Transportation Research Part B 91 (2016) 565–581

Contents lists available at ScienceDirect

Transportation Research Part B

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

A container loading algorithm with static mechanical equilibrium stability constraints

A. Galrão Ramos[a,b,*], José F. Oliveira[a,c], José F. Gonçalves[a,d], Manuel P. Lopes[b]

[a] INESC-TEC, Portugal
[b] CIDEM, School of Engineering, Polytechnic of Porto, Portugal
[c] Faculty of Engineering, University of Porto, Portugal
[d] Faculty of Economics, University of Porto, Portugal

ARTICLE INFO

Keywords:
Container loading problem
Static stability
Maximal-spaces
Genetic algorithms

ABSTRACT

The Container Loading Problem (CLP) literature has traditionally guaranteed cargo static stability by imposing the full support constraint for the base of the box. Used as a proxy for real-world static stability, this constraint excessively restricts the container space utilization and has conditioned the algorithms developed for this problem. In this paper we propose a container loading algorithm with static stability constraints based on the static mechanical equilibrium conditions applied to rigid bodies, which derive from Newton's laws of motion. The algorithm is a multi-population biased random-key genetic algorithm, with a new placement procedure that uses the *maximal-spaces* representation to manage empty spaces, and a layer building strategy to fill the *maximal-spaces*. The new static stability criterion is embedded in the placement procedure and in the evaluation function of the algorithm. The new algorithm is extensively tested on well-known literature benchmark instances using three variants: no stability constraint, the classical full base support constraint and with the new static stability constraint—a comparison is then made with the state-of-the-art algorithms for the CLP. The computational experiments show that by using the new stability criterion it is always possible to achieve a higher percentage of space utilization than with the classical full base support constraint, for all classes of problems, while still guaranteeing static stability. Moreover, for highly heterogeneous cargo the new algorithm with full base support constraint outperforms the other literature approaches, improving the best solutions known for these classes of problems.

1. Introduction

The Container Loading Problem (CLP) is a real-world driven, combinatorial optimization problem that addresses the optimization of the spatial arrangement of cargo inside containers or transportation vehicles, maximizing the usage of space.

As an assignment problem, it can have two basic objectives: the maximization of the value of the cargo loaded, when the number of containers is not sufficient to accommodate all the cargo, or the minimization of the value of containers, when there are sufficient containers to accommodate all the cargo.

The problem belongs to the wider combinatorial optimization class of Cutting and Packing problems. According to the typology defined by Wäscher et al. (2007) for Cutting and Packing problems, these can be classified according to

* Corresponding author.
E-mail address: agr@isep.ipp.pt (A. Galrão Ramos).

http://dx.doi.org/10.1016/j.trb.2016.06.003
0191-2615/

dimensionality, assortment of large items, assortment of small items, kind of assignment and shape of small items. In this paper we will consider two types of problem with an output maximization objective. These problems can be classified either as three-dimensional, rectangular single large object placement problems (3D-SLOPP) or as three-dimensional, rectangular single knapsack problems (3D-SKP), depending on the cargo heterogeneity.

The CLP is highly relevant to the field of transport management. The effect of globalization led to a world where products and services are exchanged in increasing numbers and distances by and to an increasing number of origins and destinations, and where containerization is the standard method of transporting goods and cargo worldwide. This scenario places a number of challenges to achieve an efficient transport system, required for maintaining prosperity and economic development. New problems that arose such as the ones related with the urban freight transportation (Sánchez-Díaz et al., 2015) or the designs of intermodal networks (Meng and Wang, 2011) can directly benefit from a reduction of the congestion of cargo transport units that an efficient container space usage can provide.

The arrangements for loading cargo into containers should comply with various requirements: cargo should not become damaged during transportation, transportation space should be used efficiently and workers' safety should not be breached during loading and unloading of cargo.

However, if the approach to the problem does not consider real-world constraints, such as cargo stability, container weight-limit or cargo orientation constraints, the solution will be of limited applicability to real-world scenarios. Cargo stability is considered in the literature as one of the most important CLP constraints. Its impact is not confined to the cargo as it can also influence the safety of both workers involved in loading operations and other persons or vehicles during transportation.

In the CLP literature, cargo stability is sometimes addressed separating static and dynamic stability. Cargo static stability has been guaranteed by imposing the full support constraint on the base of the boxes. Although guaranteeing static stability, it excessively restricts the container space usage and does not necessarily meet real-world needs when e.g., overhanging cargo is allowed. The rather oversimplified way static stability has been treated by the majority of the authors it is also present in existing approaches to dynamic stability, where stability is measured by the mean number of boxes supporting the items excluding those placed directly on the floor and the percentage of boxes with insufficient lateral support (Ramos et al., 2015).

The CLP addressed in this work can be stated as follows: A given set of small items of parallelepiped shape of type $k(k = 1, \ldots, K)$ (known as boxes), $B = b_1, b_2, \ldots, b_K$, where each box type, in quantity n_k, is characterized by its depth, width and height (d_k, w_k, h_k) are to be loaded into a large object of parallelepiped shape (known as a container), C, characterized by its depth, width and height, (D, W, H), with the objective of achieving a maximum utilization of the volume of the container, while meeting the following geometric loading constraints:

- Each face of a box must be parallel to one of the faces of the container;
- There must be no overlap between the boxes;
- All boxes must lie entirely within the container;
- Each box must be placed according to one of its possible orientations—each box type can have up to six possible orientations.

The mechanical properties of the container and the boxes also necessitate the following additional practical constraints:

- boxes can only be loaded through the container entrance;
- static stability—each box must be able to maintain its loading position undisturbed during cargo loading;
- all boxes are rigid;
- the centre of gravity of each box is assumed to be its geometric centre.

The dimensions (D, W, H) of container C lie parallel to the x, y and z axes, respectively, of the first octant of a Cartesian coordinates system, with the back-bottom-left corner lying at the origin of the coordinates system. The placement of a box b_i in the container is given by its minimum and maximum coordinates, (x_{1i}, y_{1i}, z_{1i}) and (x_{2i}, y_{2i}, z_{2i}), respectively.

The aim of this work is to present an algorithm for the CLP that addresses cargo stability under a realistic framework. The proposed algorithm combines a multi-population biased random-key genetic algorithm with a constructive heuristic that enforces a static stability constraint based on the static mechanical equilibrium conditions applied to rigid bodies, which derive from Newton's laws of motion. The constructive heuristic uses a *maximal-spaces* representation to manage empty spaces, and a layer approach for filling the *maximal-spaces*.

The remainder of the paper is organized as follows. Section 2 presents an overview of the literature covering the CLP and static stability within the CLP. In Section 3 the Container Loading Algorithm with Static Stability is presented. Section 4 reports the results from the computational experiments. Finally, Section 5 draws some conclusions from the findings.

2. Literature review

Many approaches have been proposed for solving the 3D-SLOPP and the 3D-SKP. The number of exact methods proposed is very limited and these can only solve problems of limited size. Exact methods were developed by Padberg (2000), Fekete et al. (2007), Junqueira et al. (2012b) and Junqueira et al. (2012a). Alternatively, other methods have been proposed to find near-optimal packing solutions. Fanslau and Bortfeldt (2010) classified these methods as conventional heuristics,

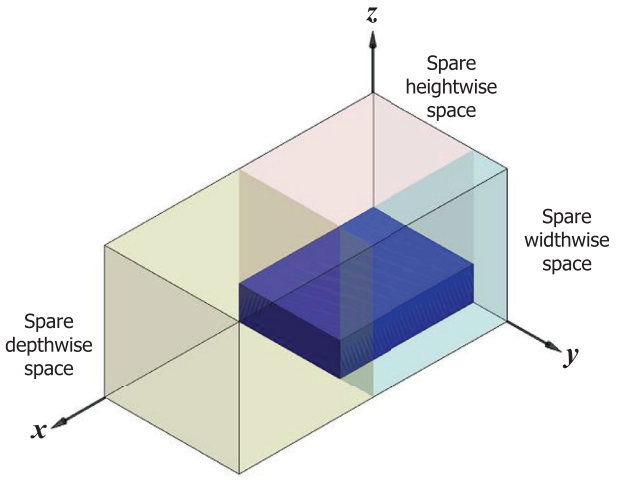

Fig. 1. George and Robinson (1980) empty space subdivision.

metaheuristics and tree-search methods. Conventional heuristics include construction heuristics and improvement heuristics. Construction heuristics derive a single feasible solution that is directly employed or used as a starting point for local search heuristics. They do not attempt to improve the obtained solution. Improvement heuristics try to iteratively improve the solution of already known solutions. Examples include methods developed by George and Robinson (1980), Bischoff et al. (1995), Lim et al. (2003) and Liu et al. (2011). Metaheuristics can be seen as general-purpose methods that aim to effectively and efficiently explore a search space using intensification (exploitation) and diversification (exploration) strategies. According to the philosophy followed, metaheuristics can be seen as extended variants of improvement heuristics that aim to escape from a local optimal solution and continue with the exploration of the search space with the expectation of finding a better solution, or as a population-based approach where the search space is explored in each iteration by a population. Examples of such approaches can be found in the Genetic Algorithms of Bortfeldt and Gehring (2001) and Gonçalves and Resende (2012), in the use of Tabu Search by Bortfeldt et al. (2003) and Liu et al. (2011) and the greedy randomized adaptative search procedures (GRASP) of Moura and Oliveira (2005) and Parreño et al. (2008). Tree-search methods include tree-search and graph-search methods. These are methods that can be used when the set of all feasible solutions of the optimization problem can be represented by a tree or a graph. Examples include the works of Fanslau and Bortfeldt (2010), Zhu and Lim (2012) and Araya and Riff (2014).

A common feature of these methods is that they search over a representation or codification of the solution (usually box sequences) and therefore require a constructive box loading heuristic to generate a feasible solution. Usually, the heuristic iteratively selects a location inside the container and a box (or set of boxes) to place at that location, until no more locations or boxes are available. Both of these decisions are related to the way the empty space of the container is managed (and therefore the way in which all potential placement locations are evaluated) and the way box arrangements are generated.

Spatial representation. Different approaches can be found in the literature to managing empty spaces. Ngoi et al. (1994) use a single three-dimensional matrix representation of objects and empty spaces as a combination of variable orthorhombic cells. Each three-dimensional matrix is composed of a chain of two-dimensional matrices that represent the details of horizontal layers of constant thickness. Bischoff (2006) proposed an adaptation of the Ngoi et al. (1994) representation that does not involve the creation of the horizontal layer matrices. Instead, a single two dimensional matrix that represents a view from the top of the container is required. Another approach was proposed by George and Robinson (1980). Their methodology is based on the assumption that after the placement of a box in a packing space, the remaining unused space opens up three new spaces. The three spaces, illustrated in Fig. 1, are created in the following order: spare depthwise space, spare widthwise space and spare heightwise space. Each space is therefore represented by its depth, height and width and the coordinates of its rear-left-bottom vertex.

The approach proposed by George and Robinson (1980) only considered one variant to partitioning the empty space. Other authors, such as Bortfeldt et al. (2003) and Fanslau and Bortfeldt (2010), later extended the George and Robinson (1980) approach and considered other variants of the empty space subdivision, as illustrated in Fig. 2. In these approaches, empty spaces are represented as a set of disjoint spaces.

For the two and three-dimensional cutting stock problem, Lai and Chan (1997) proposed a representation of empty spaces as a set of non-disjoint empty spaces. These empty spaces have the largest parallelepiped shape that can be considered and are managed using the "Interval Generation" procedure. However, this procedure was not applied, by the authors, to the three-dimensional CLP. Later Parreño et al. (2008) used this representation in the CLP and designated the non-disjoint empty spaces as *maximal-spaces*. The *maximal-spaces* representation is illustrated in Fig. 3. A *maximal-space s* representation

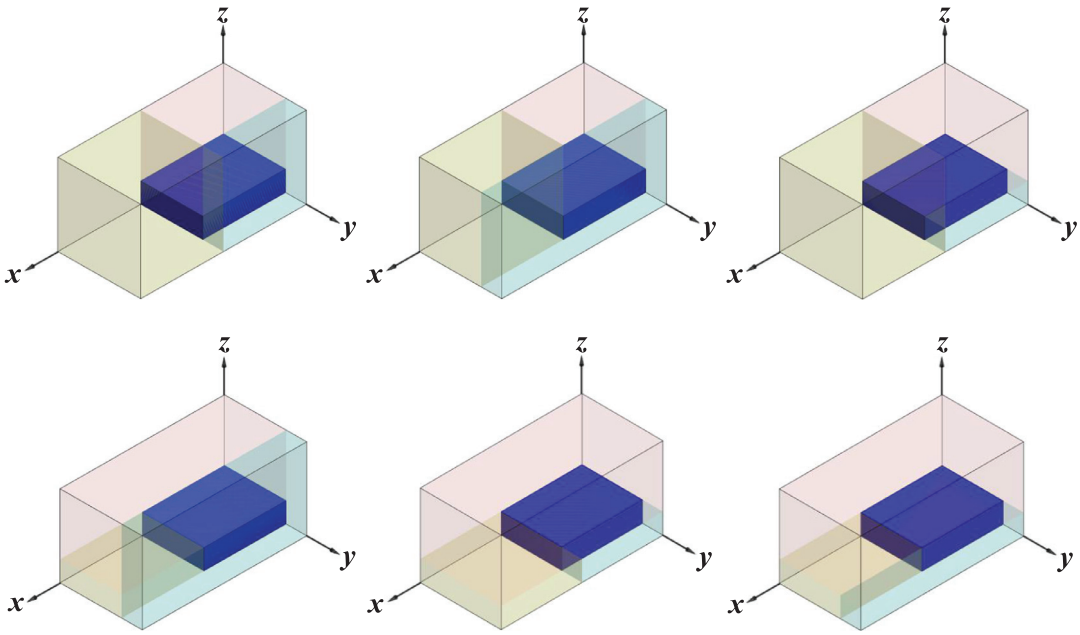

Fig. 2. Variants of the empty space subdivision.

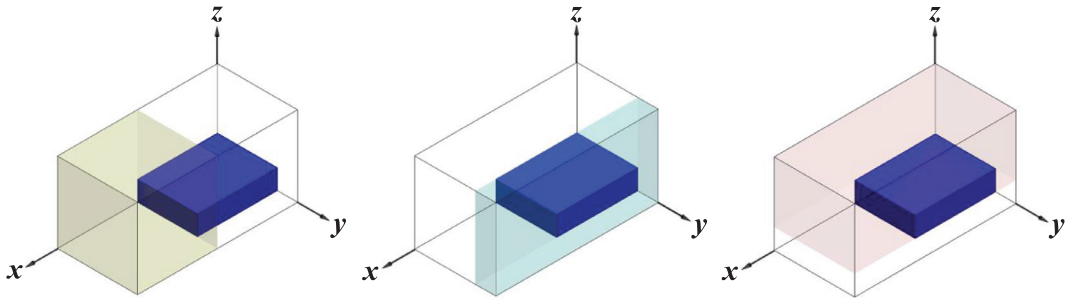

Fig. 3. *Maximal-spaces* representation.

is given by its minimum and maximum coordinates, (x_{1s}, y_{1s}, z_{1s}) and (x_{2s}, y_{2s}, z_{2s}), respectively, and an Insertion Vertex V, (that is, the vertex of the *maximal-space* where the boxes will be packed).

Box arrangement strategy. Either single box or multiple box strategies can be followed at each placement; each iteration of the former places only one box inside the container, each iteration of the latter places a set of boxes together. In a multiple box strategy the arrangements can be formed from sets of identical or non-identical boxes. Multiple boxes can also be arranged by dimensionality, (that is, one, two and three dimensions). In the one dimensional arrangement, boxes are grouped along a single axis, and therefore generate a column. In a two dimensional arrangement, boxes are grouped in a plane, (that is, along two axes, generating layers). In the three dimensional arrangement, boxes are grouped along the three axes, generating blocks.

The constructive heuristic results from the combination of the selected spatial representation and box arrangement strategy and a set of rules that select location and box arrangement at each iteration.

The best performing CLP algorithm for the 3D-SLOPP and 3D-SKP that uses a two dimensional arrangement strategy is the multi-population biased random-key genetic algorithm developed by Gonçalves and Resende (2012), while the best performing three dimensional arrangement strategy is the Beam Search Approach developed by Araya and Riff (2014).

2.1. Static stability

A recent literature review of container loading constraints by Bortfeldt and Wäscher (2013) highlights the particular relevance placed on stability constraints. Considered to be one of the most important CLP constraints, stability has been addressed by a large number of authors usually following a rather simplified approach that often considers the term “stability”

Table 1
Comparison of the best existing algorithms using Full Support and Unsupported variants.

	Full Support (FS)		Unsupported (U)		Diference (U-FS)	
Problem	BSG	BRKGA	BSG	BRKGA	BSG	BRKGA
BR 1–7	94.74	94.53	96.11	95.74	1.36	1.20
BR 8–15	91.22	90.23	94.78	93.49	3.56	3.26
BR 1–15	92.87	92.24	95.40	94.54	2.53	2.30

as if it was self-explanatory. Concepts, such as loading stability and transportation stability, are sometimes not addressed separately, but there are some approaches to stability found in the literature that make a clear distinction between static (vertical) and dynamic (horizontal) stability, (that is, the stability of cargo during loading operations and the stability of cargo during transportation). However, the majority of approaches only focus on static stability (Bortfeldt and Wäscher, 2013).

The approaches to static stability found in the CLP literature can be classified according to the type of stability constraint to be enforced: full base support, partial base support or static mechanical equilibrium. Of these, full base support and static mechanical equilibrium both guarantee static stability, while partial support does not.

- **Full base support** requires the entire base of a box be in contact with the base of the container or with the top surface of other boxes. As a result, no overhanging boxes are allowed. Examples can be found in Bischoff and Ratcliff (1995), Bortfeldt and Gehring (2001), Gonçalves and Resende (2012) and Zhu and Lim (2012).
- **Partial base support** requires that either the entire base of a box be in contact with the base of the container, or a pre-specified percentage of the area of a box base be in contact with the top surface of other boxes, thereby allowing overhanging. As an example, Carpenter and Dowsland (1985) requires the contact area to fall in the range of 95% to 75%, while Christensen and Rousœ (2009) require a minimum of 80%, Gendreau et al. (2006), Fuellerer et al. (2010), Tarantilis et al. (2009) and (Zhang et al., 2015), 75%, Gehring and Bortfeldt (1997), 70% and Mack et al. (2004), 55%.
- **Static mechanical equilibrium** requires that the entire base of a box be in contact with the base of the container or,
  - the sum of external forces acting on the box is zero and;
  - the sum of torque exerted by the external forces is zero.

  An example, applied to the three-dimensional bin packing problem, can be found in de Castro Silva et al. (2003). The center of gravity condition is a condition found in the literature which is derived from the static mechanical equilibrium conditions applied to rigid bodies. This condition requires the center of gravity of a box be located above the contact surface of the supporting boxes (Lin et al., 2006; Mack et al., 2004). However, by itself, enforcing this condition does not guarantee static stability (Ramos et al., 2016).

Recent approaches in the literature consider the concept of static stability as equivalent to enforcing full base support (Araya and Riff, 2014; Gonçalves and Resende, 2012; Zhu and Lim, 2012). Consequently these approaches developed both algorithms that enforce full base support and algorithms that have no such requirement (Unsupported). The performance benchmarking used for these algorithms is the percentage of volume loaded with or without full base support. As a result, the goal is not to obtain a loading arrangement that is statically stable but a loading arrangement where all boxes have full base support.

Table 1 summarizes the results of the previously identify best existing CLP approaches without the static stability constraint (Unsupported) and with enforcement of the full base support constraint (Full Support). In the table BSG-CLP refers to the Beam Search Approach of Araya and Riff (2014) and BRKGA refers to the multi-population biased random-key genetic algorithm of Gonçalves and Resende (2012). The values in columns 2 to 5 of the table correspond to the average percentage of volume utilization for the test instances of Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) organized in 15 classes, with a total of 100 instances per class. Classes BR1 to BR7 are weakly heterogeneous classes while BR8 to BR15 are strongly heterogeneous classes. The values in the columns below the label “Difference”, represent the difference between the results of the Unsupported and Full Support variants. Looking at the results leads to the conclusion that the full base support constraint is very costly for algorithm efficiency, particularly in the strongly heterogeneous instances. To give an idea of the impact this represents for transportation cost, 3% of the volume of a 40-foot container is around 1.5 m^3 of space.

The best performing algorithms use two different heuristic approaches: BSG-CLP uses a beam search heuristic approach while BRKGA uses the genetic algorithm. The BSG-CLP provides the best results for the Unsupported and Full Support CLP variants. However, these algorithms are not so flexible when faced with additional constraints, such as load bearing, weight limit or weight distribution, that can only be evaluated after the loading arrangement has been completed.

3. The container loading algorithm with static stability

The proposed container loading algorithm hybridizes a multi-population biased random-key genetic algorithm (BRKGA) with a constructive heuristic embedded with a static stability constraint, based on the static mechanical equilibrium conditions applied to rigid bodies, which derive from Newton’s laws of motion. In a BRKGA, as in all metaheuristics, there is

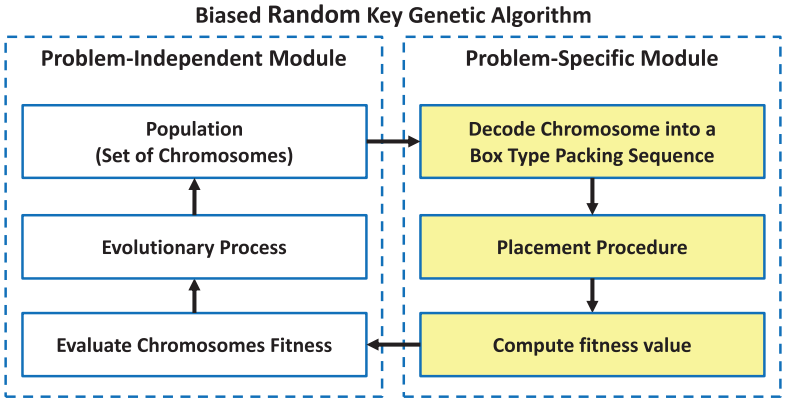

Fig. 4. Architecture of the base algorithm.

a problem-independent part, responsible for the evolution of coded solutions (chromosomes), and a problem-specific part, responsible for decoding the chromosome, generating a solution and evaluating its fitness (Gonçalves and Resende, 2011).

The use of a BRKGA for the CLP was first proposed by Gonçalves and Resende (2012) to solve the 3D-SLOPP and 3D-SKP and by Gonçalves and Resende (2013) for the three-dimensional, Single Bin-Size Bin Packing Problem (3D-SBSBPP).

Gonçalves and Resende (2011) stated that it is the chromosome representation and the problem-specific part of the algorithm that requires research effort, and distinguishes the different BRKGA's approaches to the same or different problems. For the here proposed algorithm, a new chromosome representation and a new problem-specific part (that is, a constructive heuristic) are specified. Fig. 4 illustrates the architecture of the algorithm.

The constructive heuristic starts with the decoding procedure that generates the sequence by which the boxes are loaded into the container. The generation of an actual solution is obtained by a box placement procedure that uses a *maximal-space* representation of the empty spaces and a layer building strategy (2D box arrangement strategy). Finally, the value of the solution is determined by the percentage of total packed volume.

The following section firstly provides a description of two indirect solution representations and decoding procedures. Then, presents a placement heuristic and the fitness function for the Unsupported and Full Support CLP variants. Finally the description of the placement heuristic and fitness function for the static mechanical equilibrium variant, used to guarantee cargo static stability, is presented.

3.1. Chromosome encoding and decoding

A chromosome in a genetic algorithm represents a solution to the problem. It can be a direct or indirect representation, depending on whether the chromosome is a representation of the solution to the original problem or whether additional procedures are needed to obtain a solution. The use of an indirect representation for the CLP, within a genetic algorithm framework, is preferable to the direct use of chromosomes as packing sequences for the CLP, since in a direct representation, the genes must represent the box placement coordinates, which would make the overlay constraints much harder to enforce. By using a biased random-key genetic algorithm, where the chromosome is encoded as a vector of random keys (real numbers between 0 and 1), an indirect representation of the solution is used which guarantees that the offspring formed by crossover of the chromosomes are feasible solutions (Gonçalves and Resende, 2011). The use of an indirect representation of the solution requires the chromosomes to be decoded for the CLP.

Two indirect representations of the solution are adopted: one strongly heterogeneous and one weakly heterogeneous. In the strongly heterogeneous case, each gene is used to represent the box type for each of the boxes to be loaded. In the weakly heterogeneous case, a first set of genes represent the box type of each of the boxes to be loaded and a second set of genes represents one of the three planes along which layers of boxes can be built up ($x-y$, $x-z$ and $y-z$) to fill the *maximal-spaces*. The second set of genes aims increasing the diversity of generated solutions by the constructive heuristic.

Let chromosome $G = \{gene_1, gene_2, \ldots, gene_g\}$ be an indirect representation of a CLP solution and let M represent the number of boxes to be loaded ($M = \sum_{k=1}^{K} n_k$). The number of genes, g, of the chromosome depends on the indirect solution representation and the total number of boxes to be loaded M.

- In the **strongly heterogeneous** representation, g is equal to M. Each $gene_j$ represents the type of box of the j^{th} box to be loaded. By sorting the genes in ascending order a new box type sequence is generated. The new vector sequence is designated the Box Type Packing Sequence (BTPS).
- In the **weakly heterogeneous** representation, g is equal to $2M$. For the first M genes, each $gene_j$ represents the type of box of the j^{th} box to be loaded while the last M genes ($gene_{M+j}$) represent the layer filling plane of the j^{th} box to load.

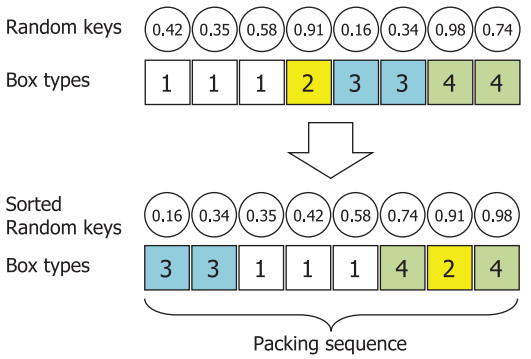

Fig. 5. Strongly heterogeneous chromosome decoding procedure example.

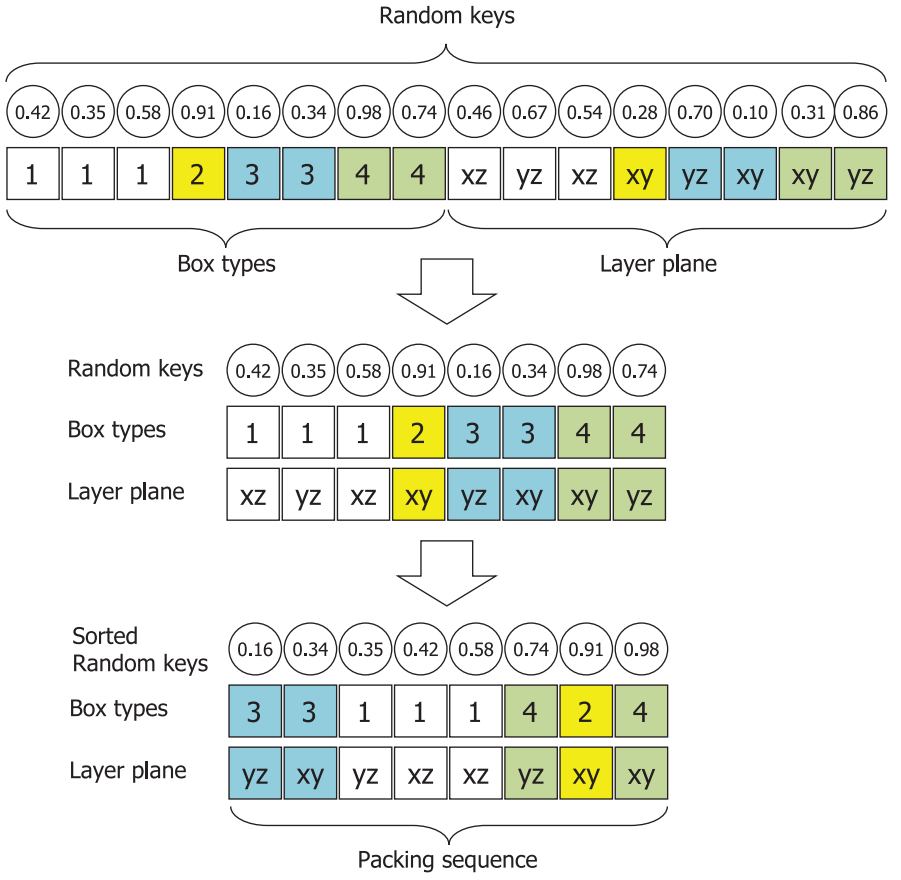

Fig. 6. Weakly heterogeneous chromosome decoding procedure example.

The layer filling plane of box j is determined using (1).

$$\begin{cases} 0 \le gene_{M+j} < 1/3 & x-y \text{ plane} \\ 1/3 \le gene_{M+j} < 2/3 & x-z \text{ plane} \\ 2/3 \le gene_{M+j} < 1 & y-z \text{ plane} \end{cases} \tag{1}$$

By sorting the first M genes in ascending order a BTPS is generated as well as a vector of Layer Filling Planes (LFP).

Figs. 5 and 6 respectively illustrate the decoding procedure for the **strongly heterogeneous** and **weakly heterogeneous** chromosome representation. In the examples there are 4 types of boxes, b_1, b_2, b_3 and b_4, with $n_1 = 3$, $n_2 = 1$, $n_3 = 2$ and $n_4 = 2$.

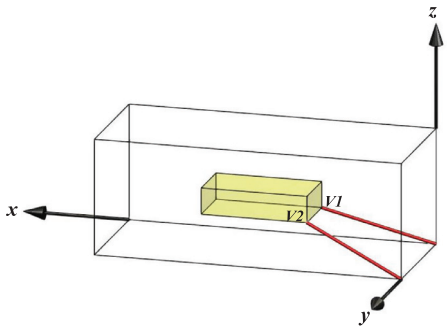

Fig. 7. Distances between container corners and the *maximal-space* reference vertex.

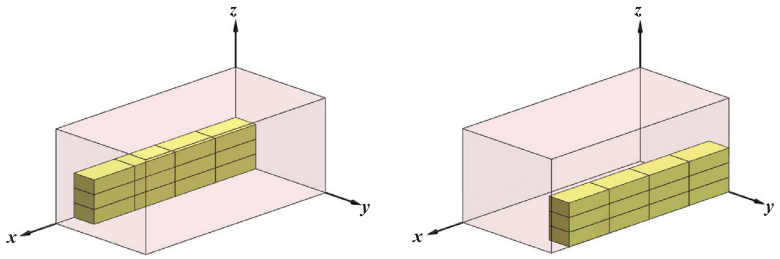

Fig. 8. Two potential placements of a layer in a *maximal-space*.

3.2. The placement heuristic

The placement heuristic is an iterative process with four main steps: selection of the box type, selection of the *maximal-space*, filling the *maximal-space*, and updating the system information. At each iteration the process tries to pack a layer of identical boxes in the container using a set of *maximal-spaces*. Three elements are combined in the process, a box type packaging sequence vector, a list of unpacked boxes and a list of *maximal-spaces*.

Step 0: Initialization

$S = \{C\}$, set of *maximal-spaces*
$B = \{b_1, b_2, \ldots, b_K\}$, set of box types—unpacked
$q_k = n_k$, number of boxes of type k—unpacked
$P = \varnothing$, set of boxes—packed

Step 1: Selecting the box type

The aim of the first step is to select the type of box to be loaded. The order in which the type of boxes are loaded into the container is provided by the BTPS vector obtained from the decoding procedure. The i^{th} type of box to be loaded is given by $BTPS(i)$. Successive iterations check to see if $q_k = 0$ for the selected box type k, whereby, if true, the next box type in the sequence is selected.

Step 2: Selecting the maximal-space

To select an empty *maximal-space* we use the back-bottom rule. The back-bottom rule first selects the *maximal-space* that is closest to the back of the container (x-axis), then selecting the space closest to the bottom of the container (z-axis), finally selecting whichever space is closer to one of the two back-bottom corners of the container (y-axis). The smallest distance between *maximal-space* s, with coordinates (x_{1s}, y_{1s}, z_{1s}) and (x_{2s}, y_{2s}, z_{2s}), to the two corners of the container, with coordinates (0, 0, 0) and (0, W, 0) respectively, is given by $y_{Vs} = min\{y_{s1}, (W - y_{s2})\}$.

Determining the smallest distance to the back-bottom corners of the container also determines the *maximal-space* Insertion Vertex, V, since it is the closest vertex to one of two back-bottom corners of the container (Fig. 7). The Insertion Vertex determines the vertex of the *maximal space* where the build layer (see step 3) will be placed. Fig. 8 illustrates two potential placements of a layer in a *maximal-space*.

This rule is similar to the back-bottom-left placement rule commonly use in CLP algorithms. However, by introducing a new potential insertion vertex, another surface of the container surface may be used, inducing the generation of larger empty spaces (Parreño et al., 2008).

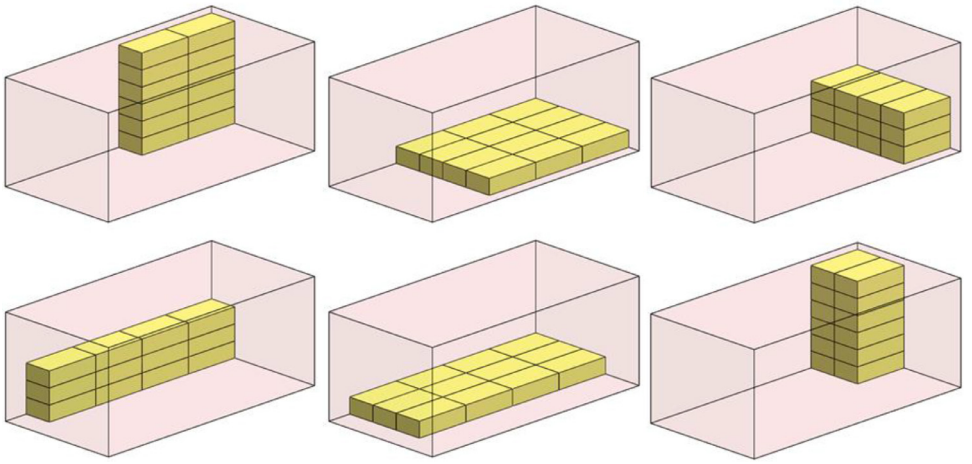

Fig. 9. Example of the six different feasible layer types for a box position.

Step 3: Filling the maximal-space

Having selected the box type and the *maximal-space*, the heuristic fills the *maximal-space* using box layers (that is, several identical boxes are arranged in rows and columns), constrained by the number of unpacked boxes. There are 6 possible layer arrangements for each box position (Fig. 9). Considering that each box can be placed in up to 6 different positions (depending on the number of rotations allowed), a total of 36 layer types can be generated. When using the weakly heterogeneous indirect representation of a solution, this value is reduced to a maximum of 12, since the building plane is determined by the chromosome. In the strongly heterogeneous indirect representation, the number of boxes per type is smaller, resulting in layers of smaller dimension which reduces the impact of the layer building plane on the solution. The Best-fit criterion proposed by Parreño et al. (2008) was used to evaluate the layer configurations.

- The **Best-fit** criterion selects the layer with the best fit inside a *maximal-space*. The layer fit is determined by calculating the gap between the parallel faces of the layer and the *maximal-space*, ordering the values in non-decreasing order and choosing the best layer as given by the lexicographic order. Ties are broken using the number of inserted boxes criterion, (that is, the layer with the higher number of boxes).

If the selected box type does not fit the selected *maximal-space*, a new *maximal-space* is selected. If the selected box type does not fit any of the current *maximal-spaces* the box is skipped and the next box on the sequence is selected.

Step 4: Updating the system information

After packing a layer of m boxes of type k, the number of unpacked boxes q_k is updated. Also, the selected *maximal-space* s is removed from list S, new *maximal-spaces* are generated using the difference process and elimination process developed by Lai and Chan (1997). These processes not only generate new *maximal-spaces* from s, but also verify the intersection of the added layer with other *maximal-spaces* and eliminate *maximal-spaces* with dimensions that cannot be filled with the remaining unpacked boxes, thus saving computational time. The list of *maximal-spaces, S*, is then updated. The placement procedure is repeated until all the boxes are packed or there are no more *maximal-spaces* available.

3.3. Solution fitness computation

The constructive heuristic finishes by determining the percentage of the container volume packed using (2), where N_k is the number of loaded boxes of type k with volume $d_k \times w_k \times h_k$ packed in a container of volume $D \times W \times H$.

$$\frac{\sum_{k=1}^{K} N_k (d_k \times w_k \times h_k)}{D \times W \times H} \times 100\% \tag{2}$$

3.4. Static stability constraint

The static stability constraint is enforced by introducing a set of conditions during the filling of the *maximal-space* and the computation of the packed container volume, based on the static mechanical equilibrium conditions applied to rigid bodies that derive from Newton's first and third laws of motion.

The first condition of equilibrium is

$$\sum \vec{F} = \vec{0} \tag{3}$$

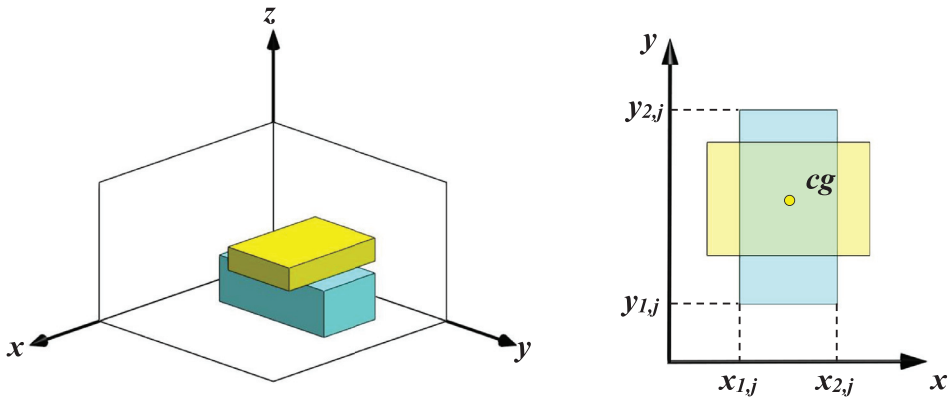

Fig. 10. Condition 2 support polygon example.

where $\vec{F}$ represents the external forces applied to a rigid body. A force is a vector quantity describing the force magnitude and the direction of its action (Hibbeler, 2010).

The second condition of equilibrium is

$$\sum \overrightarrow{M_O} = \sum (\vec{r} \times \vec{F}) = \vec{0} \tag{4}$$

where $\vec{r}$ represents the vector from point O to the line of action of force $\vec{F}$. $\overrightarrow{M_O}$ represents the moment of a force, that is, the tendency of a force to rotate a body about a point or axis. The first condition allows us to evaluate the translational equilibrium and the second condition allows us to evaluate the rotational equilibrium of a body (Hibbeler, 2010).

These conditions, however, can only be fully validated with the complete loading arrangement. Therefore, in the proposed algorithm with static stability constraints, stability is first partially enforced in the *maximal-space* filling procedure, and secondly, in the fitness computation procedure, once a solution has been generated.

3.4.1. Filling the maximal-space

The partial static stability condition, enforced when filling the *maximal-spaces*, only evaluates the stability of a box in relation to its direct supporting boxes, when is placed in the selected *maximal-space*. As such, it considers that for every box, the resultant of the forces acting downwards, parallel to the z-axis (that is, the weight), is located on the geometric center of the box. Therefore, a box b can be loaded at (x, y, z), if the projection of its geometric center cg, with coordinates (x_{cg}, y_{cg}, z_{cg}) in plane $(0, 0, z)$, lies inside the box b support polygon SP. The support polygon SP of box b is formed by the convex hull of all horizontal support points of box b. The support polygon concept is frequently used in the research field of human movement simulation for stability modelling purposes (Badler et al., 1980; Vukobratović and Borovac, 2004). Therefore, to guarantee that the partial static stability condition during filling holds, one of the three following conditions must be satisfied.

1. The support polygon of the box is the container floor, that is, whenever z (the z-axis coordinate of a box) equals zero;

$$z = 0$$

2. The support polygon is defined by the top edges of a support box b_j (Fig. 10); The condition holds for box b, supported by box b_j placed at $((x_{1j}, y_{1j}, z_{1j}), (x_{2j}, y_{2j}, z_{2j}))$, when

$$x_{1j} \leq x_{cg} \leq x_{2j}$$

$$y_{1j} \leq y_{cg} \leq y_{2j}$$

$$z = z_{2j}$$

3. the support polygon is a convex polygon defined by the convex hull of the vertices of the polygons generated by the intersection of box b with its supporting boxes (Fig. 11). Consider SP to be a convex polygon in plane $(0, 0, z)$ with m vertices $(p_0, p_1, \ldots, p_{m-1})$ defined by a sequence of points $(x_0, y_0), (x_1, y_1), \ldots, (x_{m-1}, y_{m-1})$ running in counterclockwise direction. Condition 3 holds whenever point cg is an interior point of the convex polygon SP. Point cg is an interior point of the convex polygon SP if cg is located on a line segment that connects a pair of points a, b in SP, such that $cg = \lambda a + (1 - \lambda)b$ for $0 \leq \lambda \leq 1$ (LaValle, 2006).

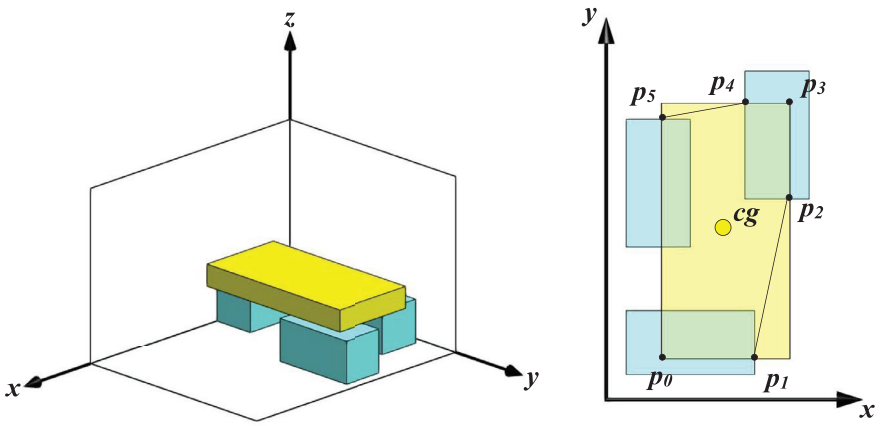

Fig. 11. Condition 3 support polygon example.

Algorithm 1 Static Stable Filling of a *Maximal-Space* (box b)

Input: Let box b be a box placed at coordinates $((x_1, y_1, z_1), (x_2, y_2, z_2))$
Output: Let *stable* be a boolean variable where TRUE represents a box that respects the stable filling conditions and FALSE otherwise

Begin
if $z = 0$ **then**
  return TRUE
end if
stable ← FALSE
Let U be the set of boxes b_j that support box b
Let cg be the geometric centre of box b with coordinates x_{cg}, y_{cg}
for each box $b_i \in U$ **do**
  if $(x_{1j} \leq x_{cg} \leq x_{2j})$ and $(y_{1j} \leq y_{cg} \leq y_{2j})$ **then**
    stable ← TRUE
  end if
end for
if *stable* = FALSE **then**
  for each box $b_j \in U$ **do**
    Determine the intersection vertices v with box b
  end for
  SP ← **Call** Gift wrapping (v) ▷ Determine box b support polygon
  stable ← **Call** Point-in-Polygon (cg, SP) ▷ Determine box stability
end if
return *stable*
End

The algorithm for enforcing the stability condition during the filling of a *maximal-space* is described in Algorithm 1. A more detailed description of the *Gift wrapping* algorithm, that determines the support polygon of a box, and the *Point-in-Polygon* algorithm, that determines if a given point is in the interior of a convex polygon, is presented in Ramos et al. (2016).

To the best of the authors' knowledge, there is not any approach in the literature that combines the use of *maximal-spaces* and the evaluation of static stability, when there are gaps between supporting boxes, as proposed in this paper. Adding this possibility adds complexity to the management and filling of *maximal-spaces*, since it is required to determine the support boxes of the *maximal-space* in order to fill it and evaluate the fitness of the build layer configuration. However, this approach increases the flexibility of the algorithm.

The criterion used to evaluate layer configurations is the *best-overhanging* criterion.

- The **best-overhanging** criterion selects the layer which best fits the supporting boxes of the selected *maximal-space*. The layer fit is determined by calculating the gap between the parallel faces of the layer that are perpendicular to the $x - y$ plane and the support polygon of the selected *maximal-space*, ordering the values in non-decreasing order and choosing

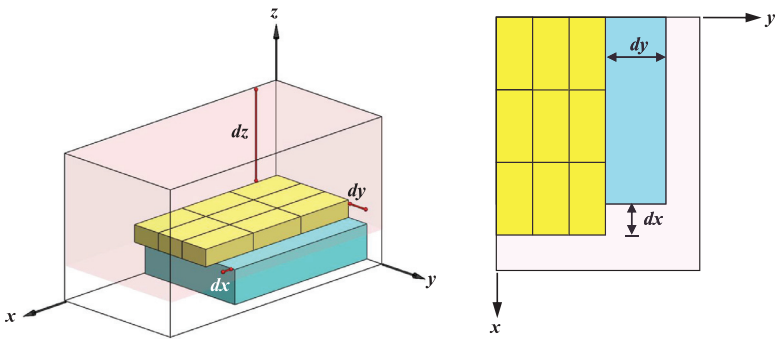

Fig. 12. Best-overhanging criterion illustration example.

the best layer, given by the lexicographic order. Ties are broken by first evaluating the gap in the *z*-axis direction and secondly by calculating the additional number of boxes to be placed. Fig. 12 illustrates the criterion.

Even thought, the partial stability conditions, by themselves, do not guarantee the static stability of the cargo arrangement, when combined with the best-overhanging criterion, potentiate the generation of statically stable solutions.

3.4.2. Solution fitness computation

As in other approaches previously published in the literature, the metric used to evaluate the fitness of a solution is the percentage of the container volume packed by boxes. The difference in this algorithm is that only boxes that can be loaded in the container while guaranteeing static stability are considered, that is, boxes that would make the cargo unstable are withdraw from the physical packing sequence and their volume not considered.

Given that static stability conditions were already applied during the *maximal-spaces* filling procedure, how can still happen that a cargo is unstable? The conditions applied during the *maximal-space* filling procedure guarantee local stability, that is, that the arrangement inside the *maximal-space* is stable. However, the effect of filling that space on the remaining cargo is not taken into account because it would be unnecessarily restrictive. As the order by which *maximal-spaces* are filled during solution generation is not the actual sequence by which boxes are loaded inside the container, when filling later on spaces physically related to the one that is currently being filled may revert a situation of potential instability into a globally stable cargo. For this reason, in the algorithm here proposed, static stability is locally enforced when filling the *maximal-spaces* and globally checked and imposed after the loading arrangement is finished, when the solution fitness is computed.

The final verification of the static stability of the finished loading arrangement is, therefore, done by the physical packing sequence algorithm (PPSA) with static stability as presented in Ramos et al. (2016). The PPSA, given a container loading arrangement, generates a physical sequence by which each box can be actually loaded inside the container, considering static stability and efficiency of loading operations constraints. The approach to stability used in the PPSA is also based on the static mechanical equilibrium conditions applied to rigid bodies, derived from Newton's laws of motion. In generic terms, PPSA starts by determining a physical sequence by which boxes should be loaded in the container. Using this sequence, it loads one box at a time and for each one evaluates the static stability of the box and the static stability of the subset of all boxes already loaded. If, as a result of loading this new box, statical stability of the overall cargo is lost, then the box is removed from de physical packing sequence. As PPSA deals with a physical loading sequence and considers the effect of each box on all the previous loaded cargo, global static stability is guaranteed. As PPSA removes from the sequence (and from the solution) unstable boxes, the solution fitness (percentage of the container's space that is used) reflects only statically stable boxes.

4. Computational experiments

This section presents the results of the computational experiments run to evaluate the efficiency of the proposed container loading algorithm with static stability (CLA-SS). The proposed algorithm is run without any static stability constraint (Unsupported), with the classical full base support constraint (Full Support) and with the new static stability constraint (Supported), with the results then being compared among themselves. These results are also compared with the most recent and efficient approaches to the CLP. The algorithm was coded in Visual C++ and run on an computer with 2 Intel Xeon CPU E5-2687W at 3.1 Ghz with 128 Gigabytes of RAM running the Windows 7 Pro 64 bit operating system.

Two versions of the algorithm are tested contrasting the use of strongly heterogeneous (CLA-SS(S)) and weakly heterogeneous (CLA-SS(W)) chromosome indirect representations. We start by comparing the CLA-SS(S) and CLA-SS(W) Unsupported and Full Support variants with the most efficient algorithms. Then we compare the CLA-SS Unsupported, Full Support and Supported variants, for both CLA-SS(W) and CLA-SS(W) versions. Finally, we compare the Supported variant of the two CLA-SS versions with the most efficient algorithms that guarantee full base support, and thus indirectly also guarantee static

Table 2
Genetic algorithm parameters used in all computational experiments.

Parameters	Values
Top	15%
Bottom	15%
Crossover probability	0.7
Population size	20 × number of boxes
Number of populations	2
Exchange between pop.	Every 15 generations
Fitness function	Maximize the % of packed container volume
Stopping criteria	after 1000 generations

stability. The column headings for the computational results in the tables that follow refer to the different algorithms used, namely: BSG—the Beam Search Approach of Araya and Riff (2014); HBMLS—the block-loading heuristic based on multi-layer search of Zhang et al. (2012) (two versions are presented, the AS version that uses simple blocks and the AC version that uses composite bocks); ID-GLTS—the iterative-doubling greedy-lookahead algorithm of Zhu and Lim (2012); BRKGA—the multi-population biased random-key genetic algorithm of Gonçalves and Resende (2012); and CLA-SS—the proposed container loading algorithm with static stability.

4.1. Test problem instances

The problem tests used to evaluate the effectiveness of the new algorithm are the 1500 problems proposed by Bischoff and Ratcliff (1995) and Davies and Bischoff (1999). These instances are organized in 15 classes, with a total of 100 instances per class. They are designated here as BR1 to BR15. The instances used cover a wide range of situations. The heterogeneity of the boxes increases from just 3 different box types in BR1 to 100 box types in BR15. The number of boxes per box type also varies from 50.15 boxes per type in BR1 to 1.33 in BR15. The dimensions of the boxes are generated independently from the dimensions of the container and the total volume of the boxes in each individual instance never exceeds the container volume. On average the total volume of the boxes to be packed represents 99.46% of the container volume. All tables presenting computational results show the average solution value, in terms of percentage of container space utilization, for the 100 instances of each class, the aggregate results for the class instances BR1–7 and BR8–15, and the aggregate results for the class instances BR1–15.

4.2. Genetic algorithm parameters

The genetic algorithm parameters used in the algorithm are based on the recommended parameter value settings proposed by Gonçalves and Resende (2011) for generic BRKGA, and the parameters used by Gonçalves and Resende (2012) and Gonçalves and Resende (2013) for BRKGA developed for the CLP. Preliminary computational experiments allowed the selection of the parameters presented in Table 2.

4.3. CLA-SS performance compared to other CLP algorithms

The first set of experiments aimed to compare the two CLA-SS versions against the most efficient algorithms, that can be found in the literature, using both Unsupported and Full Support variants. Comparing the results (presented in Table 3) of the Unsupported variant against the equivalent algorithms (BSG, HBMLS(AS), HBMLS(AS) and ID-GLTS) there is an overall difference to the best (BSG) of −0.98 percentage points, for the CLA-SS(S), and −0.69 for the CLA-SS(W). It must be stated that the BSG outperforms all the algorithms in the Unsupported variant. A comparison against the original BRKGA algorithm shows an overall difference of −0.12 percentage points to the CLA-SS(S) and of 0.17 to the CLA-SS(W). It can be observed that both CLA-SS versions underperform against the BRKGA for weakly heterogeneous instances, and outperform for the strongly heterogeneous ones.

Analysing the results for the Full Support variant (presented in Table 4) show that the equivalent algorithms with the best performance, BSG and HBMLS(AC), have an overall difference to the CLA-SS(S) of −0.53 and −0.29 percentage points respectively and an overall difference to the CLA-SS(W) of −0.05 and 0.19 percentage points; comparing with the results for the original BRKGA algorithm there is an overall difference of 0.10 percentage points to the CLA-SS(S) and 0.58 to the CLA-SS(W). Notably, the CLA-SS(W) presents the best overall results for the BR8–15 classes of problems.

These results show that the proposed versions of the CLA-SS algorithm have a level of performance similar to the best-in-class CLP algorithms, both in the Unsupported and Full Support variants.

The comparison of computational times between the different approaches would only provide significant information if the software programming technologies, such as the use of parallel programming or memory management, were similar, and the tests run on identical computers and operating systems. Regardless this fact, Table 5 presents the reported average running time of each approach.

Table 3
Performance comparison of Unsupported variants.

Class Problems	BSG (2014)	HBMLS (AS) (2012)	HBMLS (AC) (2012)	ID-GLTS (2012)	BRKGA (2010)	CLA-SS(S)	CLA-SS(W)
BR_1	**95.69**	94.87	93.54	95.59	95.28	93.54	95.10
BR_2	**96.24**	95.41	94.47	96.13	95.90	94.83	95.76
BR_3	**96.49**	95.56	95.12	96.30	96.13	95.40	95.85
BR_4	**96.31**	95.38	95.10	96.15	96.01	95.38	95.74
BR_5	**96.18**	95.22	95.08	95.98	95.84	95.43	95.65
BR_6	**96.05**	95.10	95.21	95.81	95.72	95.36	95.61
BR_7	**95.77**	94.69	94.87	95.36	95.29	95.18	95.32
BR_8	**95.33**	94.16	94.60	94.80	94.76	94.80	95.07
BR_9	**95.07**	93.76	94.24	94.53	94.34	94.64	94.77
BR_10	**94.97**	93.38	94.08	94.35	93.86	94.32	94.47
BR_11	**94.80**	92.87	93.86	94.14	93.60	94.01	94.14
BR_12	**94.64**	92.59	93.67	94.10	93.22	93.77	93.93
BR_13	**94.59**	92.25	93.45	93.86	92.99	93.56	93.37
BR_14	**94.49**	91.84	93.34	93.83	92.68	93.28	93.22
BR_15	**94.37**	91.53	93.14	93.78	92.46	92.81	92.65
Mean (BR 1–7)	**96.11**	95.18	94.77	95.90	95.74	95.02	95.58
Mean (BR 8–15)	**94.78**	92.80	93.80	94.17	93.49	93.90	93.95
Mean (BR 1–15)	**95.40**	93.91	94.25	94.98	94.54	94.42	94.71

∗ The best values appear in bold.

Table 4
Performance comparison of Full Support variants.

Class Problems	BSG (2014)	HBMLS (AS) (2012)	HBMLS (AC) (2012)	ID-GLTS (2012)	BRKGA (2010)	CLA-SS(S)	CLA-SS(W)
BR_1	**94.50**	94.30	93.95	94.40	94.34	92.36	93.86
BR_2	**95.03**	94.74	94.39	94.85	94.88	93.68	94.55
BR_3	**95.17**	94.89	94.67	95.10	95.05	94.21	94.75
BR_4	**94.97**	94.69	94.54	94.81	94.75	94.23	94.63
BR_5	**94.80**	94.53	94.41	94.52	94.58	94.05	94.38
BR_6	**94.65**	94.32	94.25	94.33	94.39	93.87	94.24
BR_7	**94.09**	93.78	93.69	93.59	93.74	93.26	93.82
BR_8	93.15	92.88	93.13	92.65	92.65	92.64	**93.16**
BR_9	92.53	92.07	92.54	92.11	91.90	92.13	**92.62**
BR_10	92.04	91.28	92.02	91.60	91.28	91.62	**92.09**
BR_11	91.40	90.48	91.45	90.64	90.39	91.19	**91.56**
BR_12	90.92	89.65	90.91	90.35	89.81	90.91	**91.28**
BR_13	90.51	88.75	90.43	89.69	89.27	90.66	**90.93**
BR_14	89.93	87.81	89.80	89.07	88.57	90.31	**90.38**
BR_15	89.33	86.94	89.24	88.36	87.96	90.01	**90.08**
Mean (BR 1–7)	**94.74**	94.46	94.27	94.51	94.53	93.66	94.32
Mean (BR 8–15)	91.22	89.98	91.19	90.56	90.23	91.18	**91.51**
Mean (BR 1–15)	**92.87**	92.07	92.63	92.40	92.24	92.34	92.82

∗ The best values appear in bold.

Table 5
Average computational times (s) for test classes BR1 to BR15.

	BSG	HBMLS	ID-GLTS	BRKGA	CLA-SS(W)
Unsupported	500	597	500	147	274
Full Support	150	519	150	232	146

4.4. CLA-SS performance for the different versions

A second set of experiments evaluates the performance of the two CLA-SS versions (CLA-SS(S) and CLA-SS(W)) across the Unsupported, Supported and Full Support variants. The average results obtained are presented in Table 6. In the case of the Unsupported variant, the results show that the CLA-SS(S) performs better over the classes BR13 to BR15; in the case of the Supported variant the CLA-SS(S) performs better over the classes BR9 to BR15; and in the case of the Full Support variant, the CLA-SS(W) always returns the best performance. It can also be observed that for each class, the average Unsupported results are higher than the comparable results of the Supported variant, which in turn are higher than the respective Full Support results.

Table 6
Performance comparison of CLA-SS versions.

Class	Unsupported		Supported		Full Support	
Problems	CLA-SS(S)	CLA-SS(W)	CLA-SS(S)	CLA-SS(W)	CLA-SS(S)	CLA-SS(W)
BR_1	93.54	**95.10**	93.16	**94.79**	92.36	**93.86**
BR_2	94.83	**95.76**	94.73	**95.40**	93.68	**94.55**
BR_3	95.40	**95.85**	95.21	**95.55**	94.21	**94.75**
BR_4	95.38	**95.74**	95.29	**95.51**	94.23	**94.63**
BR_5	95.43	**95.65**	95.15	**95.43**	94.05	**94.38**
BR_6	95.36	**95.61**	95.09	**95.31**	93.87	**94.24**
BR_7	95.18	**95.32**	94.93	**95.14**	93.26	**93.82**
BR_8	94.80	**95.07**	94.69	**94.78**	92.64	**93.16**
BR_9	94.64	**94.77**	**94.51**	94.45	92.13	**92.62**
BR_10	94.32	**94.47**	**94.07**	93.95	91.62	**92.09**
BR_11	94.01	**94.14**	**93.68**	93.38	91.19	**91.56**
BR_12	93.77	**93.93**	**93.23**	92.61	90.91	**91.28**
BR_13	**93.56**	93.37	**92.59**	91.64	90.66	**90.93**
BR_14	**93.28**	93.22	**91.68**	90.72	90.31	**90.38**
BR_15	**92.81**	92.65	**90.58**	90.10	90.01	**90.08**
Mean (BR 1–7)	95.02	**95.58**	94.79	**95.30**	93.66	**94.32**
Mean (BR 8–15)	93.90	**93.95**	**93.13**	92.70	91.18	**91.51**
Mean (BR 1–15)	94.42	**94.71**	93.91	**93.92**	92.34	**92.82**

∗ The best values of each variant (that is, Unsupported, Supported and Full Support) appear in bold.

Table 7
Number of times the CLA-SS versions provided better solutions.

Class	Unsupported		Supported		Full Support	
Problems	CLA-SS(S)	CLA-SS(W)	CLA-SS(S)	CLA-SS(W)	CLA-SS(S)	CLA-SS(W)
BR_1	2	96	4	94	5	86
BR_2	5	93	18	81	13	85
BR_3	22	78	32	68	18	82
BR_4	25	75	35	65	31	68
BR_5	32	68	32	68	32	68
BR_6	28	70	33	65	31	69
BR_7	40	60	33	65	28	72
BR_8	25	72	45	55	22	78
BR_9	41	59	52	48	22	78
BR_10	44	56	55	45	23	76
BR_11	42	56	53	47	33	67
BR_12	30	70	63	37	27	73
BR_13	53	46	71	29	35	63
BR_14	52	48	69	31	49	51
BR_15	59	41	59	41	46	53
Mean (BR 1–7)	22.0	77.1	26.7	72.3	22.6	75.7
Mean (BR 8–15)	43.3	56.0	58.4	41.6	32.1	67.4
Mean (BR 1–15)	33.3	65.9	43.6	55.9	27.7	71.3

Table 7 presents the number of times for each class and variant, that each of the CLA-SS versions outperforms the other. All three CLA-SS(W) variants outperform the CLA-SS(S) variants in the weakly heterogeneous classes of instances. As for the strongly heterogeneous classes, CLA-SS(S) outperforms CLA-SS(W) in the Supported variant. However in the Unsupported and Full Support variants it is CLA-SS(W) that has the best overall performance.

The impact of the number of generations of the algorithm is depicted in Fig. 13. The evolution of the solution is fastest under the Full Support variant, with the Unsupported and the Supported variants requiring progressively more iterations.

4.5. Performance of statically stable algorithms

Finally, the CLP algorithms with the best performance in the Full Support variant (BSG and CLA-SS(W)), that is, with static stability, are compared with the two versions of the CLA-SS Supported variant. The average results obtained are presented in Table 8. The Supported variant CLA-SS(W), had the best average performance for BR1 to BR8 classes of instances, while the CLA-SS(S) version outperformed all approaches for BR9 to BR15 classes of instances.

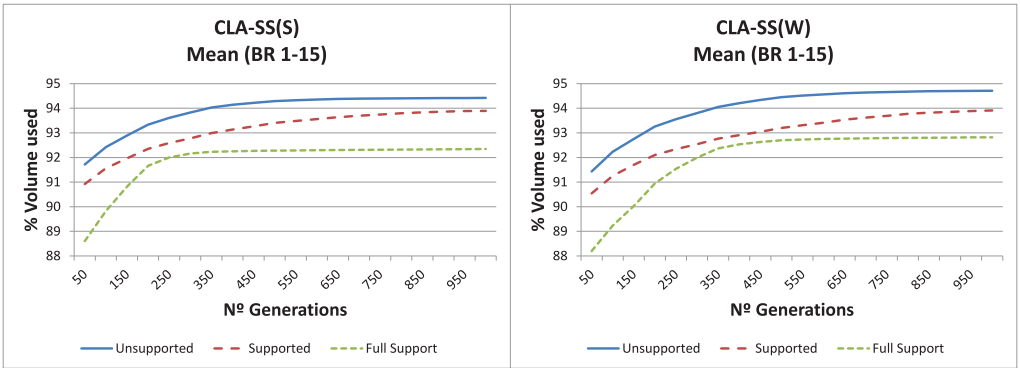

Fig. 13. Influence of the number of generations.

Table 8
Performance comparison of statically stable solutions.

Class Problems	BSG (2014)	CLA-SS(W) Full Support	CLA-SS(S) Supported	CLA-SS(W) Supported
BR_1	94.50	93.86	93.16	**94.79**
BR_2	95.03	94.55	94.73	**95.40**
BR_3	95.17	94.75	95.21	**95.55**
BR_4	94.97	94.63	95.29	**95.51**
BR_5	94.80	94.38	95.15	**95.43**
BR_6	94.65	94.24	95.09	**95.31**
BR_7	94.09	93.82	94.93	**95.14**
BR_8	93.15	93.16	94.69	**94.78**
BR_9	92.53	92.62	**94.51**	94.45
BR_10	92.04	92.09	**94.07**	93.95
BR_11	91.40	91.56	**93.68**	93.38
BR_12	90.92	91.28	**93.23**	92.61
BR_13	90.51	90.93	**92.59**	91.64
BR_14	89.93	90.38	**91.68**	90.72
BR_15	89.33	90.08	**90.58**	90.10
Mean (BR 1–7)	94.74	94.32	94.79	**95.30**
Mean (BR 8–15)	91.22	91.51	**93.13**	92.70
Mean (BR 1–15)	92.87	92.82	93.91	**93.92**

∗ The best values appear in bold.

5. Conclusion

In this paper we addressed the static stability constraint within the three-dimensional rectangular single CLP. We proposed two versions of an hybrid genetic algorithm based on a multi-population biased random key genetic algorithm and a constructive heuristic that uses a two dimensional box arrangement strategy and a *maximal-spaces* representation of empty spaces inside the container. A new procedure for filling the *maximal-spaces*, based on the static mechanical equilibrium conditions applied to rigid bodies derived from Newton's laws of motion, that allows the evaluation of stability when there are gaps between supporting boxes, was also proposed. The two versions of the algorithm were tested using the well known benchmark instances of Bischoff and Ratcliff (1995) and Davies and Bischoff (1999) and compared to the best known CLP solutions in the literature. The proposed approach improved the best known average results for the instances BR-8 to BR15 from 91.22% to 91.51%, using the full base support constraint, and improved the overall average of the solutions from 92.87% to 93.92%, for statically stable solutions.

Acknowledgements

This work was partially funded by Project "TEC4Growth - Pervasive Intelligence, Enhancers and Proofs of Concept with Industrial Impact/NORTE-01-0145-FEDER-000020", financed by the North Portugal Regional Operational Programme (NORTE 2020) under the PORTUGAL 2020 Partnership Agreement, and through the European Regional Development Fund (ERDF), and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EMS/00615/2013.

References

Araya, I., Riff, M.C., 2014. A beam search approach to the container loading problem. Comput. Oper. Res. 43, 100–107.
Badler, N., O'Rourke, J., Kaufman, B., 1980. Special problems in human movement simulation. ACM SIGGRAPH Comput. Graph. 14 (3), 189–197.
Bischoff, E.E., 2006. Three-dimensional packing of items with limited load bearing strength. Eur. J. Oper. Res. 168 (3), 952–966.
Bischoff, E.E., Janetz, F., Ratcliff, M.S.W., 1995. Loading pallets with non-identical items. Eur. J. Oper. Res. 84 (3), 681–692.
Bischoff, E.E., Ratcliff, M.S.W., 1995. Issues in the development of approaches to container loading. Omega 23 (4), 377–390.
Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. Eur. J. Oper. Res. 131 (1), 143–161.
Bortfeldt, A., Gehring, H., Mack, D., 2003. A parallel tabu search algorithm for solving the container loading problem. Parallel Comput. 29 (5), 641–662.
Bortfeldt, A., Wäscher, G., 2013. Constraints in container loading-a state-of-the-art review. Eur. J. Oper. Res. 229 (1), 1–20.
Carpenter, H., Dowsland, W.B., 1985. Practical considerations of the pallet-loading problem. J. Oper. Res. Soc. 36 (6), 489–497.
de Castro Silva, J.L., Soma, N.Y., Maculan, N., 2003. A greedy search for the three-dimensional bin packing problem: the packing static stability case. Int. Trans. Oper. Res. 10 (2), 141–153.
Christensen, S.G., Rousœ, D.M., 2009. Container loading with multi-drop constraints. Int. Trans. Oper. Res. 16 (6), 727–743.
Davies, A., Bischoff, E.E., 1999. Weight distribution considerations in container loading. Eur. J. Oper. Res. 114 (3), 509–527.
Fanslau, T., Bortfeldt, A., 2010. A tree search algorithm for solving the container loading problem. INFORMS J. Comput. 22 (2), 222–235.
Fekete, S.P., Schepers, J., van der Veen, J.C., 2007. An exact algorithm for higher-dimensional orthogonal packing. Oper. Res. 55 (3), 569–587.
Fuellerer, G., Doerner, K.F., Hartl, R.F., Iori, M., 2010. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. Eur. J. Oper. Res. 201 (3), 751–759.
Gehring, H., Bortfeldt, A., 1997. A genetic algorithm for solving the container loading problem. Int. Trans. Oper. Res. 4 (5–6), 401–418.
Gendreau, M., Iori, M., Laporte, G., Martello, S., 2006. A tabu search algorithm for a routing and container loading problem. Transp. Sci. 40 (3), 342–350.
George, J., Robinson, D., 1980. A heuristic for packing boxes into a container. Comput. Oper. Res. 7 (3), 147–156.
Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. J. Heuristics 17, 487–525.
Gonçalves, J.F., Resende, M.G.C., 2012. A parallel multi-population biased random-key genetic algorithm for a container loading problem. Comput. Oper. Res. 39, 179–190.
Gonçalves, J.F., Resende, M.G.C., 2013. A biased random key genetic algorithm for 2d and 3d bin packing problems. Int. J. Prod. Econ. 145 (2), 500–510.
Hibbeler, R.C., 2010. Engineering Mechanics: Statics. Prentice Hall, Upper Saddle River, NJ.
Junqueira, L., Morabito, R., Sato Yamashita, D., 2012a. MIP-based approaches for the container loading problem with multi-drop constraints. Ann. Oper. Res. 199 (1), 51–75.
Junqueira, L., Morabito, R., Sato Yamashita, D., 2012b. Three-dimensional container loading models with cargo stability and load bearing constraints. Comput. Oper. Res. 39 (1), 74–85.
Lai, K., Chan, J., 1997. Developing a simulated annealing algorithm for the cutting stock problem. Comput. Ind. Eng. 32 (1), 115–127.
LaValle, S.M., 2006. Planning Algorithms. Cambridge University Press.
Lim, a., Rodrigues, B., Wang, Y., 2003. A multi-faced buildup algorithm for three-dimensional packing problems. Omega 31 (6), 471–481.
Lin, J.-L., Chang, C.-H., Yang, J.-Y., 2006. A study of optimal system for multiple-constraint multiple-container packing problems. In: Ali, M., Dapoigny, R. (Eds.), Advances in Applied Artificial Intelligence. Lecture Notes in Computer Science, vol. 4031. Springer, Berlin / Heidelberg, pp. 1200–1210.
Liu, J., Yue, Y., Dong, Z., Maple, C., Keech, M., 2011. A novel hybrid tabu search approach to container loading. Comput. Oper. Res. 38 (4), 797–807.
Mack, D., Bortfeldt, A., Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. Int. Trans. Oper. Res. 11, 511–533.
Meng, Q., Wang, X., 2011. Intermodal hub-and-spoke network design: Incorporating multiple stakeholders and multi-type containers. Transp. Res. Part B 45 (4), 724–742.
Moura, A., Oliveira, J.F., 2005. A GRASP approach to the container-loading problem. IEEE Intell. Syst. 20 (4), 50–57.
Ngoi, B.K.A., Tay, M.L., Chua, E.S., 1994. Applying spatial representation techniques to the container packing problem. Int. J. Prod. Res. 32 (1), 111–123.
Padberg, M., 2000. Packing small boxes into a big box. Math. Methods Oper. Res. (ZOR) 52 (1), 1–21.
Parreño, F., Alvarez-Valdes, R., Tamarit, J.M., Oliveira, J.F., 2008. A maximal-space algorithm for the container loading problem. INFORMS J. Comput. 20 (3), 412–422.
Ramos, A.G., Oliveira, J.F., Gonçalves, J.F., Lopes, M.P., 2015. Dynamic stability metrics for the container loading problem. Transp. Res. Part C 60, 480–497.
Ramos, A.G., Oliveira, J.F., Lopes, M.P., 2016. A physical packing sequence algorithm for the container loading problem with static mechanical equilibrium conditions. Int. Trans. Oper. Res. 23, 215–238.
Sánchez-Díaz, I., Holguín-Veras, J., Ban, X.J., 2015. A time-dependent freight tour synthesis model. Transp. Res. Part B 78, 144–168.
Tarantilis, C., Zachariadis, E., Kiranoudis, C., 2009. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. IEEE Trans. Intell. Transp. Syst. 10 (2), 255–271.
Vukobratović, M., Borovac, B., 2004. Zero-moment point thirty five years of its life. Int. J. Humanoid Robot. 01 (01), 157–173.
Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. Eur. J. Oper. Res. 183 (3), 1109–1130.
Zhang, D., Peng, Y., Leung, S.C., 2012. A heuristic block-loading algorithm based on multi-layer search for the container loading problem. Comput. Oper. Res. 39 (10), 2267–2276.
Zhang, Z., Wei, L., Lim, A., 2015. An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. Transp. Res. Part B 82, 20–35.
Zhu, W., Lim, A., 2012. A new iterative-doubling greedylookahead algorithm for the single container loading problem. Eur. J. Oper. Res. 222 (3), 408–417.