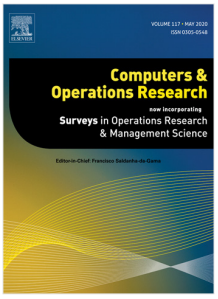
Journal Pre-proof



A block-building constraint programming model for the container loading problem

Chang Liu, Kate Smith-Miles, Tony Wauters, Alysson M. Costa

PII: S0305-0548(25)00139-X
DOI: https://doi.org/10.1016/j.cor.2025.107111
Reference: CAOR 107111

To appear in: *Computers and Operations Research*

Received date : 1 November 2024
Revised date : 19 March 2025
Accepted date : 22 April 2025

Please cite this article as: C. Liu, K. Smith-Miles, T. Wauters et al., A block-building constraint programming model for the container loading problem. *Computers and Operations Research* (2025), doi: https://doi.org/10.1016/j.cor.2025.107111.

This is a PDF file of an article that has undergone enhancements after acceptance, such as the addition of a cover page and metadata, and formatting for readability, but it is not yet the definitive version of record. This version will undergo additional copyediting, typesetting and review before it is published in its final form, but we are providing this version to give early visibility of the article. Please note that, during the production process, errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

© 2025 Published by Elsevier Ltd.

Manuscript

Click here to view linked References

Journal Pre-proof

A Block-Building Constraint Programming Model for the Container Loading Problem

Chang Liu[a,b,c,*], Kate Smith-Miles[a,c], Tony Wauters[b], Alysson M. Costa[a,c]

[a] School of Mathematics and Statistics, The University of Melbourne, Parkville, Melbourne, Australia.
[b] Department of Computer Science, CODeS, KU Leuven, Gebroeders De Smetstraat 1, 9000 Ghent, Belgium.
[c] ARC Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), The University of Melbourne, Parkville, Melbourne, Australia.
[*] Corresponding author
Email addresses: cliu2@student.unimelb.edu.au (C. Liu), smith-miles@unimelb.edu.au (K. Smith-Miles), tony.wauters@kuleuven.be (T. Wauters), alysson.costa@unimelb.edu.au (A. M. Costa)
Declarations of interest: none

Journal Pre-proof

Journal Pre-proof

Abstract

The container loading problem involves packing a set of given rectangular boxes into a larger rectangular container of fixed size, with the objective of maximizing the volume of the loaded boxes. Most of the literature on the container loading problem and its variants proposes heuristic approaches that can find good solutions quickly. Current exact methods are mostly limited to mixed-integer programming (MIP) formulations, which often struggle to obtain good solutions for large problem instances.

In this paper, we introduce two exact constraint programming models for the container loading problem. The first model uses integer and binary variables to assign boxes to valid positions and orientations within the container. The second model enhances this by incorporating the concept of block-building, commonly used in heuristic methods. Extensive computational experiments on classical benchmark instances from the literature show that the solutions obtained with the proposed models significantly outperform those achieved with existing MIP models. We also perform an instance space analysis of the proposed models to map the models' performances across problem instances, providing deeper insights into the strengths and weaknesses of the block-building approach.

Keywords: Packing, Container loading problem, Constraint programming, Mathematical modeling, Block building, Instance Space Analysis

1. Introduction

The container loading problem (CLP) is a three-dimensional (3D) packing problem that involves packing a set of given rectangular boxes into a larger rectangular container of fixed size. The boxes in the container must be aligned parallel to the axes and cannot overlap (intersect). The objective is to maximize the *volume occupancy*, defined as the proportion of the container's volume occupied by boxes. Problem instances are classified as *homogeneous* if there is only one type of box, *weakly heterogeneous* if there are a few types, and *strongly heterogeneous* if there are many types of boxes (Bischoff and Ratcliff, 1995). The number of boxes available for each type is limited.

The CLP is part of a broader class of combinatorial optimization problems known as Cutting and Packing problems. Wäscher et al. (2007) proposed a typology that divides these problems into two broad categories: Input Minimization Problems, which require using all boxes and aim to minimize the containers' value or dimensions, and Output Maximization Problems, which limit the number of containers and strive to maximize the value of the packed boxes. The CLP is classified as an Output Maximization Problem, and is further categorized as a three-dimensional identical item packing problem (3D-IIPP) for homogeneous instances, a single large object placement problem (3D-SLOPP) for weakly heterogeneous cases, and a single knapsack problem (3D-SKP) for strongly heterogeneous cases.

Journal Pre-proof

3D Cutting and Packing problems often involve additional constraints that are relevant in real-world scenarios, such as weight limits, weight distribution, orientation constraints (which determine the allowed box rotations), and support constraints (ensuring each box is supported by the base of the container or by other boxes). The survey paper by Bortfeldt and Wäscher (2013) provides a comprehensive review of these practical constraints. In this paper, we include orientation constraints by specifying the allowed rotations for each box as part of the instance data. We also present extensions of the models for the case where support constraints need to be enforced.

Since 3D Cutting and Packing problems are challenging to solve exactly, most algorithms for the CLP and its variants are heuristic methods designed to find good solutions quickly. Recent contributions include Liu et al. (2014); Galrão Ramos et al. (2016); Toffolo et al. (2017); Araya et al. (2017); Dias Saraiva et al. (2019); Bayraktar et al. (2021), which proposed various non-exact methods for these problems. The survey papers by Zhao et al. (2016); Ali et al. (2022) provide thorough reviews of solution methods proposed for 3D Cutting and Packing problems; the former compares algorithm performance across benchmark instances from the literature, while the latter emphasizes online algorithms.

A few exact approaches have also been developed for solving 3D Cutting and Packing problems. Silva et al. (2019) provides a recent comparative study of exact methods, many of which were initially developed for different variants but can be adapted for the CLP. Aside from the tailored branch-and-bound algorithm for the 3D bin packing problem by Martello et al. (2000), all the exact methods considered by Silva et al. (2019) use state-of-the-art black-box solvers on mixed-integer programming models (MIPs). The main references, as well as some more recent ones, are discussed in the next subsection.

1.1. Exact methods for 3D Cutting and Packing problems

Chen et al. (1995) proposed a MIP model for the 3D bin packing problem, an Input Minimization Problem that involves packing a given set of boxes into larger containers (bins) while minimizing the total volume of the bins used. The model accommodates multiple bin sizes and box rotation, and the paper explains how to formulate additional real-world constraints, including weight distribution and container load bearing.

Hifi et al. (2010) proposed a model for the 3D bin packing problem with identical bin sizes, that does not consider rotations or any additional constraints. Paquay et al. (2016) introduced a formulation for the bin packing problem with multiple bin sizes and box rotation, incorporating constraints relevant to air transportation, such as stability, fragility, and weight distribution.

Tsai et al. (2014) developed a MIP model for the 3D open dimension problem, an Input Minimization Problem where the container dimensions are variable and the objective is to minimize the container volume while fitting all boxes. The model, based on Chen et al. (1995), includes rotations and is reformulated to reduce the number of binary variables needed for the constraints that prevent overlapping of boxes.

Fasano (2004) proposed a MIP model for the CLP that includes rotations and balancing constraints. The paper notes that the MIP model is difficult to solve using standard techniques, and hence introduces an MIP-based heuristic approach. Similarly, Junqueira et al. (2012) proposed a MIP model for the CLP and

Journal Pre-proof

explained that it can be easily adapted to allow box rotation or the use of multiple containers. The authors also demonstrated how to include cargo stability and load-bearing constraints. Unlike the other models presented in this section, which use a combination of binary and integer variables, Junqueira et al. (2012) uses only binary variables.

More recently, Kurpel et al. (2020) proposed MIP models for both the bin packing and container loading problems. The models are based on the formulation by Junqueira et al. (2012) and account for item rotation, separation of boxes, and stability. Kurpel et al. (2020) compared four techniques for generating potential placement points for boxes, expanding on the one used in the previous work.

Martin et al. (2024) presented a constraint programming model for the 3D open dimension problem, which allows for item rotation but lacks additional constraints. In this paper, we adapt their model to the CLP and use it as a foundation for developing our own constraint programming models.

1.2. Our contributions

In this paper, we present two constraint programming models for the CLP. The first is adapted from a constraint programming approach originally proposed for the 3D open dimension problem by Martin et al. (2024). We modify this approach to address the container loading problem and ensure it is amenable to extension into our second model. The second model enhances the first by incorporating *block-building*, a concept commonly used in heuristic methods. Block-building enables multiple boxes of the same type to be placed simultaneously without gaps between them, often resulting in tighter and more efficient packing. This approach is particularly effective for homogeneous and weakly heterogeneous instances with few box types and many identical boxes, allowing the creation of large blocks.

We evaluate the performance of our models against existing MIP models using benchmark instances from the literature. Our results demonstrate that the results obtained with the proposed models outperform existing ones across all classes of test cases. We also conduct an instance space analysis (Smith-Miles and Muñoz, 2023) to investigate further how the characteristics of instances impact the benefit of the block-building approach.

The remainder of the paper is structured as follows. Section 2 presents our constraint programming models for the container loading problem, including the changes necessary to enforce support constraints. Section 3 describes the computational experiments conducted to assess the performance of the models, including comparisons with existing MIP models. Section 4 includes an instance space analysis of exact models for the container loading problem. Finally, Section 5 provides conclusions and outlines directions for future research.

2. Constraint programming models for the container loading problem

An instance of the Container Loading Problem can be fully described by the following data:

- L, W, and H: the length, width and height of the container,
- n: the number of different box types,

Journal Pre-proof

- l_i, w_i, and $h_i, \forall i \in \{1, \ldots, n\}$: the dimensions of the box types,
- $d_i, \forall i \in \{1, \ldots, n\}$: the number of boxes of each box type, and
- $R_i, \forall i \in \{1, \ldots, n\}$: the set of allowed orientations for each box type. R_i is always a subset of $\{1, \ldots, 6\}$.

The dimensions of a box with original dimensions (l, w, h), along the x-axis, y-axis, and z-axis, respectively, are as follows:

- orientation 1: (l, w, h),
- orientation 2: (l, h, w),
- orientation 3: (w, l, h),
- orientation 4: (w, h, l),
- orientation 5: (h, l, w),
- orientation 6: (h, w, l).

This section presents a constraint programming model, adapted with modifications from the constraint programming model introduced by Martin et al. (2024) for the 3D open dimension problem. We then enhance our model by incorporating the concept of block-building, a technique used in many heuristic methods for the CLP (Zhu et al., 2012). We refer to the initial model as the constraint programming (CP) model and the enhanced version as the block-building constraint programming (BBCP) model. Finally, we show how to implement partial or full support constraints in the CP model, as well as full support constraints in the BBCP model.

2.1. CP model

We define $I = \{1, \ldots, n\}$ as the set of box types. For each $i \in I$, we define $J_i = \{1, \ldots, d_i\}$ as a set with d_i elements. Each combination (i, j) corresponds to a single box of type i. We use "box (i, j)" to refer to the box corresponding to indices (i, j).

We define the following decision variables:

- s_{ij} is 1 if box (i, j) is in the container, and 0 if it is not,
- (x_{ij}, y_{ij}, z_{ij}) represents the coordinates of the bottom-left-front corner of box (i, j),
- d^x_{ij}, d^y_{ij}, and d^z_{ij} represent the dimensions of box (i, j) after rotation, in the container's x-axis, y-axis, and z-axis, respectively,
- $b^x_{iji'j'}$ is 1 only if box (i, j) is to the left of box (i', j'). Similarly, $b^y_{iji'j'}$ and $b^z_{iji'j'}$ indicate if box (i, j) is in front of, or below, box (i', j'), respectively,
- for $k \in \{1, \ldots, 6\}$, r^k_{ij} is 1 if box (i, j) has orientation k, and 0 otherwise.

For all constraints in the CP model, the domains of the indexes are $i \in I$ and $j \in J_i$. Constraints (5)–(8) and (20) also have $i' \in I$ and $j' \in J_{i'}$. To avoid unnecessary repetition and save space, we write $\forall i, j$ instead of $\forall i \in I, j \in J_i$ and $\forall i, j, i', j'$ instead of $\forall i \in I, j \in J_i, i' \in I, j' \in J_{i'}$.

Journal Pre-proof

The model is presented below:

$$\text{max:} \quad \sum_{i\in I}\sum_{j\in J_i} l_i w_i h_i s_{ij}, \tag{1}$$

$$\text{s.t.:} \quad x_{ij} + d^x_{ij} \le L, \qquad \forall i,j, \tag{2}$$

$$y_{ij} + d^y_{ij} \le W, \qquad \forall i,j, \tag{3}$$

$$z_{ij} + d^z_{ij} \le H, \qquad \forall i,j, \tag{4}$$

$$(b^x_{iji'j'} = 1) \implies (x_{ij} + d^x_{ij} \le x_{i'j'}), \qquad \forall i,j,i',j',(i,j)\neq(i',j'), \tag{5}$$

$$(b^y_{iji'j'} = 1) \implies (y_{ij} + d^y_{ij} \le y_{i'j'}), \qquad \forall i,j,i',j',(i,j)\neq(i',j'), \tag{6}$$

$$(b^z_{iji'j'} = 1) \implies (z_{ij} + d^z_{ij} \le z_{i'j'}), \qquad \forall i,j,i',j',(i,j)\neq(i',j'), \tag{7}$$

$$(s_{ij} = 1 \wedge s_{i'j'} = 1) \implies (b^x_{iji'j'} + b^x_{i'j'ij} + b^y_{iji'j'} + b^y_{i'j'ij} + b^z_{iji'j'} + b^z_{i'j'ij} \ge 1), \qquad \forall i,j,i',j', i<i' \vee (i=i' \wedge j<j'), \tag{8}$$

$$(r^1_{ij} = 1) \implies (d^x_{ij} = l_i \wedge d^y_{ij} = w_i \wedge d^z_{ij} = h_i), \qquad \forall i,j, \tag{9}$$

$$(r^2_{ij} = 1) \implies (d^x_{ij} = l_i \wedge d^y_{ij} = h_i \wedge d^z_{ij} = w_i), \qquad \forall i,j, \tag{10}$$

$$(r^3_{ij} = 1) \implies (d^x_{ij} = w_i \wedge d^y_{ij} = l_i \wedge d^z_{ij} = h_i), \qquad \forall i,j, \tag{11}$$

$$(r^4_{ij} = 1) \implies (d^x_{ij} = w_i \wedge d^y_{ij} = h_i \wedge d^z_{ij} = l_i), \qquad \forall i,j, \tag{12}$$

$$(r^5_{ij} = 1) \implies (d^x_{ij} = h_i \wedge d^y_{ij} = l_i \wedge d^z_{ij} = w_i), \qquad \forall i,j, \tag{13}$$

$$(r^6_{ij} = 1) \implies (d^x_{ij} = h_i \wedge d^y_{ij} = w_i \wedge d^z_{ij} = l_i), \qquad \forall i,j, \tag{14}$$

$$r^1_{ij} + r^2_{ij} + r^3_{ij} + r^4_{ij} + r^5_{ij} + r^6_{ij} = 1, \qquad \forall i,j, \tag{15}$$

$$(k \notin R_i) \implies (r^k_{ij} = 0), \qquad \forall i,j,k \in \{1,2,3,4,5,6\}, \tag{16}$$

$$s_{ij} \in \{0,1\}, \qquad \forall i,j, \tag{17}$$

$$x_{ij}, y_{ij}, z_{ij} \in \mathbb{Z}_{\ge 0}, \qquad \forall i,j, \tag{18}$$

$$d^x_{ij}, d^y_{ij}, d^z_{ij} \in \{l_i, w_i, h_i\}, \qquad \forall i,j, \tag{19}$$

$$b^x_{iji'j'}, b^y_{iji'j'}, b^z_{iji'j'} \in \{0,1\}, \qquad \forall i,j,i',j',(i,j)\neq(i',j'), \tag{20}$$

$$r^1_{ij}, r^2_{ij}, r^3_{ij}, r^4_{ij}, r^5_{ij}, r^6_{ij} \in \{0,1\}, \qquad \forall i,j. \tag{21}$$

The objective function (1) is the total volume of the boxes placed in the container. The inequalities (2)–(4) ensure that every box is entirely within the container. Conditions (5)–(7) define the meanings of the binary variables $b^x_{iji'j'}$, $b^y_{iji'j'}$, and $b^z_{iji'j'}$. If two distinct boxes are in the container, constraints (8) guarantee they do not overlap. Specifically, one of the boxes must be to the left of, in front of, or below the other box. The non-overlapping condition must be checked once for each pair of boxes, so we enforce it only if (i,j) is smaller than (i',j') under lexicographic ordering.

Constraints (9)–(14) enforce the interpretations of the binary variables r^k_{ij} for $k \in \{1,2,3,4,5,6\}$ by linking the values of d^x_{ij}, d^y_{ij}, and d^z_{ij} with the dimensions of the rotated box. Note that $(d^x_{ij}, d^y_{ij}, d^z_{ij})$ is always

Journal Pre-proof

a permutation of (l_i, w_i, h_i), with each permutation corresponding to an orientation. Constraints (15) ensure that every box is in exactly one of the six orientations. Finally, recall that some box types may be only allowed in specific orientations, given by the sets R_i. Constraints (16) ensure that boxes always have valid orientations.

Compared to the constraint programming model by Martin et al. (2024), this model has two key differences. Firstly, it is a formulation for the CLP, not the 3D open dimension problem, which changes our objective function. Boxes are not guaranteed to be in the container, necessitating the s_{ij} variables to indicate whether each box is used. Meanwhile, the container's side lengths are fixed, so L, W, and H are no longer integer variables.

Additionally, we present our model using the i and j indices to emphasize that some boxes are identical, even though this model does not utilize this beyond checking orientation constraints. In contrast, the model by Martin et al. (2024) uses a single indexing set and variable for all boxes. This is a presentation choice for now but makes extending our model to the BBCP model presented in the next subsection easier.

2.2. *BBCP model*

In the BBCP model, we allow the formation of *blocks* made up of multiple boxes. Each block consists of only one box type, with all boxes placed in the same orientation and arranged in a grid formation to form a larger rectangular cuboid. In many block-building heuristics, these blocks are called *simple blocks*, as opposed to *general blocks*, which may consist of multiple types of boxes or boxes in different orientations (Zhu et al., 2012). By using simple blocks instead of individual boxes, the BBCP model can place many boxes simultaneously without gaps between them. Additionally, to ensure that boxes do not overlap, the model only needs to check that the blocks do not overlap. This approach can significantly reduce the number of overlap constraints that need to be enforced.

We again define $I = \{1, \ldots, n\}$ as the set of box types, and for each $i \in I$, we define the set $J_i = \{1, \ldots, d_i\}$. However, instead of referring to individual boxes, each combination (i, j) now corresponds to a block consisting of boxes of type i. By initializing d_i blocks of type i, we ensure that all possibilities for placing boxes in the container are covered, including placing each box individually. We use "block (i, j)" to refer to the block of boxes of type i corresponding to indices (i, j). Note that we do not enumerate every possibility for blocks consisting of boxes of type i; instead, for each block, we define integer variables that record the number of boxes in each direction and the total number of boxes.

The following decision variables from the previous model are reused:

- (x_{ij}, y_{ij}, z_{ij}) represents the coordinates of the bottom-left-front corner of block (i, j),
- d^x_{ij}, d^y_{ij}, and d^z_{ij} represent the dimensions of a box in block (i, j) after rotation, in the container's x-axis, y-axis, and z-axis, respectively,
- $b^x_{iji'j'}$ is 1 only if block (i, j) is to the left of block (i', j'). Similarly, $b^y_{iji'j'}$ and $b^z_{iji'j'}$ indicate whether block (i, j) is in front of, or below, block (i', j'), respectively,
- for $k \in \{1, \ldots, 6\}$, r^k_{ij} is 1 if boxes in block (i, j) have orientation k, and 0 otherwise.

We also define the following new variables:

- For each $i \in I$ and $j \in J_i$, we no longer have the binary variables s_{ij} indicating whether boxes are loaded. Instead, we define the integer variables: n^x_{ij}, n^y_{ij}, n^z_{ij}, and n_{ij}. If $n_{ij} = 0$, then block (i, j) contains no boxes. Otherwise, block (i, j) corresponds to a block in the container with n^x_{ij} boxes in the container's x-direction, n^y_{ij} boxes in its y-direction, and n^z_{ij} boxes in its z-direction, forming a block with n_{ij} total boxes of type i,
- l^x_{ij}, l^y_{ij}, and l^z_{ij} represent the dimensions of block (i, j), in the container's x-axis, y-axis, and z-axis, respectively.

Figure 1 provides a visual explanation of the variables that make up a block in the BBCP model within a two-dimensional context. The figure shows a block containing $n^x_{ij} = 8$ boxes in the x direction and $n^y_{ij} = 6$ boxes in the y direction, totaling $n_{ij} = n^x_{ij} \cdot n^y_{ij} = 48$ boxes of type i. Each box has dimensions $d^x_{ij} \times d^y_{ij}$ and the size of the block is $l^x_{ij} \times l^y_{ij}$.

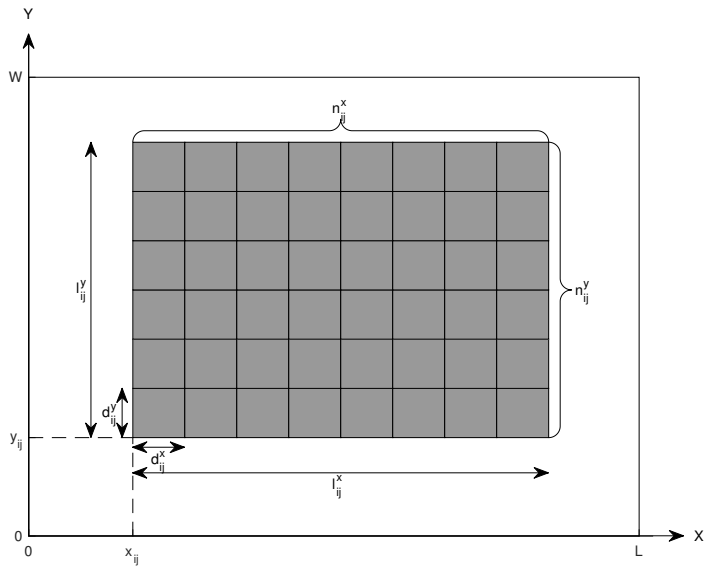


Figure 1: Variable definitions for blocks in the BBCP model

As with the CP model, for all constraints in the BBCP model, the domains of the indexes are $i \in I$, $j \in J_i$, $i' \in I$, and $j' \in J_{i'}$. Again, we omit the domains while presenting the constraints. The model is presented below:

$$\text{max:} \quad \sum\nolimits_{i \in I} (l_i w_i h_i \sum\nolimits_{j \in J_i} n_{ij}), \tag{22}$$

$$\text{s.t.:} \quad n_{ij} = n^x_{ij} n^y_{ij} n^z_{ij}, \qquad \forall i, j, \tag{23}$$

$$\sum\nolimits_{j \in J_i} n_{ij} \leq d_i, \qquad \forall i, \tag{24}$$

Journal Pre-proof

$$l^x_{ij} = d^x_{ij} n^x_{ij}, \qquad \forall i, j, \tag{25}$$
$$l^y_{ij} = d^y_{ij} n^y_{ij}, \qquad \forall i, j, \tag{26}$$
$$l^z_{ij} = d^z_{ij} n^z_{ij}, \qquad \forall i, j, \tag{27}$$
$$x_{ij} + l^x_{ij} \le L, \qquad \forall i, j, \tag{28}$$
$$y_{ij} + l^y_{ij} \le W, \qquad \forall i, j, \tag{29}$$
$$z_{ij} + l^z_{ij} \le H, \qquad \forall i, j, \tag{30}$$
$$(b^x_{iji'j'} = 1) \implies (x_{ij} + l^x_{ij} \le x_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{31}$$
$$(b^y_{iji'j'} = 1) \implies (y_{ij} + l^y_{ij} \le y_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{32}$$
$$(b^z_{iji'j'} = 1) \implies (z_{ij} + l^z_{ij} \le z_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{33}$$
$$b^x_{iji'j'} + b^x_{i'j'ij} + b^y_{iji'j'} + b^y_{i'j'ij} + b^z_{iji'j'} + b^z_{i'j'ij} \ge 1, \qquad \forall i, j, i', j', i < i' \vee (i = i' \wedge j < j'), \tag{34}$$
$$(r^1_{ij} = 1) \implies (d^x_{ij} = l_i \wedge d^y_{ij} = w_i \wedge d^z_{ij} = h_i), \qquad \forall i, j, \tag{35}$$
$$(r^2_{ij} = 1) \implies (d^x_{ij} = l_i \wedge d^y_{ij} = h_i \wedge d^z_{ij} = w_i), \qquad \forall i, j, \tag{36}$$
$$(r^3_{ij} = 1) \implies (d^x_{ij} = w_i \wedge d^y_{ij} = l_i \wedge d^z_{ij} = h_i), \qquad \forall i, j, \tag{37}$$
$$(r^4_{ij} = 1) \implies (d^x_{ij} = w_i \wedge d^y_{ij} = h_i \wedge d^z_{ij} = l_i), \qquad \forall i, j, \tag{38}$$
$$(r^5_{ij} = 1) \implies (d^x_{ij} = h_i \wedge d^y_{ij} = l_i \wedge d^z_{ij} = w_i), \qquad \forall i, j, \tag{39}$$
$$(r^6_{ij} = 1) \implies (d^x_{ij} = h_i \wedge d^y_{ij} = w_i \wedge d^z_{ij} = l_i), \qquad \forall i, j, \tag{40}$$
$$r^1_{ij} + r^2_{ij} + r^3_{ij} + r^4_{ij} + r^5_{ij} + r^6_{ij} = 1, \qquad \forall i, j, \tag{41}$$
$$(k \notin R_i) \implies (r^k_{ij} = 0), \qquad \forall i, j, k \in \{1, 2, 3, 4, 5, 6\}, \tag{42}$$
$$x_{ij}, y_{ij}, z_{ij}, n^x_{ij}, n^y_{ij}, n^z_{ij}, n_{ij}, l^x_{ij}, l^y_{ij}, l^z_{ij} \in \mathbb{Z}_{\ge 0}, \qquad \forall i, j, \tag{43}$$
$$d^x_{ij}, d^y_{ij}, d^z_{ij} \in \{l_i, w_i, h_i\}, \qquad \forall i, j, \tag{44}$$
$$b^x_{iji'j'}, b^y_{iji'j'}, b^z_{iji'j'} \in \{0, 1\}, \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{45}$$
$$r^1_{ij}, r^2_{ij}, r^3_{ij}, r^4_{ij}, r^5_{ij}, r^6_{ij} \in \{0, 1\}, \qquad \forall i, j. \tag{46}$$

The objective function (22) is the total volume of the boxes placed in the container. Constraints (23) count the number of boxes in each block. For each i, constraint (24) ensures that the number of boxes of type i in the container does not exceed the total number available. Constraints (25)–(27) measure the block's length, width, and height, respectively. Inequalities (28)–(30), similar to inequalities (2)–(4) in the CP model, ensure that every block is entirely within the container. Recall that $l^x_{ij} \times l^y_{ij} \times l^z_{ij}$ represents the size of a block, just as $d^x_{ij} \times d^y_{ij} \times d^z_{ij}$ represents the size of a box in the CP model.

Constraints (31)–(33) are equivalent to constraints (5)–(7) in the CP model and ensure the correct definition of binary variables $b^x_{iji'j'}, b^y_{iji'j'}, b^z_{iji'j'}$. Constraints (34) ensure that distinct blocks do not overlap. We can enforce this constraint even for "empty" blocks as a block with no boxes occupies no space

Journal Pre-proof

($l^x_{ij} = l^y_{ij} = l^z_{ij} = 0$), so it will not overlap with other blocks unless its position variables are inside another block. The remaining constraints, which address rotation, are identical to those in the CP model.

We also add the following valid inequalities and constraints to the BBCP model:

$$n^x_{ij} \le d_i, \qquad \forall i, j, \tag{47}$$
$$n^y_{ij} \le d_i, \qquad \forall i, j, \tag{48}$$
$$n^z_{ij} \le d_i, \qquad \forall i, j, \tag{49}$$
$$n_{ij} \ge n^x_{ij}, \qquad \forall i, j, \tag{50}$$
$$n_{ij} \ge n^y_{ij}, \qquad \forall i, j, \tag{51}$$
$$n_{ij} \ge n^z_{ij}, \qquad \forall i, j. \tag{52}$$

The upper bounds (47)–(49) are imposed when defining the integer variables, though they are not strictly needed because constraints (23) already limit the number of boxes of each type. Constraints (50)–(52) guarantee that if $n_{ij} = 0$, $n^x_{ij} = n^y_{ij} = n^z_{ij} = 0$.

For box types with side lengths equal to each other, we include some additional restrictions on the orientation variables to enhance the CP and BBCP models:

$$r^3_{ij} = r^4_{ij} = r^6_{ij} = 0, \qquad \forall i, j, l_i = w_i, \tag{53}$$
$$r^4_{ij} = r^5_{ij} = r^6_{ij} = 0, \qquad \forall i, j, l_i = h_i, \tag{54}$$
$$r^2_{ij} = r^5_{ij} = r^6_{ij} = 0, \qquad \forall i, j, w_i = h_i. \tag{55}$$

Constraints (53)–(55) are adapted from similar constraints used to enhance the model presented by Martin et al. (2024). They ensure that unnecessary orientations are not considered. For example, if $l_i = w_i$, orientation 3 is equivalent to orientation 1, orientation 4 is equivalent to orientation 2, and orientation 6 is equivalent to orientation 5, so there is no reason to consider orientations 3, 4, or 6. If all the side lengths of the box are equal, rotating the box has no effect so we only consider orientation 1.

We could also impose additional symmetry-breaking constraints. For example, in the CP model, we could require that $s_{ij} \ge s_{i,j+1}, \forall i \in I, j \in J_i, j < d_i$. This would ensure that, for each box type i, higher-index boxes can be used only if lower-index boxes are also used. However, preliminary experiments showed that these symmetry-breaking constraints did not improve our results, so we decided to omit them from the models.

2.3. Support constraints

In this subsection, we describe how support constraints can be enforced for our constraint programming models. Full support requires every box to be fully supported by the container floor or the tops of other boxes. Partial support requires some fraction α of the bottom area to be supported, where the support factor α is between 0 and 1. $\alpha = 1$ is the full support case and $\alpha = 0$ is the unsupported case. To enforce support

Journal Pre-proof

constraints for the CP model, we define new decision variables with the following interpretations:

- a_{ij} is the area of the bottom face of box (i, j),
- b^0_{ij} is 1 only if box (i, j) is on the container floor,
- $c_{iji'j'}$ is 1 only if the top of box (i', j') is at the same height as the bottom of box (i, j),
- $x^{\Delta}_{iji'j'}$ is the overlap between boxes (i, j) and (i', j') in the x-direction, and $y^{\Delta}_{iji'j'}$ is the overlap between boxes (i, j) and (i', j') in the y-direction,
- $a^{\Delta}_{iji'j'}$ is the amount of area that box (i', j') supports box (i, j). It is zero if box (i', j') does not support box (i, j),
- The $x^{+}_{iji'j'}$, $x^{-}_{iji'j'}$, $y^{+}_{iji'j'}$, and $y^{-}_{iji'j'}$ integer variables are used to ensure $x^{\Delta}_{iji'j'}$ and $y^{\Delta}_{iji'j'}$ take the correct values, while the $c^{\Delta}_{iji'j'}$ variables are used to ensure $a^{\Delta}_{iji'j'}$ takes the correct values.

To enforce these support constraints, the following constraints must be added to the CP model:

$$a_{ij} = d^x_{ij} d^y_{ij}, \qquad \forall i, j, \tag{56}$$

$$(b^0_{ij} = 1) \implies (z_{ij} = 0), \qquad \forall i, j, \tag{57}$$

$$(b^0_{ij} = 0) \implies \Big(\sum_{i' \in I} \sum_{\substack{j' \in J_{i'} \\ (i',j') \neq (i,j)}} a^{\Delta}_{iji'j'} \geq \alpha a_{ij}\Big), \qquad \forall i, j, \tag{58}$$

$$(c_{iji'j'} = 1) \implies (z_{ij} = z_{i'j'} + d^z_{i'j'} \wedge s_{i'j'} = 1 \wedge a^{\Delta}_{iji'j'} = c^{\Delta}_{iji'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{59}$$

$$(c_{iji'j'} = 0) \implies (a^{\Delta}_{iji'j'} = 0), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{60}$$

$$c^{\Delta}_{iji'j'} = x^{\Delta}_{iji'j'} y^{\Delta}_{iji'j'}, \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{61}$$

$$x^{+}_{iji'j'} = \min(x_{ij} + d^x_{ij}, x_{i'j'} + d^x_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{62}$$

$$x^{-}_{iji'j'} = \max(x_{ij}, x_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{63}$$

$$x^{\Delta}_{iji'j'} = \max(0, x^{+}_{iji'j'} - x^{-}_{iji'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{64}$$

$$y^{+}_{iji'j'} = \min(y_{ij} + d^y_{ij}, y_{i'j'} + d^y_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{65}$$

$$y^{-}_{iji'j'} = \max(y_{ij}, y_{i'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{66}$$

$$y^{\Delta}_{iji'j'} = \max(0, y^{+}_{iji'j'} - y^{-}_{iji'j'}), \qquad \forall i, j, i', j', (i, j) \neq (i', j'), \tag{67}$$

$$a_{ij} \in \mathbb{Z}_{\geq 0}, \qquad \forall i, j, \tag{68}$$

$$b^0_{ij} \in \{0, 1\}, \qquad \forall i, j, \tag{69}$$

$$c_{iji'j'} \in \{0, 1\}, \qquad \forall i, j, i', j', \tag{70}$$

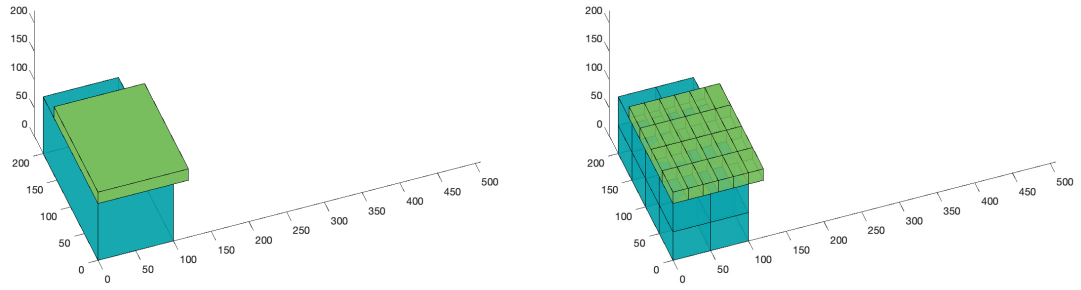
$$x^{+}_{iji'j'}, x^{-}_{iji'j'}, x^{\Delta}_{iji'j'}, y^{+}_{iji'j'}, y^{-}_{iji'j'}, y^{\Delta}_{iji'j'}, a^{\Delta}_{iji'j'}, c^{\Delta}_{iji'j'} \in \mathbb{Z}_{\geq 0}, \qquad \forall i, j, i', j', (i, j) \neq (i', j'). \tag{71}$$

Constraints (56) calculate the area of the bottom face of each box. Constraints (57) and (58) ensure that box (i, j) is either on the floor of the container (if $b^0_{ij} = 1$) or has at least αa_{ij} of its bottom area supported by other boxes (if $b^0_{ij} = 0$). Implications (59) states that if box (i', j') supports box (i, j), the top of box (i', j') must have the same height as the bottom of box (i, j), and box (i', j') must be placed in the container. In

that case, constraints (59) combined with equalities (61) ensure that $a^{\Delta}_{iji'j'} = x^{\Delta}_{iji'j'} y^{\Delta}_{iji'j'}$, meaning the area supported by box (i', j') is the product of the overlaps in the x and y directions. Otherwise, box (i', j') does not support box (i, j), as enforced by constraints (60). Finally, constraints (62)–(64) together ensure that $x^{\Delta}_{iji'j'}$ is equal to the amount of overlap of the boxes in the x-direction, which is the overlap of the intervals $[x_{ij}, x_{ij} + d^x_{ij}]$ and $[x_{i'j'}, x_{i'j'} + d^x_{i'j'}]$. Constraints (65)–(67) are identical for the y-direction.

Enforcing support constraints adds a large number of new integer variables to the model, making it much bigger and more complex than the model for the unsupported variant.

In the BBCP model, full support can be enforced by treating blocks as single units. Only boxes in the bottom layer of each block need to be supported, as higher layers are naturally supported by the ones below. This approach applies only to full support because partial support of a block does not guarantee sufficient support for each box within the block. Figure 2 illustrates this distinction. For example, when $\alpha = 0.5$, the arrangement in Figure 2a satisfies the stability constraints, but the arrangement in Figure 2b does not, since the right-most boxes in the top block are unsupported.



(a) Arrangement with boxes satisfies stability constraints

(b) Arrangement with blocks does not satisfy stability constraints

Figure 2: Illustrative examples for the partial support variant ($\alpha = 0.5$)

We re-use the decision variables, but their interpretations now refer to blocks instead of boxes. Specifically, the following variables have altered interpretations:

- a_{ij} is the area of the bottom face of block (i, j),
- b^0_{ij} is 1 only if block (i, j) is on the container floor,
- $c_{iji'j'}$ is 1 only if the top of block (i', j') is at the same height as the bottom of block (i, j),
- $x^{\Delta}_{iji'j'}$ is the overlap between blocks (i, j) and (i', j') in the x-direction, and $y^{\Delta}_{iji'j'}$ is the overlap between blocks (i, j) and (i', j') in the y-direction,
- $a^{\Delta}_{iji'j'}$ is the amount of area that block (i', j') supports block (i, j). It is zero if block (i', j') does not support block (i, j).

Journal Pre-proof

The variables and constraints needed to enforce full stability for the BBCP model are presented below:

$$a_{ij} = l^x_{ij} l^y_{ij}, \qquad \forall i, j, \tag{72}$$

$$(b^0_{ij} = 1) \implies (z_{ij} = 0), \qquad \forall i, j, \tag{73}$$

$$(b^0_{ij} = 0) \implies \Big(\sum_{i' \in I} \sum_{\substack{j' \in J_{i'} \\ (i',j') \neq (i,j)}} a^\Delta_{iji'j'} \geq a_{ij}\Big), \qquad \forall i, j, \tag{74}$$

$$(c_{iji'j'} = 1) \implies (z_{ij} = z_{i'j'} + l^z_{i'j'} \wedge a^\Delta_{iji'j'} = c^\Delta_{iji'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{75}$$

$$(c_{iji'j'} = 0) \implies (a^\Delta_{iji'j'} = 0), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{76}$$

$$c^\Delta_{iji'j'} = x^\Delta_{iji'j'} y^\Delta_{iji'j'}, \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{77}$$

$$x^+_{iji'j'} = \min(x_{ij} + l^x_{ij}, x_{i'j'} + l^x_{i'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{78}$$

$$x^-_{iji'j'} = \max(x_{ij}, x_{i'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{79}$$

$$x^\Delta_{iji'j'} = \max(0, x^+_{iji'j'} - x^-_{iji'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{80}$$

$$y^+_{iji'j'} = \min(y_{ij} + l^y_{ij}, y_{i'j'} + l^y_{i'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{81}$$

$$y^-_{iji'j'} = \max(y_{ij}, y_{i'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{82}$$

$$y^\Delta_{iji'j'} = \max(0, y^+_{iji'j'} - y^-_{iji'j'}), \qquad \forall i, j, i', j', (i,j) \neq (i',j'), \tag{83}$$

$$a_{ij} \in \mathbb{Z}_{\geq 0}, \qquad \forall i, j, \tag{84}$$

$$b^0_{ij} \in \{0, 1\}, \qquad \forall i, j, \tag{85}$$

$$c_{iji'j'} \in \{0, 1\}, \qquad \forall i, j, i', j', \tag{86}$$

$$x^+_{iji'j'}, x^-_{iji'j'}, x^\Delta_{iji'j'}, y^+_{iji'j'}, y^-_{iji'j'}, y^\Delta_{iji'j'}, a^\Delta_{iji'j'}, c^\Delta_{iji'j'} \in \mathbb{Z}_{\geq 0}, \qquad \forall i, j, i', j', (i,j) \neq (i',j'). \tag{87}$$

Compared to the constraints for the CP model, we have replaced all instances of d^x_{ij}, d^y_{ij}, and d^z_{ij} with l^x_{ij}, l^y_{ij}, and l^z_{ij}, respectively, to indicate the dimensions of the blocks. Constraints (74) have been additionally modified by replacing αa_{ij} with a_{ij}. Unlike the CP model, which can accommodate both partial and full support constraints, we can only enforce full support for the BBCP model. Constraints (75) have also been modified from constraints (59) to no longer require that block (i', j') be used, since if block (i', j') is empty, it will support zero area anyway. Aside from these modifications, the constraints are identical to those for the CP model.

3. Computational results

To assess the performance of the models, we test them on the "BR" and "LN" instances from the literature. The BR instances consist of 1600 test cases organized into 16 classes labeled from BR0 to BR15, with each class containing 100 instances. The BR1-BR7 instances were generated by Bischoff and Ratcliff (1995), while BR0 and BR8-BR15 were generated by Davies and Bischoff (1999). The number of box types increases progressively across the classes. BR0 consists of instances with identical boxes, which are

Journal Pre-proof

homogeneous instances. Instances from BR1 to BR7 have between 3 and 20 box types, and those from BR8 to BR15 instances have between 30 and 100 box types. Most publications classify BR1 to BR7 as *weakly heterogeneous* and BR8 to BR15 as *strongly heterogeneous*.

The BR instances are widely used in the literature to evaluate algorithm performance, particularly for heuristic methods. According to Silva et al. (2019), these instances, which contain between 41 and 1169 boxes, are extremely challenging due to their size, and none of the exact methods considered in the comparative study can solve any of them to optimality. Instead, the paper reports the average container occupation for the best solution found within a given time limit. The constraint programming models presented in this paper are similarly unable to obtain a certificate of performance for any of the BR instances, so we continue the practice of Silva et al. (2019) and report the best solution (lower bound) obtained by the solver for each model.

In contrast to the BR instances, the LN instances are a smaller set of 15 classic benchmark cases proposed by Loh and Nee (1992). These instances are classified as weakly heterogeneous, with between 6 and 8 box types. Unlike the BR instances, most LN instances allow for every box to be placed in the container, enabling exact methods to attain and prove optimal solutions.

Our constraint programming models are implemented in Python and solved using the Google OR-Tools CP-SAT Solver (v9.10.4067). The OR-Tools CP-SAT Solver is a state-of-the-art constraint programming solver. In recent years, it has consistently demonstrated excellent performance in the MiniZinc Challenge (Stuckey et al., 2014), where it is compared against other solvers on a variety of benchmarks. We performed some preliminary tests with another state-of-the-art solver, the IBM ILOG CP Optimizer solver, but we found that OR-Tools CP-SAT performed significantly better for both constraint programming models.

The constraint programming models are solved with a time limit of 3600 seconds, 64 GB of RAM, and 4 workers, with all other solver settings set to their defaults. These settings are chosen to align with those used in the comparative study, Silva et al. (2019).

The CP and BBCP models are compared with the two best-performing existing exact methods for the BR instances as reported by Silva et al. (2019): the MIP models adapted from Tsai et al. (2014) and Chen et al. (1995). These models are solved using the Gurobi 10.0.1 solver, applying the same time, memory, and worker settings, while using default settings for all other parameters.

Our experiments are conducted on the Spartan HPC system (Meade et al., 2017) with an Intel(R) Xeon(R) Gold 6448H CPU processor.

3.1. BR instances

Table 1 presents results from the CP and BBCP models for the BR instances, highlighting the performance differences between the two. The models are compared with the MIP models adapted from Tsai et al. (2014) and Chen et al. (1995), which are listed under the “Tsai” and “Chen” columns.

For all four models and 1600 BR instances, the upper bound obtained by the solver is equal to the total volume of the items, which is typically around 99.5% of the volume of the container. The procedure by which the BR instances are generated (Bischoff and Ratcliff, 1995) ensures the total volume of the items is

Journal Pre-proof

barely smaller than the volume of the container. Since they are trivial and identical for all models, we do not report upper bounds in Table 1.

We also include a column presenting the results for the BSG-VCS heuristic (Araya et al., 2017), which is the current state-of-the-art among non-exact methods. BSG-VCS constructs solutions using block-building and uses beam search to efficiently explore the solution space and identify high-quality solutions. For consistency, the heuristic was allocated 3600 seconds for each instance.

The table displays each of the 16 classes, followed by the average volume utilization from each solution method for the 100 instances in each class. The final four rows display the average volume occupancy for weakly heterogeneous, strongly heterogeneous, non-homogeneous, and all, instances. Entries in Table 1 are bolded if they represent the best performance for the instance class relative to the other models.

Table 1: BR instances (average % occupation)

Class	Models				Gap	Heuristic
	Tsai	Chen	CP	BBCP	BBCP - CP	BSG-VCS
BR0	76.79	75.33	81.76	**90.40**	8.64	90.47
BR1	75.30	79.33	86.52	**94.35**	7.83	95.91
BR2	73.17	79.74	87.16	**94.46**	7.30	96.72
BR3	71.01	78.10	87.67	**93.71**	6.04	97.05
BR4	70.14	78.06	87.63	**93.59**	5.96	97.08
BR5	69.73	77.34	87.65	**92.87**	5.22	97.02
BR6	69.40	77.01	87.73	**92.64**	4.91	96.99
BR7	68.23	76.70	87.62	**91.56**	3.94	96.82
BR8	67.90	76.81	87.62	**90.42**	2.81	96.60
BR9	67.59	76.99	87.69	**89.52**	1.83	96.37
BR10	67.41	76.45	87.50	**89.07**	1.57	96.18
BR11	67.30	77.02	87.57	**88.91**	1.34	95.96
BR12	67.08	76.38	87.64	**88.63**	0.99	95.83
BR13	66.65	75.99	87.56	**88.42**	0.86	95.74
BR14	66.76	76.49	87.64	**88.03**	0.39	95.66
BR15	66.85	76.42	**87.84**	87.57	-0.28	95.59
Avg(1-7)	71.00	78.04	87.43	**93.31**	5.89	96.80
Avg(8-15)	67.19	76.57	87.63	**88.82**	1.19	95.99
Avg(1-15)	68.97	77.26	87.54	**90.92**	3.38	96.37
Avg(all)	69.46	77.14	87.18	**90.88**	3.67	96.00

The constraint programming models significantly outperform the existing MIP models for the BR instances. Both the CP and BBCP models obtain better average results for all classes of instances. Considering all 16 classes, the CP model achieves an average volume occupancy that is 10.04% higher than the model adapted from Chen et al. (1995) and 17.72% higher than the model adapted from Tsai et al. (2014).

Comparing the CP and BBCP models, Table 1 indicates that the BBCP model is superior for all instance classes except BR15. The performance difference is largest for homogeneous and weakly heterogeneous instances. For example, in BR0 (homogeneous instances) and BR1 (weakly heterogeneous instances with three item types), the performance gaps are 8.64% and 7.83%, respectively. On the other hand, the difference

Journal Pre-proof

is only 0.39% for BR14 instances, and the BBCP model underperforms the CP model by 0.28% for BR15 instances. The CP model is extremely consistent across categories, with only a notable drop for BR0 and, to a lesser extent, BR1-BR2. The BBCP model performs exceptionally well on weakly heterogeneous instances but gradually declines as the number of box types increases.

This behavior can be explained by the properties of the constraint programming formulations. The CP model treats each box individually and does not exploit the fact that some boxes are identical, so weakly and strongly heterogeneous instances are not particularly different. In contrast, the BBCP model's block-building approach is highly effective when large blocks can be formed, allowing it to place many boxes simultaneously with no gaps. Homogeneous and weakly heterogeneous instances have few item types and many boxes per item type, so large blocks can be formed using identical boxes. However, in strongly heterogeneous instances, where there are numerous box types with only a few boxes per type, only small blocks can be formed, diminishing the benefit of forming blocks.

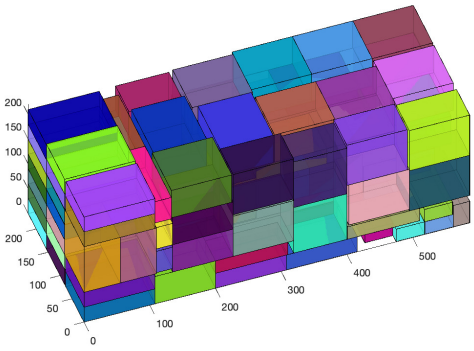
Figure 3 illustrates this effect, showing the solutions obtained with the constraint programming formulations for a typical weakly heterogeneous and strongly heterogeneous instance. The figure presents the solutions obtained for the first instance in the BR1 and BR15 classes. For the CP models, each box is assigned a different color to reflect their individual assignment, while for the BBCP models, each block has a unique color shared by all the boxes in it.

Figure 3b demonstrates that the BBCP model can form large blocks containing many boxes for weakly heterogeneous instances, resulting in tightly packed solutions with high volume occupancy. In this case, the BBCP model achieves a volume occupancy of 93.38%, compared to 87.84% for the CP model. In contrast, Figure 3d illustrates that block-building is less advantageous for strongly heterogeneous instances, where most boxes end in individual blocks due to the limited number of boxes per type. The few blocks that do form contain only a small number of boxes. For this instance, the BBCP model achieves an occupancy of 88.34%, slightly lower than the CP model's 89.42%.

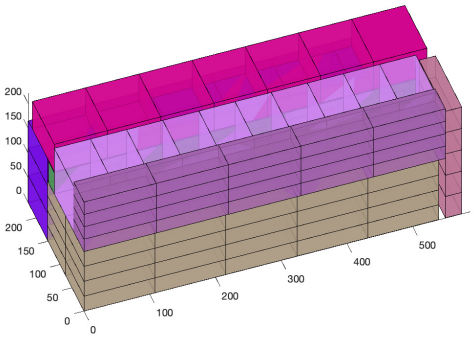
While the CP and BBCP models offer significant improvements over the MIP models, their results remain inferior to those achieved by the state-of-the-art BSG-VCS heuristic, which consistently achieves better volume occupancy across all BR instance classes. However, the BBCP model performs relatively close to the heuristic for homogeneous and very weakly heterogeneous instances, with gaps of only 0.07% for BR0 instances and 1.56% for BR1 instances. Upon reviewing the results for individual instances, we found that BSG-VCS always produces better solutions for instances with multiple box types. However, for the 100 BR0 instances, the BBCP model outperforms the heuristic for 12 instances, underperforms for 22 cases, and they are tied for the remaining 66.

Figure 4 illustrates the performance of each model over time. For each model, and across time limits ranging from 1 to 60 minutes, we compute the average volume occupancy of the best solution found by the solver up to that point for all the BR instances. If no feasible solution has been found, the volume occupancy is recorded as 0. Each line in the graph corresponds to a different model. Figure 5 shows the same graphs for BR1 and BR15 instances.

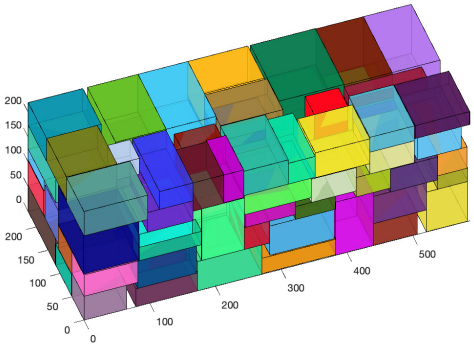
For all BR instances and BR1 instances, the relative rankings of the models remain the same regardless



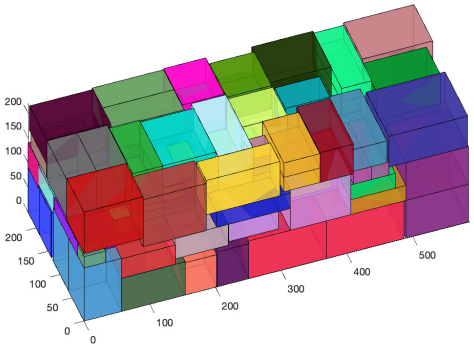
(a) CP, weakly heterogeneous, 87.84% occupancy



(b) BBCP, weakly heterogeneous, 93.38% occupancy



(c) CP, strongly heterogeneous, 89.42% occupancy



(d) BBCP, strongly heterogeneous, 88.34% occupancy

Figure 3: Solutions obtained by the constraint programming models for a weakly heterogeneous instance (BR1-1) and a strongly heterogeneous instance (BR15-1)

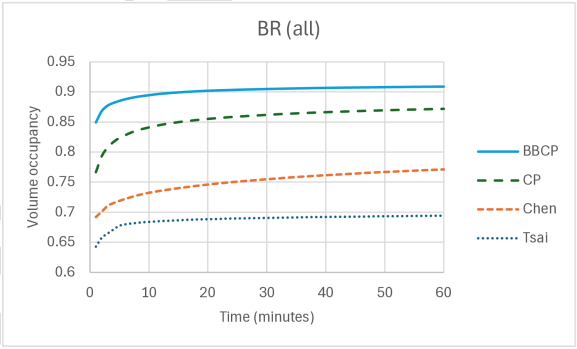


Figure 4: Average performance of models with time for all BR instances

Journal Pre-proof

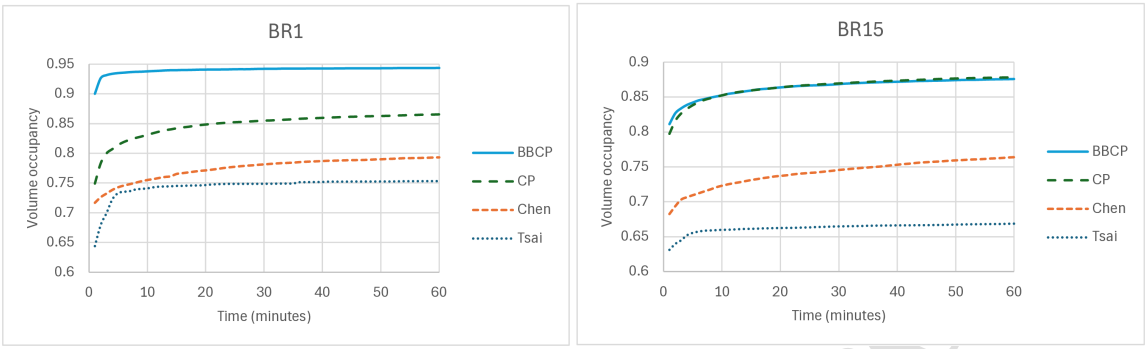


Figure 5: Average performance of models with time for BR1 and BR15 instances

of the time limit. The Tsai and BBCP models usually obtain a solution quickly within 5 minutes and do not significantly improve upon it, while the Chen and CP models demonstrate more gradual and consistent improvement for the entire allotted time. For BR15 instances, the CP and BBCP models are very close in their performances. The BBCP model obtains better solutions quickly but the CP model improves their solutions and outperforms after around 10 minutes, although the average performance difference is tiny throughout.

3.2. LN instances

We also evaluate our models using the LN instances, a collection of 15 classic benchmark problems proposed by Loh and Nee (1992). Entries in Table 2 are bolded if they represent the best results in the row (compared to the other models). For the LN instances, in addition to the volume occupancy, we also record the time the solver takes to obtain a certificate of optimality. If the solver cannot prove optimality within the time limit, we record the time limit, which is 3600 seconds. For instances that are solved to proven optimality by at least one model, we also bold the model that takes the least time. The final row shows the number of instances solved to proven optimality by each model.

The results show that most LN instances are easy, since all models can find optimal solutions for 10 or more instances. The differences for the more challenging instances are consistent with those for the BR instances.

The BBCP model is not always the fastest at obtaining a certificate of optimality. The CP model is fastest for 8 instances (instances 1, 3, 4, 8, 9, 11, 14, and 15), while the BBCP model is fastest for 5 instances (instances 5, 7, 10, 12, and 13). Notably, instance 7 is only solved by the BBCP model within the time limit. No model obtains a certificate of optimality for instances 2 and 6, although the BBCP model finds the best feasible solutions for these.

A closer look at Table 2 reveals a clear pattern. The CP model tends to be fastest when the optimal solution has a relatively low volume occupancy. Of the 8 instances for which the CP model is fastest, the highest volume occupancy is just 62.81%. Meanwhile, the BBCP model performs best when the optimal

Journal Pre-proof

Table 2: LN instances (average % occupation and time in seconds)

Instance	Models							
	Tsai		Chen		CP		BBCP	
	Volume (%)	Time (s)	Volume (%)	Time (s)	Volume (%)	Time (s)	Volume (%)	Time (s)
1	**62.50**	14.54	**62.50**	4.84	**62.50**	**3.59**	**62.50**	4.52
2	77.82	3600.00	86.87	3600.00	87.56	3600.00	**95.92**	3600.00
3	**53.43**	63.21	**53.43**	56.23	**53.43**	**15.42**	**53.43**	55.34
4	**54.96**	5.44	**54.96**	4.76	**54.96**	**3.18**	**54.96**	8.13
5	69.90	3600.00	**77.19**	343.28	**77.19**	21.17	**77.19**	**10.17**
6	76.80	3600.00	83.03	3600.00	85.62	3600.00	**94.61**	3600.00
7	78.74	3600.00	80.50	3600.00	83.59	3600.00	**84.66**	**110.86**
8	**59.42**	13.66	**59.42**	7.92	**59.42**	**6.09**	**59.42**	12.53
9	**61.89**	73.06	**61.89**	76.88	**61.89**	**18.10**	**61.89**	77.36
10	**67.29**	195.92	**67.29**	464.91	**67.29**	373.37	**67.29**	**59.37**
11	**62.16**	8.08	**62.16**	10.86	**62.16**	**2.87**	**62.16**	8.36
12	77.37	3600.00	**78.52**	1098.87	**78.52**	145.18	**78.52**	**16.75**
13	75.58	3600.00	**85.61**	2445.04	**85.61**	3406.95	**85.61**	**38.05**
14	**62.81**	21.21	**62.81**	7.62	**62.81**	**4.86**	**62.81**	10.97
15	**59.46**	1423.09	**59.46**	88.39	**59.46**	**43.69**	**59.46**	94.19
Avg	66.68	1561.21	69.04	1027.31	69.47	989.63	**70.70**	**513.77**
Optimal	10		12		12		**13**	

solution has a higher volume occupancy; of the 5 instances it solves quickest, the lowest volume occupancy is 67.29%. This suggests that the CP model is efficient at quickly finding solutions with volume occupancy below approximately 65%, while the BBCP model excels at finding better solutions when available.

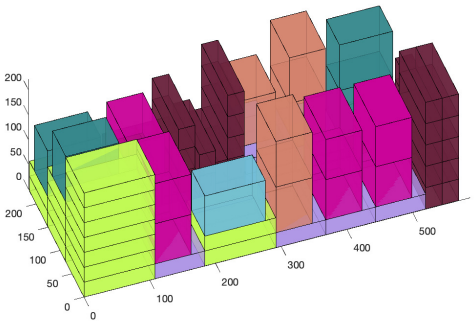
3.3. BR instances with full support

Finally, we evaluate the CP and BBCP models with the full support constraint applied. Due to the increased complexity of these models, we double the number of workers from 4 to 8 so that solutions can be obtained in a reasonable time while keeping all other settings unchanged. The average volume occupancy for each instance class is shown in Table 3. The table also includes a column displaying the results for the BSG-VCS heuristic.

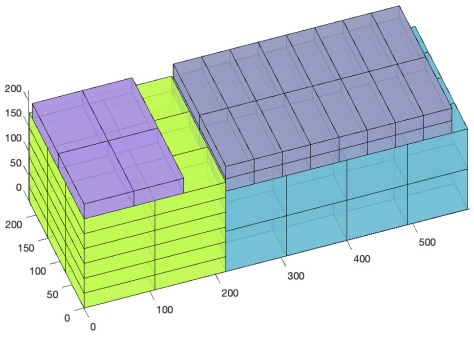
Figure 6 displays the full support solutions obtained for the same instances as in Figure 3. Unlike Figure 3, Figure 6 uses the same color for any identical boxes in the same orientation, even if they are placed separately. This color scheme emphasizes that it is effective to stack identical boxes on each other.

The BBCP model significantly outperforms the CP model across all instance classes. While this is expected for weakly heterogeneous instances, the same trend holds for strongly heterogeneous instances, unlike in the unsupported case. Figure 6c and Figure 6d illustrate the challenges faced by both models when attempting to place numerous boxes of predominantly different sizes. The boxes tend to have smaller face areas as each stack grows taller, ultimately preventing the model from finding sufficiently small boxes to place on top. With so many boxes and so much space left unused, even the small blocks consisting of two or three boxes formed by the BBCP model in Figure 6d result in a 10% increase in volume occupancy.

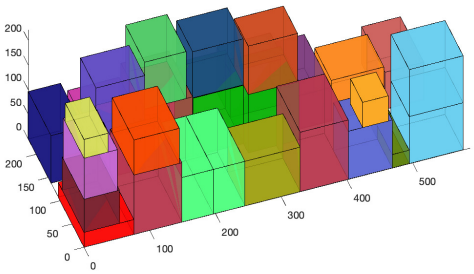
Journal Pre-proof



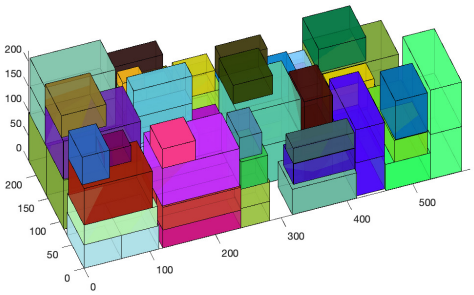
(a) CP, weakly heterogeneous, 54.95% occupancy



(b) BBCP, weakly heterogeneous, 88.90% occupancy



(c) CP, strongly heterogeneous, 44.93% occupancy



(d) BBCP, strongly heterogeneous, 54.93% occupancy

Figure 6: Full support solutions obtained by the constraint programming models for a weakly heterogeneous instance (BR1-1) and a strongly heterogeneous instance (BR15-1)

Journal Pre-proof

Table 3: BR instances with full support (average % occupation)

Class	Models		Gap	Heuristic
	CP	BBCP	BBCP - CP	BSG-VCS
BR0	47.23	**82.01**	34.78	89.96
BR1	47.37	**90.72**	43.35	94.81
BR2	44.12	**90.10**	45.98	95.57
BR3	42.90	**86.23**	43.33	95.91
BR4	40.89	**83.77**	42.87	95.81
BR5	39.87	**80.32**	40.46	95.76
BR6	39.13	**77.71**	38.58	95.61
BR7	38.31	**73.95**	35.64	95.16
BR8	36.73	**66.96**	30.23	94.41
BR9	36.10	**62.40**	26.30	93.87
BR10	34.69	**59.08**	24.39	93.39
BR11	35.76	**58.77**	23.00	92.83
BR12	37.21	**56.81**	19.60	92.34
BR13	36.16	**54.93**	18.77	91.75
BR14	36.24	**52.45**	16.22	91.14
BR15	36.52	**50.10**	13.58	90.47
Avg(1-7)	41.80	**83.26**	41.46	95.52
Avg(8-15)	36.18	**57.69**	21.51	92.53
Avg(1-15)	38.80	**69.62**	30.82	93.92
Avg(all)	39.33	**70.39**	31.07	93.68

This difficulty further explains why both models perform better on homogeneous and weakly heterogeneous instances than strongly heterogeneous ones. Figure 6a demonstrates that even the CP model benefits from having many boxes of each type since it can stack identical boxes. This is despite the model not explicitly using the idea of block-building, unlike the BBCP model which utilizes large blocks to obtain highly efficient full support solutions (Figure 6b).

For the full support variant, it is clear that the heuristic consistently obtains significantly better solutions across all instance classes compared to the models. The difference is most pronounced for strongly heterogeneous instances. While the constraint programming models can find feasible solutions, they are of limited quality and far from optimal.

4. Instance space analysis

Reporting average performances for instance classes does not always provide a complete picture of each algorithm's strengths and weaknesses. The recently developed Instance Space Analysis (ISA) methodology (Smith-Miles and Muñoz, 2023) provides a robust framework for analyzing algorithm performance by projecting problem instances onto a two-dimensional (2D) *instance space*, with each instance represented as a point. This projection is achieved by calculating features of instances that explain algorithm performance and establishing a mapping from these features to the instance space.

Mapping instances to a 2D space enables clearer visualization. By overlaying performance metrics, we

Journal Pre-proof

can identify regions where each algorithm excels or struggles. This visual approach also allows us to scrutinize the diversity and unbiasedness of our problem set. Machine learning techniques can identify regions where each algorithm is expected to perform well. This data can be integrated into an algorithm selector which chooses the most suitable algorithm for any given instance. Ultimately, this methodology provides a deeper insight into the structure of the problem than simply reporting average algorithm performances.

This section outlines the experimental meta-data utilized in our ISA, followed by the instance space and its insights. For full details of the ISA methodology and its impact on algorithms, we refer the reader to the recent paper Smith-Miles and Muñoz (2023), and for recent applications of ISA to optimization and other problems, to Liu et al. (2024).

4.1. Experimental meta-data

To conduct an ISA, four key components are required. First, we need a set of test instances to form our *instance subset*. These instances should exhibit as many of the following characteristics as possible: be demonstrably diverse, lacking bias, sufficiently challenging, discriminating of algorithms' performances, and representative of real-world applications (Smith-Miles and Geng, 2022). Second, we must develop a set of *features* that can be calculated for every problem instance. These features should explain algorithm performance and instance difficulty. Third, we need to choose a set of *algorithms*. Finally, we require a performance metric and a binary definition of *good performance* for each (algorithm, instance) pair. These components together form the *experimental meta-data* necessary for ISA.

For an insightful ISA, it is essential that no single algorithm consistently dominates or is dominated by others across all test instances. Table 1 illustrates that the MIP models are unlikely to be competitive with our constraint programming models. However, the CP model can outperform the BBCP model for some strongly heterogeneous instances. Hence, our analysis will use the two constraint programming models as the selected algorithms. Our performance metric is the volume occupancy; we define a model as good if its solution occupies at least as much volume as the solution from the other model, and bad otherwise. This definition of good performance is relative since each algorithm's binary performance depends on the performance of the other algorithm. In contrast, absolute definitions of good performance compare each algorithm's performance to a fixed threshold.

To expand our set of test instances and increase the number of problems where the CP model can outperform the BBCP model, we generate 300 new instances in addition to the original BR instances, using the same methodology as outlined in Bischoff and Ratcliff (1995). These instances are split into three classes of 100 each, with 110, 120, and 130 different box types. We designate these classes as BR16, BR17, and BR18, representing a natural extension of BR0-BR15.[1]

We calculate 13 features for the CLP, shown in Table 4. Many features are inspired by those used for the 2D Bin Packing Problem (Liu et al., 2024) and the 0-1 Knapsack Problem (Smith-Miles et al., 2021). Specifically, features 1, 4-5, and 7-8 are directly adapted from the work of Liu et al. (2024), while feature

[1] All instances used in this paper are available at `https://github.com/kcliu2/CLP-Datasets`.

Journal Pre-proof

10 is adapted from a knapsack feature with the same name used in Smith-Miles et al. (2021). Features 2-3, 6, 9, and 11-13 are specifically developed for this analysis and not directly adapted from either paper. The Source column in the table states the origin of each feature, listing "CLP" for newly developed features.

Table 4: Table of features

No.	Feature	Source	Description
1	NumBoxes	2BP (Liu et al., 2024)	The number of boxes.
2	NumTypes	CLP	The number of box types.
3	MeanBoxesPerType	CLP	The average number of boxes per box type.
4	MeanVolume	2BP (Liu et al., 2024)	The average of the volumes of the boxes.
5	StdVolume	2BP (Liu et al., 2024)	The standard deviation of the volumes of the boxes.
6	CovVolume	CLP	The coefficient of variation of the volumes of the boxes.
7	MeanLength	2BP (Liu et al., 2024)	The average of all the side lengths of the boxes.
8	StdLength	2BP (Liu et al., 2024)	The standard deviation of all the side lengths of the boxes.
9	CovLength	CLP	The coefficient of variation of all the side lengths of the boxes.
10	CapacityFraction	KP (Smith-Miles et al., 2021)	The volume of the container divided by the total volume of the boxes.
11	AvgOrientsBoxes	CLP	The average number of orientations of the boxes.
12	AvgOrientsTypes	CLP	The average number of orientations of the box types.
13	VolumeRatio	CLP	The ratio of the volumes of the largest and smallest box types.

4.2. The instance space

Once the meta-data is prepared, the ISA methodology is applied to generate the instance space. A detailed explanation of the procedure can be found in Smith-Miles and Muñoz (2023). This paper provides only a brief overview of the key steps.

After some pre-processing of the meta-data, the methodology begins by selecting an optimal subset of features that explain algorithm performance. Next, ISA computes a projection of the features onto the 2D instance space.

This projection, combined with the empirical upper and lower bounds of the selected features, is then used to define the experimental boundary of the instance space.

The features selected and the projection obtained are shown in equation (88).

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.4376 & -0.2759 \\ 0.8119 & 0.9576 \\ 0.9650 & -0.7247 \end{bmatrix}^T \begin{bmatrix} NumTypes \\ MeanBoxesPerType \\ VolumeRatio \end{bmatrix} \tag{88}$$

The Z_1 and Z_2 coordinates are linear combinations of the selected features and position instances within the instance space. The coefficients of these linear combinations are determined by solving an optimization

problem that seeks to create linear trends in both the features and algorithm performance across the space.

The positions of the instances, color-coded by class, are displayed in Figure 7, while Figure 8 illustrates the instances and boundary on the instance space. The distributions of the features are shown in Figure 9.

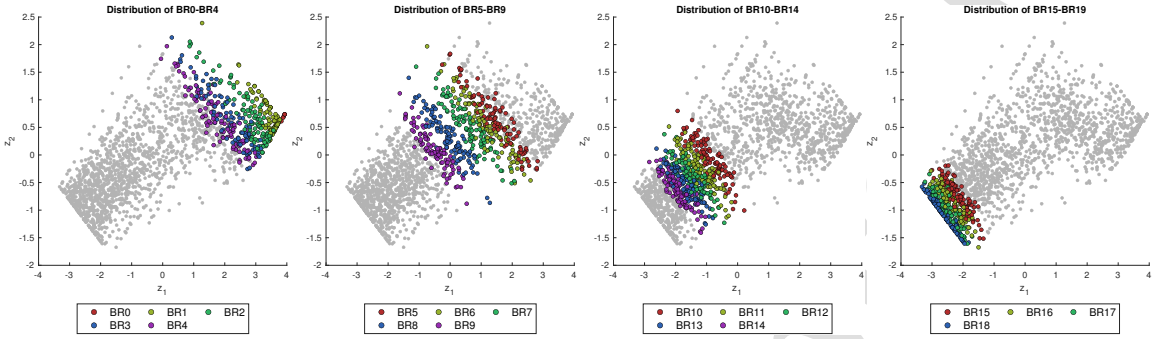


Figure 7: Distribution of instances in the instance space

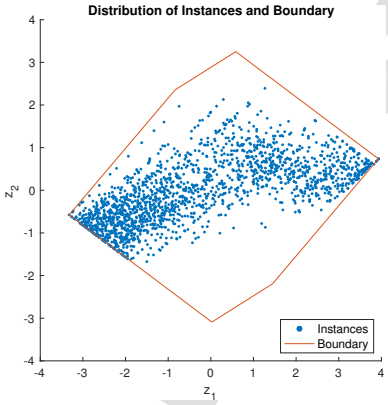


Figure 8: Distribution of instances and boundary in the instance space

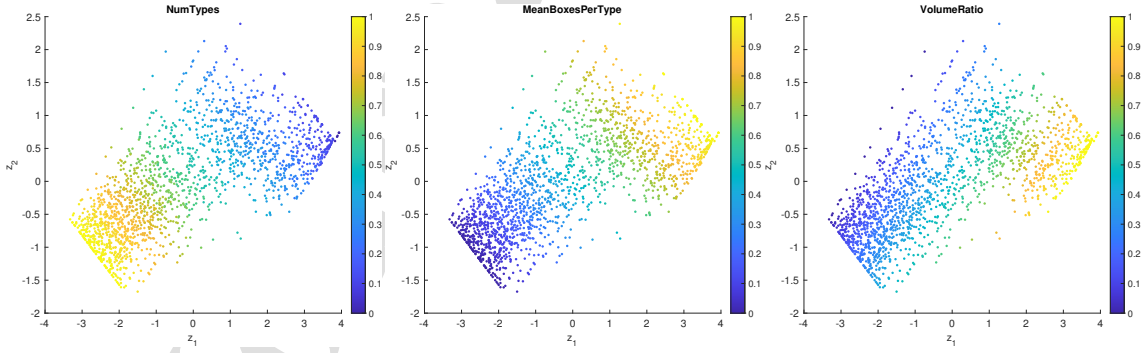


Figure 9: Distribution of features in the instance space

The homogeneous instances (BR0) are clustered at the far right near $(3.9, 0.7)$. Since the NumTypes

and VolumeRatio features are constant for BR0 instances, the instances have only one varying feature and lie close together along a straight line. As the number of box types increases, the instances shift towards the left of the space. From BR0 to BR6-BR7, the classes display greater diversity in feature values and positions in the instance space. Beyond this point, as the instances become more strongly heterogeneous, they move toward the bottom-left corner and cluster again.

Equation (88) and Figure 9 show that the VolumeRatio feature varies diagonally from the top-left to the bottom-right. In contrast, the other two features vary from the bottom left to the top right. Together with Figure 7, the equation and visuals demonstrate that VolumeRatio is the primary feature responsible for separating instances within the same class, while the other two features differentiate instances from different classes.

From Figure 8, we can observe that the instances meet the left and right corners of the boundary and have some presence near the top but do not reach the bottom of the boundary. Our test instances are diverse and fill most of the instance space, but it may be possible to generate new test cases at the bottom of the space near the coordinates $(0, -2)$.

We want to visualize how model performance varies across the instance space. Table 5 presents each model's overall (binary) performance for each BR instance class. The models are tied for 34 instances in BR0, but there are no ties in any other class, so we have omitted a ties column in the table.

Figure 10 displays each model's (binary) performance distribution in the instance space. In the figure, blue dots represent good performance and orange dots represent bad performance. Since binary performance does not show the gap between the two models, we also include Figure 11, which colors all instances by the difference in performance of the two models. Instances are yellow if the BBCP model obtains the better solution and blue if the CP model obtains the better solution. The intensity of the colors reflects the magnitude of the difference in volume occupancies.

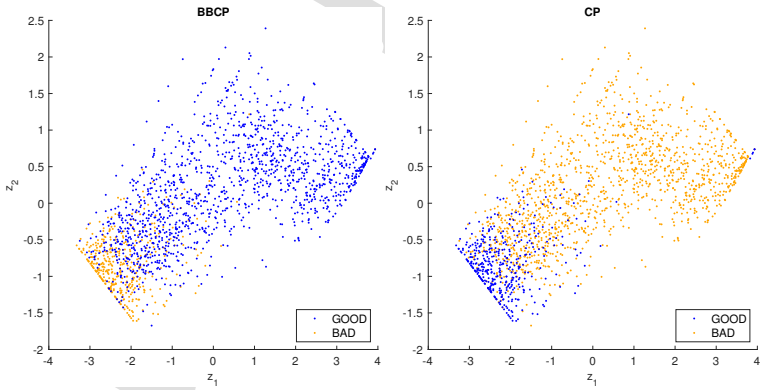


Figure 10: Performance (binary) distribution of models in the instance space

The BBCP model achieves significantly better solutions for nearly all weakly heterogeneous instances. It also outperforms the CP model for most homogeneous instances, although the models are tied for around one-third of them. Since they only have one box type, ties can easily arise as it suffices that both solutions

Journal Pre-proof

Table 5: Actual performance (binary) of models

Class	BBCP better	CP better
BR0	**63**	3
BR1	**100**	0
BR2	**100**	0
BR3	**99**	1
BR4	**100**	0
BR5	**100**	0
BR6	**100**	0
BR7	**100**	0
BR8	**98**	2
BR9	**92**	8
BR10	**89**	11
BR11	**85**	15
BR12	**77**	23
BR13	**78**	22
BR14	**69**	31
BR15	35	**65**
BR16	37	**63**
BR17	13	**87**
BR18	12	**88**
Total	**1447**	419

pack the same number of boxes.

For strongly heterogeneous instances, BBCP is still the better model for BR8-BR9 instances, which include those near the $(0, 0)$ coordinates in the space. However, Figure 11 shows that the gap between the models is generally smaller than for weakly heterogeneous instances. As the number of box types increases, this gap tends to decrease until we reach the BR14-BR15 classes, which include the instances around the coordinates $(-2, -0.5)$. For these instances, the two models are often close in performance, with BBCP outperforming CP in some cases while CP excels in others. The CP model demonstrates a significant advantage for the instance classes BR17 and BR18 in the far bottom-left.

The ISA reaffirms our observations from Section 3.1 regarding the BR0-BR15 instances. Results for the BR16-BR18 classes further illustrate that the CP model outperforms the BBCP model for instances where no significant block-building is possible, confirming that the results for BR15 were not merely an anomaly.

Finally, ISA applies machine learning to train a Support Vector Machine (SVM) for each model. The SVM takes the instance space coordinates as inputs and outputs a binary performance measure, indicating whether the model is expected to perform well or badly. A 5-fold cross-validation experiment is used to evaluate the SVMs.

The SVM predictors are then combined to form an SVM selector, choosing the most trusted model to perform well at each point in the instance space. If both models are predicted to perform well, the selector will choose the SVM with the higher overall precision; in this case, the BBCP model. If neither model is expected to perform well at a given point, the selector will indicate "None," indicating a lack of confidence

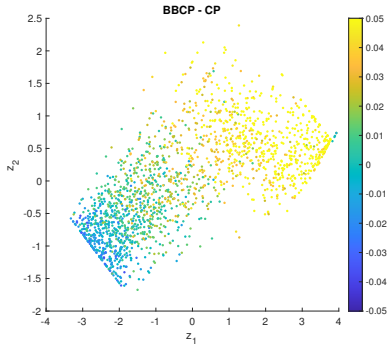


Figure 11: Performance gap between models in the instance space

in either model. Figures 12 and 13 illustrate the SVM performance predictions for both models and the SVM selector, respectively. The SVM table (Table 6) shows the accuracy, precision, and recall of each algorithm's SVM and the SVM selector's precision.

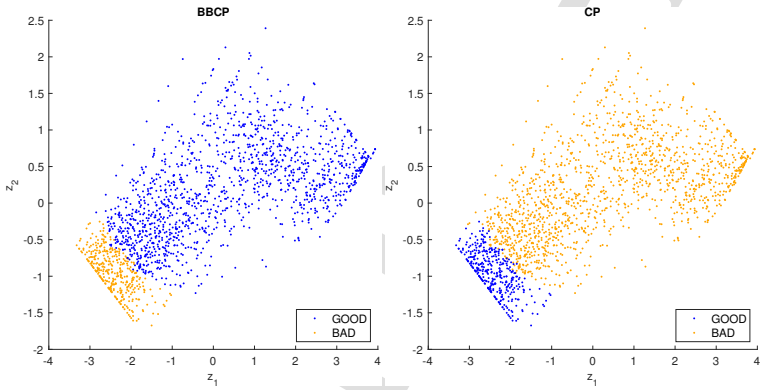


Figure 12: SVM predicted performance of models in the instance space

Table 6: SVM table for the instance space

Algorithm	Accuracy	Precision	Recall
BBCP	88.8%	91.6%	94.3%
CP	87.3%	78.6%	64.0%
SVM Selector	N/A	89.2%	N/A

We aim to compare the performance of the SVM selector to naive approaches that choose one model for all instances. We also consider a rule-based selector that selects the better on-average model for each instance class: the BBCP model for BR0-BR14 instances and the CP model for BR15-BR18. If the SVM selector demonstrates greater accuracy, it would indicate that ISA provides deeper insights beyond merely analyzing average performances across instance classes.

For 1894 instances, the SVM selector chooses a model it trusts to obtain good performance. It returns

Journal Pre-proof

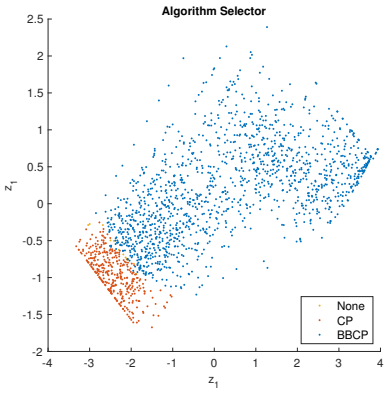


Figure 13: SVM algorithm selector for the instance space

"None" for the other six instances since neither model's SVM predicts good performance. For the "None" instances, we will arbitrarily select BBCP; this choice does not impact the overall accuracy, since BBCP is optimal for three of these instances while CP is optimal for the other three.

Table 7 displays the number of good performances achieved by each model, along with the SVM selector and the rule-based approach. The results show that the SVM selector attains marginally better performance than the rule-based selector, while both significantly outperform the individual models.

Table 7: Comparison of selectors

Algorithm	No. (%) of Good Performances
BBCP always	1481 (77.9%)
CP always	453 (23.8%)
Rule-based Selector	1687 (88.8%)
SVM Selector	1692 (89.1%)

Overall, the ISA reinforces and supports our observations from Section 3. The BBCP model consistently outperforms the CP model for weakly heterogeneous instances located on the right side of the instance space. The BBCP model is also better for some strongly heterogeneous instances in the middle of the space, where the number of boxes of each type remains sufficiently high to enable beneficial block-building. The CP model is better for instances in the bottom-left region, which have many box types but only a single box of most types. This category includes the newly generated BR16-BR18 instances, which extend the BR15 instances by including even more box types.

5. Conclusion

This paper presented two constraint programming formulations for the container loading problem. We first developed a basic constraint programming formulation by adapting a recently introduced model for the open dimension problem. Computational experiments demonstrated that this model can be used to

Journal Pre-proof

achieve significantly better solutions for benchmark instances than the best-performing exact methods in the literature. Then, we extended this model to incorporate the concept of block-building, which allowed the new formulation to place multiple boxes of the same type into the container in a "block" formation with no gaps. The second constraint programming model outperformed the first model across almost all instance classes, demonstrating particular effectiveness for homogeneous and weakly heterogeneous instances with numerous boxes of the same type, enabling the formation of large blocks. Finally, we performed an instance space analysis to investigate further how the characteristics of instances impact the benefit of the block-building approach. The analysis demonstrated that while the block-building model obtains better solutions for most problem instances, it is worse than the basic constraint programming formulation for some strongly heterogeneous instances with many box types and very few boxes of each type.

Our results opened several avenues for further research. While we have implemented support constraints, modifying our models to incorporate other real-world constraints (Nascimento et al., 2021) could further enhance their applicability. Our models could also be used as part of a larger heuristic method. Alternatively, the BBCP model could be directly converted into a heuristic by reducing the number of blocks allowed for each box type. This would remove some feasible solutions from the solution space, but the model would require fewer variables, eventually allowing the solver to obtain good solutions more quickly. Additionally, if possible, formulating the construction of general blocks that accommodate boxes of different types and orientations may prove beneficial. While general blocks can lead to wasted volume, many block-building heuristics find that using general blocks can enhance solution quality for strongly heterogeneous instances.

Acknowledgements

This work was supported by The University of Melbourne, Australia and KU Leuven, Belgium through the KU Leuven - Melbourne Joint PhD program. This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Australia, Project ID IC200100009. This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

References

Ali, S., Ramos, A.G., Carravilla, M.A., Oliveira, J.F., 2022. On-line three-dimensional packing problems: A review of off-line and on-line solution approaches. Computers & Industrial Engineering 168, 108122. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835222001929, doi:10.1016/j.cie.2022.108122.

Araya, I., Guerrero, K., Nuñez, E., 2017. VCS: A new heuristic function for selecting boxes in the single container loading problem. Computers & Operations Research 82, 27–35. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054817300023, doi:10.1016/j.cor.2017.01.002.

Bayraktar, T., Ersöz, F., Kubat, C., 2021. Effects of memory and genetic operators on Artificial Bee Colony algorithm for Single Container Loading problem. Applied Soft Computing 108, 107462. URL: https://linkinghub.elsevier.com/retrieve/pii/S1568494621003859, doi:10.1016/j.asoc.2021.107462.

Journal Pre-proof

Bischoff, E., Ratcliff, M., 1995. Issues in the development of approaches to container loading. Omega 23, 377–390. URL: https://linkinghub.elsevier.com/retrieve/pii/030504839500015G, doi:10.1016/0305-0483(95)00015-G.

Bortfeldt, A., Wäscher, G., 2013. Constraints in container loading – A state-of-the-art review. European Journal of Operational Research 229, 1–20. URL: https://linkinghub.elsevier.com/retrieve/pii/S037722171200937X, doi:10.1016/j.ejor.2012.12.006.

Chen, C., Lee, S., Shen, Q., 1995. An analytical model for the container loading problem. European Journal of Operational Research 80, 68–76. doi:10.1016/0377-2217(94)00002-T.

Davies, A., Bischoff, E.E., 1999. Weight distribution considerations in container loading. European Journal of Operational Research 114, 509–527. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221798001398, doi:10.1016/S0377-2217(98)00139-8.

Dias Saraiva, R., Nepomuceno, N., Rogério Pinheiro, P., 2019. A Two-Phase Approach for Single Container Loading with Weakly Heterogeneous Boxes. Algorithms 12, 67. URL: https://www.mdpi.com/1999-4893/12/4/67, doi:10.3390/a12040067.

Fasano, G., 2004. A MIP approach for some practical packing problems: Balancing constraints and tetris-like items. Quarterly Journal of the Belgian, French and Italian Operations Research Societies 2. URL: http://link.springer.com/10.1007/s10288-004-0037-7, doi:10.1007/s10288-004-0037-7.

Galrão Ramos, A., Oliveira, J.F., Gonçalves, J.F., Lopes, M.P., 2016. A container loading algorithm with static mechanical equilibrium stability constraints. Transportation Research Part B: Methodological 91, 565–581. URL: https://linkinghub.elsevier.com/retrieve/pii/S0191261515302022, doi:10.1016/j.trb.2016.06.003.

Hifi, M., Kacem, I., Nègre, S., Wu, L., 2010. A Linear Programming Approach for the Three-Dimensional Bin-Packing Problem. Electronic Notes in Discrete Mathematics 36, 993–1000. URL: https://linkinghub.elsevier.com/retrieve/pii/S1571065310001277, doi:10.1016/j.endm.2010.05.126.

Junqueira, L., Morabito, R., Sato Yamashita, D., 2012. Three-dimensional container loading models with cargo stability and load bearing constraints. Computers & Operations Research 39, 74–85. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054810001486, doi:10.1016/j.cor.2010.07.017.

Kurpel, D.V., Scarpin, C.T., Pécora Junior, J.E., Schenekemberg, C.M., Coelho, L.C., 2020. The exact solutions of several types of container loading problems. European Journal of Operational Research 284, 87–107. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221719310136, doi:10.1016/j.ejor.2019.12.012.

Liu, C., Smith-Miles, K., Wauters, T., Costa, A.M., 2024. Instance space analysis for 2D bin packing mathematical models. European Journal of Operational Research 315, 484–498. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221723009335, doi:10.1016/j.ejor.2023.12.008.

Liu, S., Tan, W., Xu, Z., Liu, X., 2014. A tree search algorithm for the container loading problem. Computers & Industrial Engineering 75, 20–30. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835214001776, doi:10.1016/j.cie.2014.05.024.

Loh, H., Nee, A., 1992. A packing algorithm for hexahedral boxes. Proceedings of the Conference of Industrial Automation , 115–126.

Martello, S., Pisinger, D., Vigo, D., 2000. The Three-Dimensional Bin Packing Problem. Operations Research 48, 256–267. URL: https://pubsonline.informs.org/doi/10.1287/opre.48.2.256.12386, doi:10.1287/opre.48.2.256.12386.

Martin, M., Alves De Queiroz, T., Morabito, R., 2024. Solving the three-dimensional open-dimension rectangular packing problem: A constraint programming model. Computers & Operations Research 167, 106651. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054824001230, doi:10.1016/j.cor.2024.106651.

Meade, B., Lafayette, L., Sauter, G., Tosello, D., 2017. Spartan HPC-Cloud Hybrid: Delivering Performance and Flexibility. doi:10.4225/49/58ead90dceaaa.

Nascimento, O.X.D., Alves De Queiroz, T., Junqueira, L., 2021. Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm. Computers & Operations Research 128, 105186. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054820303038, doi:10.1016/j.cor.2020.105186.

Paquay, C., Schyns, M., Limbourg, S., 2016. A mixed integer programming formulation for the three-dimensional bin packing

Journal Pre-proof

problem deriving from an air cargo application. International Transactions in Operational Research 23, 187–213. URL: https://onlinelibrary.wiley.com/doi/10.1111/itor.12111, doi:10.1111/itor.12111.

Silva, E.F., Toffolo, T.A.M., Wauters, T., 2019. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Computers & Operations Research 109, 12–27. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054819301030, doi:10.1016/j.cor.2019.04.020.

Smith-Miles, K., Christiansen, J., Muñoz, M.A., 2021. Revisiting where are the hard knapsack problems? via Instance Space Analysis. Computers & Operations Research 128, 105184. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054820303014, doi:10.1016/j.cor.2020.105184.

Smith-Miles, K., Geng, X., 2022. Revisiting Facial Age Estimation With New Insights From Instance Space Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 44, 2689–2697. URL: https://ieeexplore.ieee.org/document/9262085/, doi:10.1109/TPAMI.2020.3038760.

Smith-Miles, K., Muñoz, M.A., 2023. Instance Space Analysis for Algorithm Testing: Methodology and Software Tools. ACM Computing Surveys 55, 1–31. URL: https://dl.acm.org/doi/10.1145/3572895, doi:10.1145/3572895.

Stuckey, P.J., Feydy, T., Schutt, A., Tack, G., Fischer, J., 2014. The MiniZinc Challenge 2008–2013. AI Magazine 35, 55–60. URL: https://onlinelibrary.wiley.com/doi/10.1609/aimag.v35i2.2539, doi:10.1609/aimag.v35i2.2539.

Toffolo, T.A., Esprit, E., Wauters, T., Vanden Berghe, G., 2017. A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. European Journal of Operational Research 257, 526–538. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221716305707, doi:10.1016/j.ejor.2016.07.033.

Tsai, J.F., Wang, P.C., Lin, M.H., 2014. A global optimization approach for solving three-dimensional open dimension rectangular packing problems. Optimization 64, 2601–2618. URL: http://dx.doi.org/10.1080/02331934.2013.877906, doi:10.1080/02331934.2013.877906.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. European Journal of Operational Research 183, 1109–1130. URL: https://linkinghub.elsevier.com/retrieve/pii/S037722170600292X, doi:10.1016/j.ejor.2005.12.047.

Zhao, X., Bennell, J.A., Bektaş, T., Dowsland, K., 2016. A comparative review of 3D container loading algorithms: A comparative review of 3D container loading algorithms. International Transactions in Operational Research 23, 287–320. URL: https://onlinelibrary.wiley.com/doi/10.1111/itor.12094, doi:10.1111/itor.12094.

Zhu, W., Oon, W.C., Lim, A., Weng, Y., 2012. The six elements to block-building approaches for the single container loading problem. Applied Intelligence 37, 431–445. URL: http://link.springer.com/10.1007/s10489-012-0337-0, doi:10.1007/s10489-012-0337-0.

Journal Pre-proof

Journal Pre-proof

- Presents two exact constraint programming models for container loading.
- Block-building improves packing efficiency in the second model for most instances.
- Our CP models outperform state-of-the-art MIP methods on benchmark instances.
- Instance Space Analysis provides further insights into model strength and weakness.

Journal Pre-proof