority being orientation and stability. The most sets of constraints which were accommodated was seven, featuring in only three papers, 1.8% of the publications, which are Bortfeldt and Gehring (1999), Egeblad et al. (2010) and Olsson et al. (2020).

Bortfeldt and Gehring (1999) proposed a genetic algorithm to handle different combinations of sets of practical constraints to place regular items inside a single container, while Egeblad et al. (2010) presented a solution approach for the problem of loading a container with pieces of furniture (irregular items). More recently, Olsson et al. (2020) introduced a specific real-world problem where regular and irregular items are first packed into stacks and then placed in a container. Due to the unique problem specification associated with each of these three papers, each considers a different set of constraints.

These three papers address the 3D Single Knapsack Problem (SKP), a counterpart of the SLOPP and the most addressed container loading variant in the literature. The only difference between the SLOPP and the SKP is that while the former considers a set of weakly heterogeneous items to be placed in a single container, the latter consists of a set of strongly heterogeneous items. Although this difference might seem negligible, solution approaches developed taking into

account the details of the items can generate higher quality solutions than those which treat all identically. Taking this into consideration, we limit the scope of our research to container loading problems with weakly heterogeneous items.

Liu et al. (2011) proposed a problem variant with five sets of practical constraints, which is the largest number of sets of practical constraints considered found in the literature for the SLOPP. The authors proposed a hybrid tabu search approach which considers orientation, stability, connectivity, weight limit and weight distribution constraints that are evaluated by solving real-world data. Other important works in the literature concerning the SLOPP in which multiple sets of practical constraints accommodated are: Eley (2002), Hifi (2002), Bischoff (2006), Christensen and Rousøe (2009), Fanslau and Bortfeldt (2010), Ren et al. (2011), Junqueira et al. (2012), Zhu and Lim (2012), Araya and Riff (2014), Moon and Nguyen (2014), Araya et al. (2017), Layeb et al. (2017) and Saraiva et al. (2019). Most recently Ramos et al. (2018) have introduced a more realistic type of weight distribution constraint, known as load balance constraints.

With this lack of approaches in the literature with a realistic set of practical constraints, our objective is to propose a well-performing matheuristic method which is able to solve problems with a considerable number of practical constraints and guarantee consistent results of good quality. We consider seven different sets of practical constraints: orientation, load balance, loading priorities, positioning, stacking, stability and weight limit. The proposed solution method consists of a wall-building approach composed of a greedy improvement heuristic and two mathematical formulations which divide between themselves the computational effort associated with considering all of the practical constraints. The method is structured in such a way that it is clear where each constraint is included and where extra constraints could be added.

To demonstrate the simplicity with which the method can be adapted to other problem variants, we also address the Single Stock-Size Cutting Stock Problem (SSSCSP), which is a multiple container variant of the SLOPP consisting of a set of identical containers where the objective is to minimize the number of containers used to place all (weakly heterogeneous) items. The results from the computational experiments concerning both SLOPP and SSSCSP problems verify the robustness and flexibility of the approach, showing how it is capable of generating high quality solutions for the tested instances when combining different sets of practical constraints.

The remainder of the paper is structured as follows. Section 2 details the sets of practical constraints considered. A description of the solution method is provided in Section 3. Computational results evaluating the efficiency and adaptability of the method are presented in Section 4, while in Section 5, we detail its adaptation to the SSSCSP and validate its performance with instances from the literature. Finally, conclusions and future research directions are presented in Section 6.

2 Practical Constraints

The practical constraints considered in this paper conform to the definitions provided by Bortfeldt and Wäscher (2013):

Orientation constraints define how many orientations and which ones are allowed for each box type. Considering that all boxes are regular cuboids and their placement is orthogonal to the container axis, up to six orientations are allowed.

Stability constraints can be divided into two types: *vertical stability* guarantees each box has a minimum percentage of its base completely supported by either the container's floor or other boxes, while *horizontal stability* ensures that each box is supported on all four sides by either other boxes or the container's walls.

Stacking constraints guarantee that no box is damaged by carrying more than its maximum load (fragility). The most well-known fragility concept considered is the one which a box type cannot bear any load above it, allowing it to be only placed above other boxes in order to not be damaged.

Weight limit constraints limit the combined weight of the boxes placed inside the container to a predefined maximum.

Loading priorities constraints prioritize types of boxes that must have their full demand placed in the container.

Positioning constraints ensure that boxes of type t must be placed such that they respect a specific position relative to boxes of type s (*in front of*, *behind*, *to the left of* or *to the right of*).

Load balance constraints are more precise than the weight distribution considered in Bortfeldt and Wäscher (2013). They are based on the characteristics of the vehicle transporting the container. Each vehicle has its own load distribution diagrams which define the feasibility domain for the location of its cargo's center of gravity. These diagrams are created in accordance

with national, regional and continental legislation and they should detail: the maximum front axle load, rear axle load and gross vehicle weight; the minimum required steering axle load as well as the driving/traction axle load.

3 Solution Method

The solution method can be summarized as a wall-based approach with a greedy improvement phase. It consists of two parts: an initial phase which quickly generates a first solution of reasonable quality followed by an improvement phase which aims to improve the first solution within a given time limit. This method has a structure composed of modules that were created in order to facilitate the removal or addition of constraints, as desired by the user (see Figure 1).

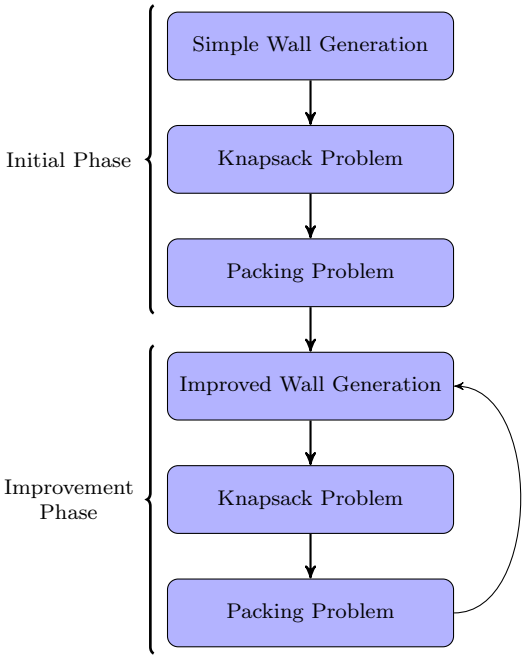


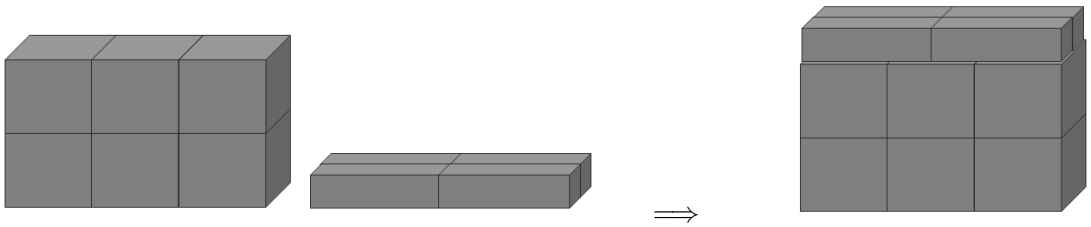
Figure 1: Structure of the solution method.

3.1 Wall Generation

The wall generation is based on the method of Zhu and Lim (2012). A summary of the procedure is detailed in the following paragraphs and readers interested in the full details can consult the complete algorithm in the original paper.

The algorithm begins by generating structures containing boxes of the same type, referred to as homogeneous blocks, and iteratively placing homogeneous blocks of different types together

in order to create more complex blocks. An illustrative example of how this works is shown in Figure 2. Walls are composed of any kind of box arrangements, starting from a homogeneous block with a single box until a combination of blocks. Due to this diversity, when considering the bounding box around each wall generated, only those which meet a certain level of occupancy percentage (*occperc*), are accepted, where *occperc* is defined as the volume of the boxes within the wall divided by the volume of the bounding box. These structures are created in such a way that all permitted orientations of the boxes are already considered and the boxes in each block have their bases fully supported. This guarantees that the **orientation** and **vertical stability** constraints are respected throughout the entire solution method. For horizontal stability, we assume that fillings can be placed in the container in order to guarantee that each box has its sides supported by other boxes, container's walls or the inserted fillings. The **stacking** constraints are considered after the creation of the walls: given the list of walls generated, any wall that violates the stacking constraint is removed from the list.



(a) Homogeneous blocks. (b) Complex block.

Figure 2: Complex block generation.

The first phase of the solution method generates what we will refer to as Simple Walls (Figure 1). These structures consist of walls containing at most two homogeneous blocks placed side by side along the Y-axis or one above another. Limiting the number of combinations of block types in this phase of the method is for speed purposes. Given a large-scale problem with more than 100 available boxes, generating a long list of good quality walls which combine an unlimited number of homogeneous blocks is very time consuming. At this stage of the solution method, the idea is to simply generate a fast initial solution of good quality that will be improved in the next stage of the method.

For the Improvement Phase, typically only a subset of boxes is considered to create new walls. Thus, more complex walls which contain more than two homogeneous blocks placed together are generated. Although considering a subset of boxes is less time consuming, the

total number of walls (*nwalls*) and the quality of the complex walls (*occperc*) accepted are also important factors which directly influence how much time is spent in this stage. A final means by which the Simple and Complex wall generation process is accelerated is by running the entire procedure in parallel.

3.2 Mathematical Formulations

Even when considering the simplest SLOPP with only non-overlapping and bounding constraints the mathematical formulations currently proposed in the literature struggle to solve medium-sized problems. The reasons for this are multiple, as reported in Silva et al. (2019). Including additional constraints to these formulations will continue to hinder their performance, negatively impacting solution quality. Therefore, our strategy is to divide the remaining sets of practical constraints into two separate formulations: one based on the Knapsack Problem and another based on the Packing Problem. The division is made in such a way that the first formulation functions as a filter, reducing the number of walls created into a small set of promising high quality walls which are then used as input to the second, more complex formulation. Each constraint is considered only where necessary so as to ensure the feasibility of the final solution. To avoid the need for any post-processing checks, some constraints are included in both formulations.

3.2.1 Knapsack Problem

The first formulation solved is based on the classical Knapsack Problem. Given a list of generated walls (Simple or Complex), the objective is to maximize the volume of the boxes placed in the container. The two basic constraints considered are that the sum of the length of the chosen walls cannot exceed the container's length and the demand for each box type must be met. The practical constraints included are **weight limit** and **loading priorities**. This changes the formulation from a General Knapsack Problem to a Multidimensional Knapsack Problem, which was first described in Lorie and Savage (1955) and Markowitz and Manne (1957).

The input data and variables employed within the formulation are detailed in Table 1, while the problem formulation itself is presented by (1)-(6):

Parameters:	
$\mathcal{I}$	set of walls.
$\mathcal{T}$	set of box types.
$\mathcal{T}'$	set of box types obligatory to be placed.
$\mathcal{D}_t$	total demand of box type t.
P_{max}	maximum permissible container weight.
L	length of container.
l_i	length of wall i.
v_i	volume of wall i.
m_i	weight of wall i.
d_{ti}	number of boxes of type t in wall i.
Variables:	
p_i	integer variables indicating the number of times wall i is used.

Table 1: Input data and variables.

$$\max \sum_{i \in \mathcal{I}} v_i \cdot p_i \tag{1}$$

$$\text{s.t.} \sum_{i \in \mathcal{I}} l_i \cdot p_i \leq L, \tag{2}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot p_i \leq P_{max}, \tag{3}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot p_i \leq \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T} \backslash \mathcal{T}', \tag{4}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot p_i = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}', \tag{5}$$

$$p_i \in \mathbb{Z}_+, \qquad \forall\, i \in \mathcal{I}. \tag{6}$$

Objective function (1) maximizes the total volume used in the container. Constraint (2) is the general knapsack constraints, which in this case ensure that the container length will be respected. Constraint (3) limits the maximum weight in the container. Constraints (4) and (5) ensure that the maximum demand for all boxes is not exceeded. Moreover, Constraints (5) ensure that the loading priority constraint is met for each box type $t \in \mathcal{T}'$. Finally, Constraints (6) set the variables' domains.

The solution of this Knapsack Problem provides a good estimate of which walls will be placed in the container and how many copies of each wall will be used. It is only considered to be an estimate given that this solution does not consider all the practical constraints. Therefore, when including the remaining constraints the current solution is not necessarily feasible. However,

should one only wish to solve the SLOPP while considering only weight limit and/or loading priorities constraints, then this solution is valid.

3.2.2 Packing Problem

The packing formulation, based on Chen et al. (1995), is mainly composed of the constraints that require information concerning the specific position of walls in the container: the **load balance** and **positioning** constraints. **Weight limit** and **loading priorities** are also included to ensure that the solution provided by this formulation is feasible for the entire problem considered by this paper. Since it is assumed that it is not possible to place a wall neither beside nor above/below another wall, the only variable associated with the coordinates of the walls is for the X-axis, which corresponds to the length of the container. Therefore, this is a one-dimensional packing problem, that aims to maximize the volume of the boxes placed. The basic constraints included guarantee that the chosen walls are positioned fully inside the container and they do not overlap one another.

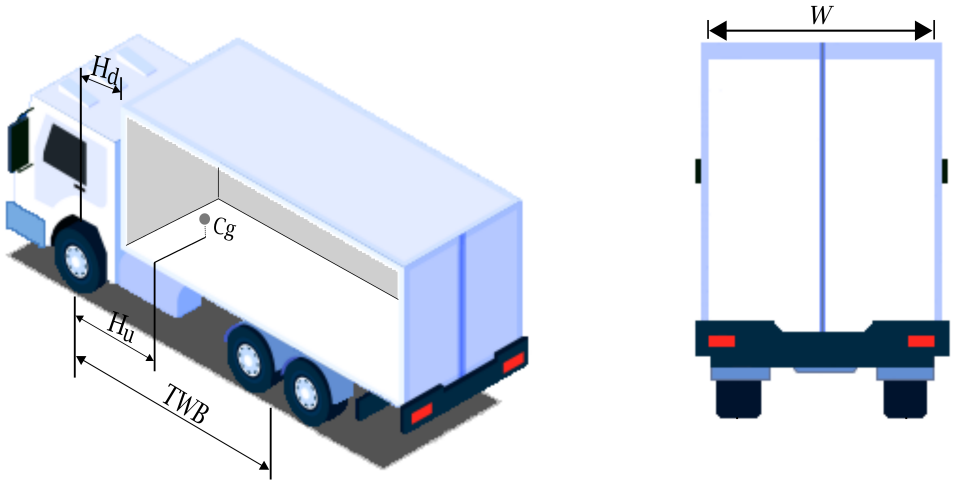


Figure 3: Truck measurements.

Figure 3 provides the necessary information from the truck in order to ensure that the load balance constraint is satisfied. The input data and supplementary variables employed within the formulation are detailed in Tables 2 and 3.

C_g	the truck's center of gravity.
U	weight of empty vehicle.
TWB	theoretical wheelbase - distance between front and rear horizontal center lines.
H_d	horizontal distance between the theoretical front-axle centre line and the back of the container.
H_u	horizontal distance between the theoretical front-axle centre line and the centre of gravity.
FT_{max}	maximum permissible front axle load.
RT_{max}	maximum permissible rear axle load.
S_{min}	minimum steering axle load.
T_{min}	minimum driving axle load.
t_1, t_2	boxes of type 1 and 2 such that $t_1, t_2 \in \mathcal{T}, t_1 \neq t_2$.

Table 2: Parameters.

x_i	continuous variable indicating the x-coordinate of wall i.
s_i	binary variable which is equal to 1 if wall i is placed in the container and 0 otherwise.
g_i	continuous variable indicating the location of the geometric centre on the X-axis of wall i in the container.
a_{ij}, b_{ij}	binary variables indicating the relative position of walls i and j (further from and closer to the container door, respectively).
P_{cargo}	continuous variables which is equal to the total weight in the container.
NC_x	continuous variable indicating the numerator of the quotient of the container's centre of gravity on the X-axis (Equation 28).

Table 3: Variables.

The problem formulation is presented by (7)-(27):

$$\max \sum_{i \in \mathcal{I}} v_i \cdot s_i \tag{7}$$

$$\text{s.t. } x_i + l_i \leq x_j + (1 - a_{ij}) \cdot L, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{8}$$

$$x_j + l_j \leq x_i + (1 - b_{ij}) \cdot L, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{9}$$

$$a_{ij} + b_{ij} \geq s_i + s_j - 1, \qquad \forall\, i, j \in \mathcal{I}, i < j, \tag{10}$$

$$x_i + l_i \leq L + (1 - s_i) \cdot L, \qquad \forall\, i \in \mathcal{I}, \tag{11}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot s_i \leq \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T} \backslash \mathcal{T}', \tag{12}$$

$$\sum_{i \in \mathcal{I}} d_{ti} \cdot s_i = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}', \tag{13}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot s_i = P_{cargo}, \tag{14}$$

$$P_{cargo} \leq P_{max}, \tag{15}$$

$a_{ij} = 0$ if wall i contains box type t_1, wall j contains box type t_2 $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

and a positioning constraint between box types t_1 and t_2 is active, $\qquad \forall\, i, j \in \mathcal{I}, i < j,$ (16)

$b_{ij} = 0$ if wall i contains box type t_2, wall j contains box type t_1 $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

and a positioning constraint between box types t_1 and t_2 is active, $\qquad \forall\, i, j \in \mathcal{I}, i < j,$ (17)

$$g_i \leq L \cdot s_i, \qquad \forall\, i \in \mathcal{I}, \tag{18}$$

$$g_i \leq x_i + \frac{l_i}{2}, \qquad \forall\, i \in \mathcal{I}, \tag{19}$$

$$g_i \geq x_i + \frac{l_i}{2} + L \cdot s_i - L, \qquad \forall\, i \in \mathcal{I}, \tag{20}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot g_i = NC_x, \tag{21}$$

$$NC_x \leq P_{cargo} \cdot (TWB - H_d) - TWB \cdot FT_{max} + U \cdot (TWB - H_u), \tag{22}$$

$$NC_x \leq TWB \cdot RT_{max} - U \cdot H_u - P_{cargo} * H_d, \tag{23}$$

$$NC_x \geq P_{cargo} \cdot (H_d - S_{min} \cdot TWB - TWB) + U \cdot (TWB - H_u - S_{min} \cdot TWB), \tag{24}$$

$$NC_x \geq U \cdot (T_{min} \cdot TWB - H_u) + P_{cargo} \cdot (T_{min} \cdot TWB - H_d), \tag{25}$$

$$s_i, a_{ij}, b_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in \mathcal{I}, \tag{26}$$

$$x_i, g_i, P_{cargo}, NC_x \geq 0, \qquad \forall\, i \in \mathcal{I}. \tag{27}$$

Objective function (7) maximizes the total volume used in the container. Constraints (8)-(10) guarantee walls do not overlap with one another. Constraints (11) ensure that all the chosen walls are placed fully inside the container. Constraints (12)-(15) are similar to those in the knapsack formulation and ensure that the demand, loading priority and weight limit constraints are satisfied, respectively. Constraints (16) - (17) ensure the positioning constraints between two box types are fully respected. To provide an example of how they should be modeled, walls containing box type t_1 must be placed in front of walls containing box type t_2.

Constraints (18)-(25) concern the load balance of the container. These constraints were adapted from Silva et al. (2018), the only paper in the literature that models this type of constraints, and are based on the formula of the moments of the forces of each specific area of the container. The moment of force formula corresponds to the turning effect of a force. It is the force being analysed multiplied by the perpendicular distance from the force's line of action to the pivot or point where the object will turn. Constraints (18)-(21) calculate the geometric centre on the X-axis, which combined with P_{cargo} determines the cargo's centre of gravity C_x on the X-axis:

$$C_x = \frac{NC_x}{P_{cargo}} \tag{28}$$

Since the walls are positioned in such a way that their center of gravity on the Y-axis is within the maximum permissible load difference between the left and right wheels, it is only necessary to guarantee the correct placement of the walls along the X-axis. Constraint (22) ensures the maximum permissible front axle load is respected and uses the formula of

moments of forces of the rear axle and the container's total weight. Constraint (23) ensures the maximum permissible rear axle load is respected. Constraint (24) is responsible for the minimum permissible steering axle load, while Constraint (25) ensures that the minimum driving/traction axle load is respected. Finally, Constraints (26)-(27) set the variables' domains.

It is worth noting that if we only consider the walls from the solution of the Knapsack Problem, the Packing Problem may result in a solution of worse quality given that the constraints considered in this formulation conflict with those present in the Knapsack Problem. To solve this issue, a more diverse set of walls is provided as input for this formulation. Therefore, in addition to the walls in the solution of the Knapsack Problem, an extra subset of walls (*extrawalls*) is provided. These walls are selected at random from the list of walls generated which were not included in the solution of the Knapsack Problem.

3.3 Improvement Phase

This phase consists of a greedy approach used to improve the initial solution (Algorithm 1). Given a solution, the idea is to initially remove the wall(s) with the lowest quality (line 6) and, considering the boxes that belong to the removed wall(s) together with those not considered in the solution (not placed in the container), to generate more complex walls with a greater diversity of boxes which will result an improved solution (line 7). The next iteration consists of solving both the Knapsack and Packing problems considering the new list of walls generated and the walls not removed from the previous solution (lines 8 and 9, respectively).

When an improved solution is found, the problem is updated and the procedure restarts the improvement phase by removing the worst wall of this new best solution (lines 11 and 12). If no improved solution is found, the first and second worst wall in the solution are removed in the next iteration, providing a bigger subset of boxes for the Improved Wall Generation. While no improvement is found this procedure is repeated always removing the next worst wall together with the previous ones (line 14). The method ends when the time limit is reached (line 5).

4 Computational Results

Experiments were performed to evaluate the method considering multiple combinations of practical constraints. Instances BR originally proposed by Bischoff and Ratcliff (1995) are utilized for this purpose. The BR instance set consists of 15 classes (BR1 - BR15) with each

Algorithm 1: Solution method.

Input: set of box types to be placed;
container dimensions and parameters (if load balance constraint is active);
Output: set of boxes placed in the container and their respective positions;
`// Initial Phase`
1 Generate *Simple Walls* considering all boxes from the input;
2 Solve the *Knapsack Problem* using the complete set of walls generated;
3 Solve the *Packing Problem* using the walls in the solution from line 2 and *extrawalls* chosen;
`// Improvement Phase`
4 n = 1;
5 **while** *time limit not reached* **do**
6 Remove the n worst wall(s) from the current best solution;
7 Generate *Improved Walls* with the removed boxes and the boxes not originally placed;
8 Solve the *Knapsack Problem* including the complete set of walls generated and the walls not removed in line 6;
9 Solve the *Packing Problem* including the walls in the solution of line 8, *extrawalls* and the walls not removed in line 6;
10 **if** *new best solution found* **then**
11 update current best solution;
12 n = 1;
13 **else**
14 n = n + 1;
15 **end**
16 **end**

class containing 100 instances. The first seven classes are characterized of having instances composed by weakly heterogeneous sets of boxes, while the remaining eight sets contain strongly heterogeneous sets of boxes. Since our goal is to solve SLOPP problems, we limit ourselves to instance classes BR1 to BR7. The sets gradually increase with respect to heterogeneity, meaning that BR1 contains the most homogeneous instances while BR7 contains the most heterogeneous instances.

More recently, Silva et al. (2018) and Ramos et al. (2018) recreated these instances to consider the boxes' weight. Further details concerning how these instances were updated can be found in Ramos et al. (2018). For our experiments, new information must be included in order to accommodate the additional constraints we consider. For example, the priority of a box to be placed in the container. Therefore, a more "complete" set of BR Instances was generated which has been made publicly available online at `http://dx.doi.org/10.17632/vcp6btht2h.1`

Concerning the parameters for wall generation, the maximum number of walls accepted (*nwalls*) is 10,000 and these walls must have a quality (*occperc*) of at least 90%. As for the additional walls (*extrawalls*) that are included in the Packing model (see Section 3.2.2), 50 walls are randomly chosen from the list of walls generated. These walls can be placed at most twice in the container, meaning that there can be up to 100 extra walls in total. The values for these parameters were chosen after preliminary experiments which sought to determine values which provided enough diversity for high quality solutions, but which did not introduce too many additional variables such that the performance of the mathematical model was negatively impacted. Section 4.1 provides more information concerning the choice of values for those parameters.

To evaluate the robustness of the method, experiments were conducted using different sets of constraints, as detailed in Table 4. Test cases 1 and 2 have already been considered in the literature with the same BR instances and therefore a comparison against the existing state of the art is available in Sections 4.2 and 4.3, respectively. The remaining experiments are presented in Section 4.4.

Test Case	Orientation	Stability	Weight Limit	Load Balance	Loading Priorities	Positioning	Stacking
1	✓	✓					
2	✓	✓	✓	✓			
3	✓	✓	✓	✓	✓		
4	✓	✓	✓	✓	✓		
5	✓	✓	✓	✓	✓	✓	✓

Table 4: Constraints considered in each set of experiments.

The computer used for the experiments was equipped with an Intel Xeon CPU E5-2860 @2.50GHz with 24 cores and 64GB of RAM. Experiments were executed with a runtime limit of 900s and in each iteration, each formulation, executed with IBM ILOG CPLEX 12.8, was limited to 60s. Wall generation was conducted in parallel, using all 24 available CPUs.

4.1 Sensitivity Analysis

To decide which parameter values to use in our method, some preliminary experiments with different values were necessary. To show how well the chosen parameters work we separated our

experiments into two groups: one concerning the quality of the walls accepted and one for the number of extra walls to be considered.

To evaluate the impact of the quality of the walls to be accepted (*occperc*), experiments were ran considering the first 10 instances of each instance set for test case 1 and test case 5, the two extremes in terms of number of constraints considered. Three different wall quality percentages were compared: 80%, 90% (our default value) and 99%. Figures 4 and 5 illustrate the performance of the two test cases considering these different values.

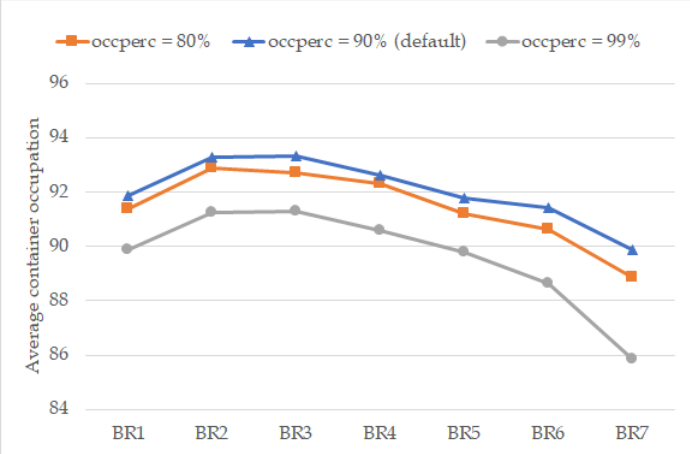


Figure 4: Impact of the wall quality parameter for test case 1.

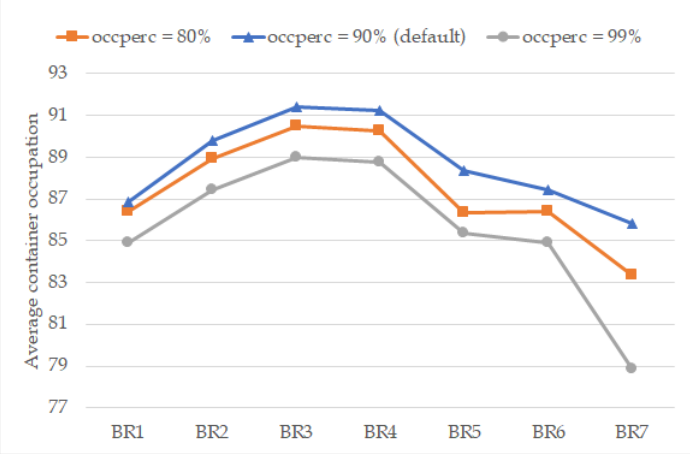


Figure 5: Impact of the wall quality parameter for test case 5.

The first observation worth noting is that considering walls with 80% or 99% of quality the solutions obtained are always worse than 90%. As new constraints are included, the gap comparing *occperc* = 90% against 80% and 99% gradually increases, until by the time all

constraints are considered, test case 5 (Figure 5), with their gaps having their most significant values. Therefore, given the results obtained across all previous test cases and the similarities in the last test case, we decided to use $occperc = 90\%$.

For the second evaluation, the number of additional walls that should be included for the packing model, we consider test case 2 and test case 5. Test case 1 is discarded from these tests since it only considers orientation and stability constraints which, as already mentioned, does not require solving the Packing model and therefore no extra walls are ever considered. Following the same idea as before, two different numbers of walls were evaluated for the first 10 instances of each instance set: 50 and 100 different extra walls. Figures 6 and 7 illustrate the performance for these parameters.

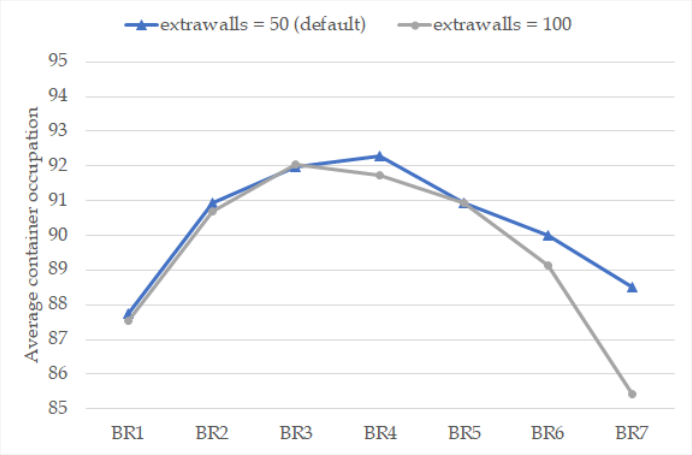


Figure 6: Impact of the extra walls parameter for test case 2.

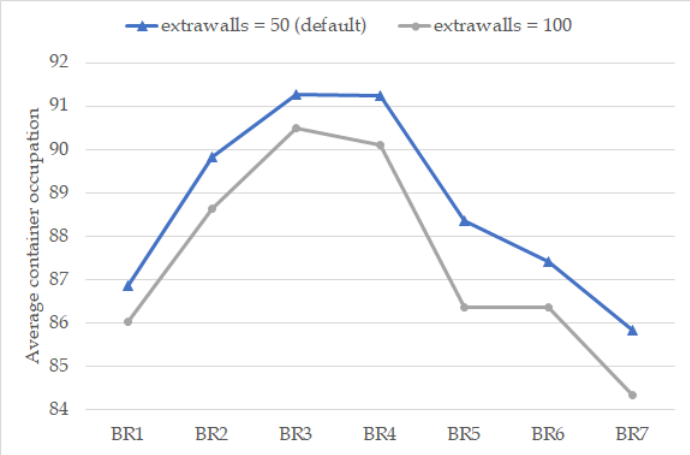


Figure 7: Impact of the extra walls parameter for test case 5.

For these experiments it is evident that considering 50 different extra walls performs best, particularly for test case 5. For test case 2, even though the results from BR3 and BR5 are, on average, better for *extrawalls* = 100 (0.10pp and 0.01pp better, respectively), better solutions are achieved for the remaining instance sets when considering *extrawalls* = 50. This is particularly evident in the last two instance sets, with BR6 and BR7 having an average solution difference of 0.89pp and 3.09pp, respectively. Therefore, our choice concerning *extrawalls* was 50.

4.2 Test Case 1

Considering only orientation and stability constraints, Araya et al. (2017) produce the current best known results in the literature. They improved the beam search method proposed by Araya and Riff (2014) for the specific packing problem which only considers these two constraints. Their improved method contains a new evaluation function with new evaluation criteria for ranking which boxes should be placed next. It is important to mention that although Araya et al. (2017) represent the current state of the art for this problem, this assessment only holds when comparing the average results per instance set. When analyzing each instance individually, other solution methods were able to find better solutions in a few cases, such as Fanslau and Bortfeldt (2010), Saraiva et al. (2019) and Zhu and Lim (2012). Liu et al. (2011), as already mentioned, is the only SLOPP approach in the literature considering multiple (> 4) constraints being the most closely related approach compared to ours. Despite considering multiple constraints, considering that not all those (connectivity and weight distribution) are similar to ours, the two handled in this section are the ones that can be properly analyzed.

We therefore compare our approach against the best results in the literature as well as the most closely related SLOPP approach. The performances of each method are presented in Table 5. The values correspond to the average container occupation when solving all 100 instances in each set.

When comparing the approaches which consider a larger set of constraints, our matheuristic was able to obtain better average results across all instance classes, with an average improvement of 2.2pp. When comparing our results against Araya et al. (2017), the best in the literature, our approach has an average of 2.93pp less container occupation. As expected, the worst results obtained by our method are for the instance class with the most heterogeneous instances (BR7). In terms of execution time, the pure heuristic beam search approach was limited to 150s. In the

Instance set	Liu et al. (2011)	Our approach	Araya et al. (2017)
BR1	88.14	92.64	94.74
BR2	89.52	92.88	95.38
BR3	90.53	93.05	95.65
BR4	90.75	92.54	95.48
BR5	90.79	92.13	95.33
BR6	90.74	91.85	95.23
BR7	90.07	90.89	94.66
Average	90.08	92.28	95.21

Table 5: Orientation and stability constraints.

experiments reported in Liu et al. (2011), the authors do not specify the execution time. It is also important to mention that all authors use different computer settings.

4.3 Test Case 2

Ramos et al. (2018) is the only paper from the literature which considers orientation, stability, load balance and weight limit constraints when solving the SLOPP. The authors proposed a genetic algorithm to solve the Container Loading Problem considering these four constraints.

To tackle load balance, the characteristics of the vehicle are required. Ramos et al. (2018) used the characteristics of the "Volvo Truck FE 62TR Air Ride" Trucks (2015), whose dimensions are compatible with the dimensions of the container used in the BR instances. Figure 3 and Table 6 detail the vehicle parameters. The performance of our method, compared against Ramos et al. (2018) is presented in Table 7.

Parameters	Values	Parameters	Values (ton-force)
H_d	0.832m	U	6.840
H_u	1.799m	P_{max}	19.160
TWB	4.397m	RT_{max}	19.000
T_{min}	25%	FT_{max}	7.100
S_{min}	25%		
LTR_{max}	10%		

Table 6: Container parameters.

As expected, the results show that adding extra constraints, in general, results in a deterioration of solution quality. The average gap with respect to the state of the art heuristic method created specifically to solve the problem with this set of constraints is 2.16pp and

Instance set	Our approach	Ramos et al. (2018)
BR1	90.08	91.31
BR2	90.83	92.40
BR3	91.85	93.32
BR4	91.56	93.58
BR5	91.33	93.60
BR6	90.67	93.48
BR7	89.64	93.35
Average	90.85	93.01

Table 7: Orientation, stability, weight limit and load balance constraints.

thus the proposed method maintains the same competitive behaviour exhibited in the previous experiments. In terms of execution time, again using different computer settings, the genetic algorithm, on average, was executed for at most 65s. As with the previous experiments, the worst results obtained by our approach were for the least homogeneous set of instances (BR7).

4.4 All Test Cases

The experiments in this section aim to show how consistent the solutions are when gradually introducing new constraints. Considering that results for the orientation, stability, weight limit and load balance have already been shown throughout the preceding sections, we begin by gradually including loading priorities, stacking and, finally, positioning. This will complete the full set of constraints.

Given that the constraints that will be included are directly related to box types, for experimentation purposes, some assumptions are required. When considering the loading priorities constraints, two test cases were created. The first (test case 3) assumes that the first half of all box types must have their full demand met. For example, if an instance has 4 box types, box types 1 and 2 must have their entire demand fulfilled. For test case 4, the second half of all box types must have their entire demand placed in the container, meaning that this time box types 3 and 4 must have their full demand fulfilled.

For test case 5, the positioning constraints also require some assumptions. Boxes of type 1 are to be delivered first and, therefore, they should be placed closer to the back of the truck than the other box types. The stacking constraints considered here use the concept of the most extreme fragility of a box. Therefore we assume that the last box type of each instance can only be placed above other boxes. For example, if an instance set contains 5 box types, then boxes of

type 5 cannot be placed under any other box. Given that box type 1 will be delivered first we maintain the same loading priorities as in test case 3, that is, the first half of all box types must have their entire demand fulfilled. Table 8 details the average occupation of each instance class for each test case.

Instance set	Test case 1		Test case 2		Test case 3		Test case 4		Test case 5
BR1	92.64		90.08		89.48		89.17		89.24
BR2	92.88		90.83		89.90		89.19		89.44
BR3	93.05		91.85		90.52		90.75		90.71
BR4	92.54		91.56		90.88		90.43		90.44
BR5	92.13		91.33		90.79		90.30		90.22
BR6	91.85		90.67		89.68		89.99		88.03
BR7	90.89		89.64		86.91		86.42		86.67
Average	92.28	(1.43)	90.85	(1.11)	89.74	(0.28)	89.46	(0.21)	89.25

Table 8: Different combinations of constraints.

Comparing the simplest version of the problem (test case 1 - orientation and stability constraints) against the most complex one (test case 5), the average difference in solution quality is 3.03pp. If we compare the difference of gradually adding new sets of constraints (consecutive test cases), then the inclusion of weight limit and load balance constraints (test case 2) are those with the highest impact, having a total average impact of 1.43pp compared to test case 1. The impact of the load balance constraints was already expected given that in the literature it is known that these are the most complex constraints.

Comparing test cases 3 and 4, which differ from each other only by which box types have loading priorities, it can be observed that the former performs better than the latter. Solutions that give loading priority to the first half of all box types result in a small occupation percentage improvement for the container (0.28pp). It is important to note that for BR3 and BR6, test case 4 actually performed slightly better (0.23pp and 0.31pp, respectively).

Test case 5, which combines all constraints considered in this paper, performs worst, achieving an average occupation of 89.25%. Nevertheless, the fact that the method maintained average results of around 89% even after including all these constraints for the other three test cases demonstrates how robust our approach is. Another test case was considered to be compared with test case 5, to evaluate a different assumption for the positioning constraints. Instead of prioritizing only box type 1 to be delivered first, we considered box type 2 to be delivered second. This means that all boxes of type 2 should be placed after boxes of type 1, but before all other

box types. Unfortunately, with the exception of BR1, the experiments across the majority of the remaining instance classes resulted in no solution that could fulfill all constraints.

In order to provide a thorough assessment of how our approach performs for all test cases, we also analyzed how much improvement there is from the initial solution compared to the final one, the average number of iterations performed in the method's final phase and how much time is spent in each phase.

Figure 8 illustrates the impact of the improvement phase. It can be observed that there is a direct correlation between the improvement percentage and the heterogeneity of instances. In other words, the most homogeneous instances (BR1 and BR2) have the least improvement from initial to final solution while the most heterogeneous instances (BR7) have the biggest improvement. By analyzing the different test cases, it is also noticeable that those which consider more constraints benefit more from the destruction and recreation of walls in the improvement phase, with test case 4 having the largest average improvement.

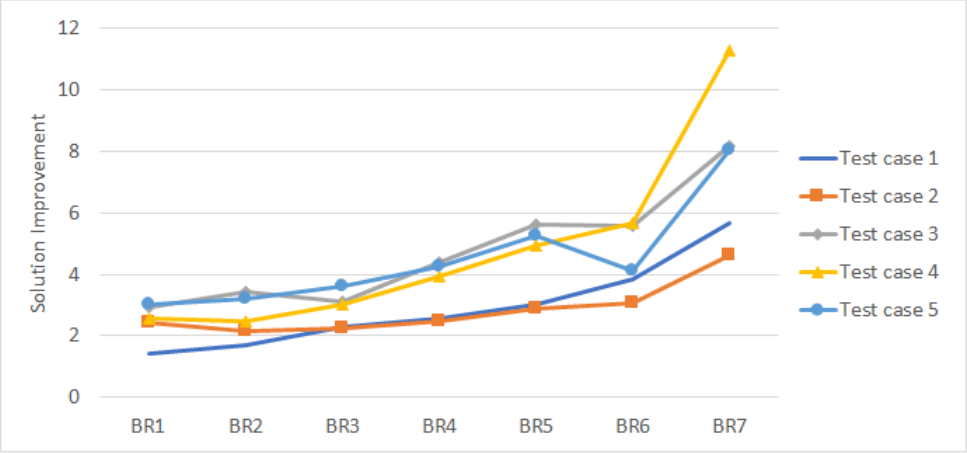


Figure 8: Average improvement percentage from first to best solution found.

As for the average number of iterations, Figure 9 shows that there is a clear division between test cases 2 and 3. For all test cases, there is an increase in the number of iterations as the instances become more heterogeneous, with BR1 having the smallest number of iterations and BR7 the greatest. The peak produced by the final instance set indicates that, although including more constraints increases the size of the model, the additional constraints work together in such a way that they reduce the convex hull of the solution space for the Packing problem, shortening its solution time each iteration (not using the limit of 60s), thereby increasing the number of iterations.

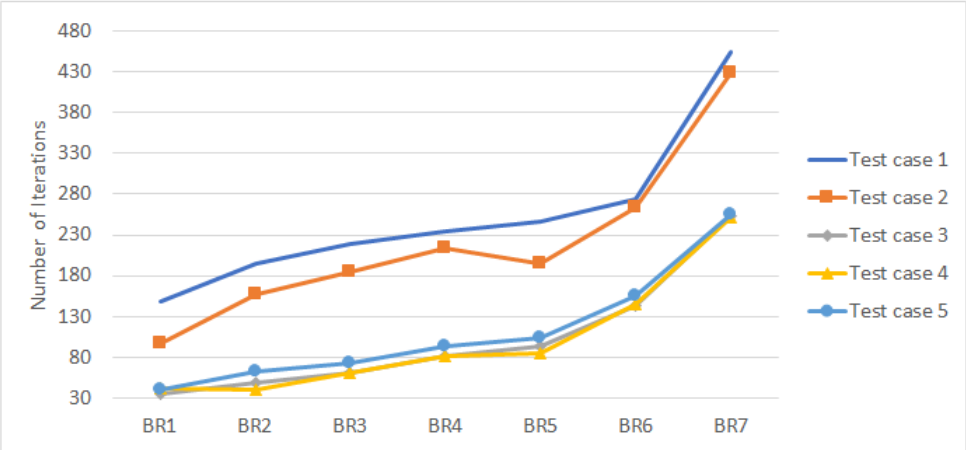


Figure 9: Average number of iterations.

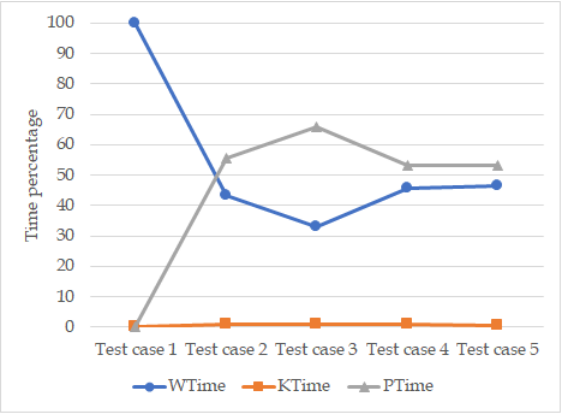
Finally, Figures 10 and 11 show how much time (%) is spent in each part of the solution method. *WTime*, *KTime* and *PTime* correspond to the percentage of the total execution time (900s) spent on the Wall Generation, Knapsack and Packing models, respectively.

The first thing made immediately obvious by these graphs is how the percentage of time spent solving the Knapsack model is very small ($< 1.5\%$). This indicates its efficiency as a "filter" for the walls to be considered in the Packing model. Since test case 1 only considers orientation and stability, and theses constraints are independent of the position of the boxes inside the container, the Packing model is not necessary, with almost 100% of its execution time spent in the wall building procedure. Test case 3 is the one that spends, on average, most of its execution time solving the Packing model.

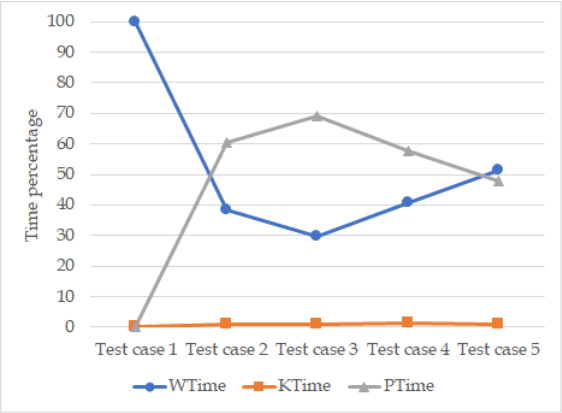
4.5 Results Discussion

The main reason which explains why the obtained results are worse than the state of the art is that for the proposed solution method to be able to handle different combinations of constraints, certain limitations must be imposed. For instance, the wall generation is implemented in such a way that reduces its diversity, as described in Section 3.1. The reason for this, besides speeding up the matheuristic, is that when preliminary experiments were performed allowing more diversity, this improved the results of some sets of constraints while simultaneously worsening others, creating a trade-off between the level of diversity and general solution quality.

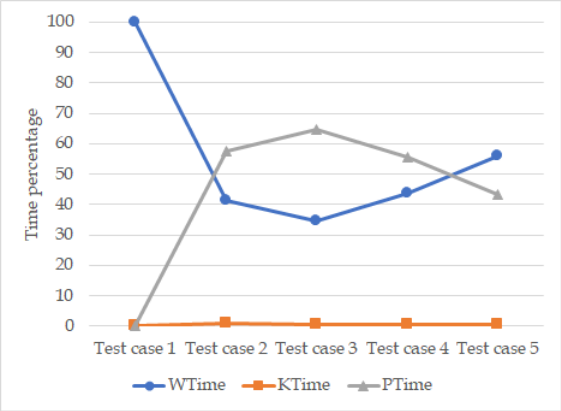
Therefore, the idea for the experiments was never to beat the state of the art in the literature, given that those results were achieved by specialized heuristics for each of the versions of



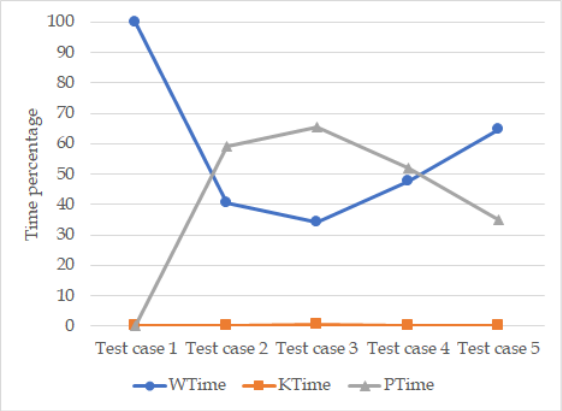
(a) Average time (%) spent per phase - BR1



(b) Average time (%) spent per phase - BR2



(c) Average time (%) spent per phase - BR3

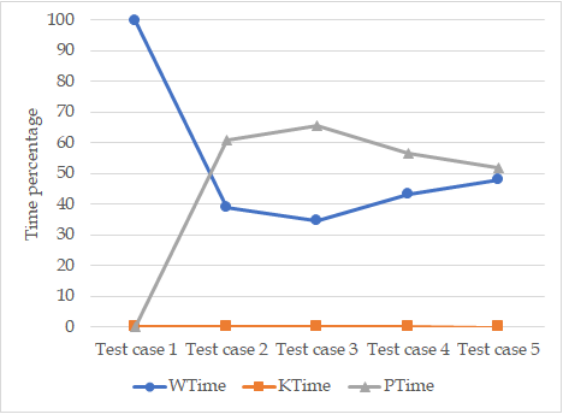


(d) Average time (%) spent per phase - BR4

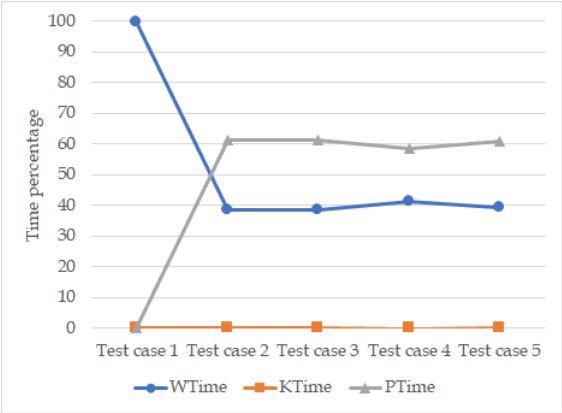
Figure 10: Time spent in each module of the solution approach (part I).

the problem tested. Our goal with the preceding experiments is to demonstrate how a more flexible solution approach can guarantee consistent results of good quality. Thus, after analyzing all executions, the results indeed demonstrate the robustness of the solution method: when considering different combinations of constraints, high quality results are consistently achieved withing the fixed time limit.

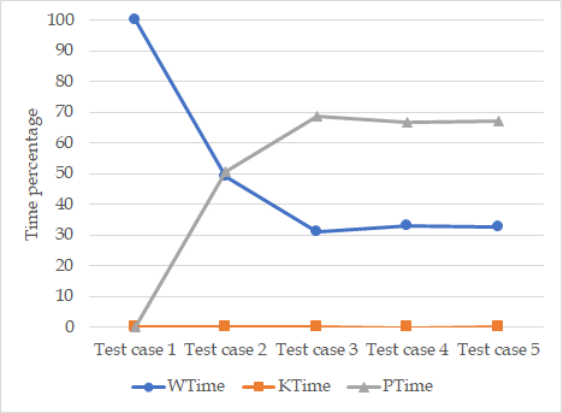
It is imperative to mention the importance of maintaining both the Knapsack and Packing models (when necessary), not only in terms of dividing the burden of the solution process, but to also facilitate the inclusion or removal of constraints, maintaining the hallmark of this approach. For the existing approaches in the literature, removing constraints would require changes in the solution procedure while the inclusion of additional constraints would require a redesign of the method or a post-processing procedure to accommodate the many unfeasible solutions which



(a) Average time (%) spent per phase - BR5



(b) Average time (%) spent per phase - BR6



(c) Average time (%) spent per phase - BR7

Figure 11: Time spent in each module of the solution approach (part II).

would occur. To show how this approach can be easily modified to handle other problems, the next Section adapts the proposed approach to consider multiple containers.

5 Multiple Container Approach

When adapting the single container solution approach to accommodate multiple containers, there are essentially two strategies for doing so: sequentially or simultaneously. The sequential method consists of reapplying the single container approach, one container at a time, and updating the demand for boxes after each new container is full. Otherwise, the simultaneous method consists of defining a set of containers that is enough to fit all the boxes to be placed and then solving the problem considering this full set of containers. Our proposed approach is straightforward to adapt from a single container to a simultaneous multiple containers solution method.

As an equivalent to the SLOPP, we consider here the Single Stock Size Cutting Stock Problem (SSSCSP). This problem consists of a set of weakly heterogeneous boxes that must be placed inside a set of identical containers. It is an input minimization problem which aims to minimize the number of containers used.

To be able to solve a SSSCSP, changes need only be made to the mathematical models used and not something else. The main difference is the addition of an index in the variables which corresponds to the walls used. For example, for the SLOPP in the Knapsack formulation, variable p_i is an integer variable that defines how many times wall i is used. For the SSSCSP, this variable changes to p_{ij} which corresponds to how many times wall i is used in container j.

Applying the same logic as for the Packing problem, variable s_i changes to s_{ij} which corresponds to wall i being used (or not) in container j. The same changes are applied to the variables that define the position of the wall inside the container (x_{ij}) and the position of the geometric center on the X-axis of the wall in the container (g_{ij} and NC_{x_j}). Furthermore, there is the addition of variable u_j in both formulations. This is a binary variable which is equal to 1 if container j is used and 0 otherwise.

In terms of constraints, aside from adapting all constraints to be enforced in each used container (by adding the extra index), it is necessary to include two sets of constraints in both models. The first set is a modification of the demand constraints: instead of limiting the number of boxes placed to be **at most** their respective demand, it is now mandatory that all the demand associated with each box type is fully met. Therefore, Constraints (5) and (13) are removed while (4) and (12) are replaced by (29) and (30), respectively:

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti}\cdot p_{ij} = \mathcal{D}_t, \qquad \forall\, t\in\mathcal{T}, \tag{29}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti}\cdot s_{ij} = \mathcal{D}_t, \qquad \forall\, t\in\mathcal{T}. \tag{30}$$

The second modification concerns the inclusion of a set of constraints that guarantees that if a box is placed in a container ($u_j = 1$) then this container is considered as used. These constraints are enforced by way of (31) and (32) in the Knapsack and the Packing formulations, respectively:

$$\sum_{i\in\mathcal{I}} p_{ij} \leq M\cdot u_j, \qquad \forall\, j\in\mathcal{J}, \tag{31}$$

$$\sum_{i \in \mathcal{I}} s_{ij} \leq M \cdot u_j, \qquad \forall\, j \in \mathcal{J}. \tag{32}$$

where M is a sufficient big number.

These constraints are used to control which containers are used in order to minimize each formulation's objective function:

$$\min \sum_{j \in \mathcal{J}} c_j \cdot u_j \tag{33}$$

where c_j is the cost of using container j.

The complete formulations to solve the SSSCSP are provided in Appendix A. To evaluate the effectiveness of the adapted solution approach, experiments were executed with well-known instances from the literature. Ivancic et al. (1989) proposed 47 instances for the SSSCSP which vary the number of necessary containers from 2 to 55 (current best known results). To evaluate our method, we compare its results against state of the art in the literature, Zhu et al. (2012), which consists of solving these instances considering only orientation and stability constraints.

The only difference between the experimental settings from the previous section is the time limit for the mathematical models. Since the formulations are bigger and more complex than those considering a single container, instead of limiting their execution time to 60s, we instead limit them to 180s. Table 9 details the instance solved, the obtained results by Zhu et al. (2012) and the results obtained by our solution method considering the same constraints as in the literature. The displayed values correspond to the number of containers used in the final solution.

The results demonstrate the same behavior as those obtained with the single container instances. When compared with the state of the art, the difference concerning the total number of containers used is only 3 (0.43%). The difference in a few solutions (three instances) happens for the same reason as in the previous experiments for the single container problem: the limitation concerning the diversity of walls.

Instance	Zhu et al. (2012)	Our approach
1	25	25
2	10	10
3	19	19
4	26	26
5	51	51
6	10	10
7	16	16
8	4	4
9	19	19
10	55	55
11	17	17
12	53	53
13	25	25
14	27	27
15	11	11
16	26	26
17	7	7
18	2	2
19	3	3
20	5	5
21	20	20
22	8	8
23	20	20
24	5	5
25	5	5
26	3	3
27	**4**	5
28	10	10
29	17	17
30	**22**	23
31	12	12
32	4	4
33	4	4
34	8	8
35	2	2
36	14	14
37	23	23
38	45	45
39	15	15
40	**8**	9
41	15	15
42	4	4
43	3	3
44	3	3
45	3	3
46	2	2
47	3	3
Total	693	696
Avg time (s)	117	900

Table 9: Number of containers used in the solution for each SSSCSP instance.

6 Conclusions

This paper proposed a matheuristic framework to consider multiple practical constraints for the Single Container Loading Problem, more specifically for the Single Large Object Placement Problem (SLOPP). In total, up to seven sets of practical constraints are considered, a degree of generality which was previously missing in the literature. Newly updated instances from the

literature considering different combinations of sets of constraints were used to evaluate the quality and robustness of the proposed method.

The experiments performed confirmed the quality of the method, achieving high-quality solutions across all test cases. When compared against the state-of-the-art in the literature, the results obtained had an average difference of at most 2.9pp in container occupation. This shows how efficient this general method can be, especially considering the fact that the best results in the literature are obtained by heuristics developed and calibrated for that specific problem with a specific set of constraints. Further experiments demonstrated the robustness of the method by analyzing the results obtained with different combinations of constraints. These results can also be used to evaluate the degree to which each of these constraints impacts the problem which, in this case, load balance had the greatest impact in solution quality.

The primary conclusion obtained is that it is possible to accommodate real-world problems while taking into account many different practical constraints in an efficient and simple way, bringing the problems solved in academia closer to those found in practice. We were able to structure a solution model with modules that can be easily modified, thereby facilitating the inclusion of additional constraints in a way which can benefit from advances in mathematical solvers. We also demonstrate how to extend this problem for the variant that considers multiple containers. Although adapting to consider different multiple containers is also straightforward we limit ourselves to explain how to use this method to solve the case with identical multiple containers (SSSCSP).

Future research includes the adaptation to other variants of container loading problems. The general solution method remains the same, with the main modifications involving the structure used to organize the boxes together, given that for problems characterized by strongly heterogeneous boxes the wall structure is not recommended. Another idea is to change the way in which walls are generated, focusing less on generating walls with boxes of the same type and allowing greater diversity. Finally, when considering problems with multiple containers, new research directions include to implement different types of constraints for those problems. For example, allocation constraints that require that items of a particular subset must go into the same container and multi-drop constraints which involves transportation costs and number of stops of each container.

Acknowledgements

This research was supported by the KU Leuven internal funds (project C24/17/048), the Brazilian National Council for Scientific and Technological Development (CNPq) (grant 308761/2018-9), State of São Paulo Research Foundation (FAPESP) (308761/2018-9) and CEPID (2013/07375-0).

We would like to thank Elsa Silva for providing the updated BR instances. Editorial consultation was provided by Luke Connolly (KU Leuven).

References

Araya, I., Guerrero, K., and Nuñez, E. (2017). VCS: A new heuristic function for selecting boxes in the single container loading problem. *Computers & Operations Research*, 82:27–35.

Araya, I. and Riff, M. C. (2014). A beam search approach to the container loading problem. *Computers & Operations Research*, 80:100–107.

Bischoff, E. E. (2006). Three-dimensional packing of items with limited load bearing strength. *European Journal of Operational Research*, 168:952–966.

Bischoff, E. E. and Ratcliff, M. S. W. (1995). Issues in the development of approaches to container loading. *European Journal of Operational Research*, 80:68–76.

Bortfeldt, A. and Gehring, H. (1999). Zur behandlung von restriktionen bei der stauraumoptimierung am beispiel eines genetischen algorithmus für das containerbeladeproblem. In Kopfer, H. and Bierwirth, C., editors, *Logistik Management: Intelligente I + K Technologien*, pages 83–100. Springer Berlin Heidelberg.

Bortfeldt, A. and Wäscher, G. (2013). Constraints in container loading - a state-of-the-art review. *European Journal of Operational Research*, 229:1–20.

Chen, C. S., Lee, S. M., and Shen, Q. S. (1995). An analytical model for the container loading problem. *European Journal of Operational Research*, 80:68–76.

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

Egeblad, J., Garavelli, C., Lisi, S., and Pisinger, D. (2010). Heuristics for container loading of furniture. *European Journal of Operational Research*, 200:881–892.

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

Fanslau, T. and Bortfeldt, A. (2010). A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, 22:222–235.

Hifi, M. (2002). Approximate algorithms for the container loading problem. *International Transactions in Operational Research*, 9:747–774.

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

Junqueira, L., Morabito, R., and Yamashita, D. S. (2012). Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, 39:74–85.

Layeb, S. B., Jabloun, O., and Jaoua, A. (2017). New heuristic for the single container loading problem. *International Journal of Economics & Strategic Management of Business Process*, 8:1–7.

Liu, J., Yue, Y., Dong, Z., Maple, C., and Keech, M. (2011). A novel hybrid tabu search to container loading. *Computers & Operations Research*, 38:797–807.

Lorie, J. H. and Savage, L. J. (1955). Three problems in capital rationing. *The Journal of Business*, 28:229–239.

Markowitz, H. M. and Manne, A. S. (1957). On the solution of discrete programming problems. *Econometrica*, 25:84–110.

Moon, I. and Nguyen, T. V. L. (2014). Container packing problem with balance constraints. *OR Spectrum*, 36:837–878.

Olsson, J., Larsson, T., and Quttineh, N.-H. (2020). Automating the planning of container loading for atlas copco: Coping with real-life stacking and stability constraints. *European Journal of Operational Research*, 280:1018–1034.

Ramos, A. G., Silva, E., and Oliveira, J. F. (2018). A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266:1140–1152.

Ren, J., Tian, Y., and Sawaragi, T. (2011). A tree search method for the container loading problem with shipment priority. *European Journal of Operational Research*, 214:526–535.

Saraiva, R. D., Nepomuceno, N., and Pinheiro, R. (2019). A two-phase approach for single container loading with weakly heterogeneous boxes. *Algorithms*, 12:67–82.

Silva, E., Ramos, A. G., and Oliveira, J. F. (2018). Load balance recovery for multi-drop distribution problems: A mixed integer linear programming approach. *Transportation Research Part B*, 116:62–75.

Silva, E. F., Toffolo, T. A. M., and Wauters, T. (2019). Exact methods for three-dimensional cutting and packing: A comparative study. *Computers and Operations Research*, 109:12–27.

Trucks, V. (2015). Data sheet specification model range fe 62tr. `http://segotn12827.rds.volvo.com/stpifiles/volvo/modelrange/fe62tra_prt_prt.pdf`. Accessed: July 26, 2019.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved topology of cutting and packing problems. *European Journal of Operational Research*, 183:1109–1130.

Zhu, W., Huang, W., and Lim, A. (2012). A prototype column generation strategy for the multiple container loading problem. *European Jouranl of Operational Research*, 223:27–39.

Zhu, W. and Lim, A. (2012). A new iterative-doubling greedy-lookahead algorithm for the single container loading problem. *European Journal of Operational Research*, 222:408–417.

A SSSCSP Formulations

Knapsack formulation for the SSSCSP problem:

$$\min \sum_{j\in\mathcal{J}} c_j \cdot u_j \tag{34}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} l_i \cdot p_{ij} \le L, \tag{35}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti} \cdot p_{ij} = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}, \tag{36}$$

$$\sum_{i\in\mathcal{I}} p_{ij} \le M \cdot u_j, \qquad \forall\, j \in \mathcal{J}, \tag{37}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} m_i \cdot p_{ij} \le P_{max}, \tag{38}$$

$$u_j \in \{0,1\}, \qquad \forall\, j \in \mathcal{J}, \tag{39}$$

$$p_{ij} \in \mathbb{Z}_+, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}. \tag{40}$$

Packing formulation for the SSSCSP problem:

$$\min \sum_{j\in\mathcal{J}} c_j \cdot u_j \tag{41}$$

$$\text{s.t. } x_i + l_i \le x_k + (1 - a_{ik}) \cdot L, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{42}$$

$$x_k + l_k \le x_i + (1 - b_{ik}) \cdot L, \qquad \forall\, i,k \in \mathcal{I}, i < k, \tag{43}$$

$$a_{ik} + b_{ik} \ge s_{ij} + s_{kj} - 1, \qquad \forall\, i,k \in \mathcal{I}, i < k, \; \forall\, j \in \mathcal{J}, \tag{44}$$

$$x_i + l_i \le L + (1 - s_{ji}) \cdot L, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{45}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} d_{ti} \cdot s_{ij} = \mathcal{D}_t, \qquad \forall\, t \in \mathcal{T}, \tag{46}$$

$$\sum_{i\in\mathcal{I}} s_{ij} \le M \cdot u_j, \qquad \forall\, j \in \mathcal{J}, \tag{47}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot s_{ij} = P_{cargo_j}, \qquad \forall\, j \in \mathcal{J}, \tag{48}$$

$$P_{cargo_j} \leq P_{max}, \qquad \forall\, j \in \mathcal{J}, \tag{49}$$

$a_{ik} = 0$, if wall i contains t_1 and wall k contains t_2 $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

and a positioning constraint between box types t_1 and t_2 is active, $\qquad \forall\, i, k \in \mathcal{I}, i < k,$ (50)

$b_{ik} = 0$, if wall i contains t_2 and wall j contains t_1 $\qquad \forall\, t_1, t_2 \in \mathcal{T}, t_1 \neq t_2,$

and a positioning constraint between box types t_1 and t_2 is active, $\qquad \forall\, i, k \in \mathcal{I}, i < k,$ (51)

$$g_{ij} \leq L \cdot s_{ij}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{52}$$

$$g_{ij} \leq x_{ij} + \frac{l_i}{2}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}, \tag{53}$$

$$g_{ij} \geq x_{ij} + \frac{l_i}{2} + L \cdot s_{ij} - L, \qquad \forall\, j \in \mathcal{J}, \tag{54}$$

$$\sum_{i \in \mathcal{I}} m_i \cdot g_{ij} = NC_{x_j}, \qquad \forall\, j \in \mathcal{J}, \tag{55}$$

$$NC_{x_j} \leq P_{cargo_j} \cdot (TWB - H_d) - TWB \cdot FT_{max} + U \cdot (TWB - H_u), \qquad \forall\, j \in \mathcal{J}, \tag{56}$$

$$NC_{x_j} \leq TWB \cdot RT_{max} - U \cdot H_u - P_{cargo} * H_d, \qquad \forall\, j \in \mathcal{J}, \tag{57}$$

$$NC_{x_j} \geq P_{cargo} \cdot (H_d - S_{min} \cdot TWB - TWB) + U \cdot (TWB - H_u - S_{min} \cdot TWB), \qquad \forall\, j \in \mathcal{J}, \tag{58}$$

$$NC_{x_j} \geq U \cdot (T_{min} \cdot TWB - H_u) + P_{cargo} \cdot (T_{min} \cdot TWB - H_d), \qquad \forall\, j \in \mathcal{J}, \tag{59}$$

$$u_j, s_{ij} \in \{0, 1\}, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J} \tag{60}$$

$$a_{ik}, b_{ik} \in \{0, 1\}, \qquad \forall\, i, k \in \mathcal{I}, \tag{61}$$

$$x_{ij}, g_{ij}, P_{cargo_j}, NC_{x_j} \geq 0, \qquad \forall\, i \in \mathcal{I}, \forall\, j \in \mathcal{J}. \tag{62}$$