A hybrid approach combining ant colony optimization and column generation for solving the 3D bin packing problem with rotation

Daniel Bento Maia[a], Denis Borenstein[b,*]

[a]*Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul, Campus Bento Gonçalves, RS, Brazil*
[b]*Management School, Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil*

Abstract

This paper proposes a fast and flexible optimization approach for solving the three-dimensional bin packing problem (3D-BPP) with rotation, a well-known NP-hard problem. The approach combines ant colony optimization and column generation to solve the problem. The hybrid approach has been implemented in several variants by combining the two developed algorithms to be efficiently used in different problem settings, considering weakly to strongly heterogeneous item problems. The algorithms were extensively evaluated using well-known benchmark problems in the literature with varying heterogeneity characteristics and dimensions of bins and items. Although the developed algorithms did not obtain the best results for all the tested instances, they offered competitive solutions very quickly compared to previously reported results. The experiments show that the hybrid approach is flexible, effective, and efficient enough to be an alternative solution method for the classical 3D-BPP problem and to solve online and related routing practical problems, where the 3D-BPP needs to be solved several times as a subproblem.

Keywords: Three-dimensional packing, Container loading problem, Ant colony optimization, Column generation

1. Introduction

The three-dimensional Bin Packing Problem (3D-BPP) has several practical applications, including loading items for transport in warehouses, containers, wagons, trucks, or pallets, provided that the compartment and the items to be accommodated are rectangular. The growth of parcel transportation and containerization [1] demonstrates the practical relevance of the 3D-BPP. This problem can have several variants regarding the number and size of containers, the item's dimensions, and the variability of items (homogeneous, weakly heterogeneous, or strongly heterogeneous). Also, the items to be packed can be rotated or not, depending on the problem's configuration.

*Corresponding author
Email addresses: danielbentomaia@gmail.com (Daniel Bento Maia), denis.borenstein@ufrgs.br (Denis Borenstein)

This work addresses a Three-Dimensional Bin Packing Problem (3D-BPP)/Multiple Container Loading Problem (m-CLP), aiming to efficiently load diverse rectangular items into homogeneous bins/containers to minimize the number of receptacles used. This problem falls under the Single Bin-Size Bin Packing Problem (SBSBPP) category [2]. We focus on problems where (i) the containers are uniform; (ii) the items can be rotated and present a broad range of variability, from weakly to strongly heterogeneous; and (iii) the constraints solely involve the container boundaries and the prevention of item overlap. Pisinger [3] demonstrated that the Container CLP is an NP-hard problem, indicating that the m-CLP also inherits this complexity.

Some solution methods were proposed in the literature to solve the 3D-BPP/m-CLP with rotation [4, 5, 6]. However, they solved problems requiring a large amount of CPU time. These efficiency limitations restrict the application of the developed solution methods to real-world problems. Further, the increase in online environments [1] demands the development of very efficient offline approaches that can be the basis of online solution methods. Further, the current solution methods present specialized algorithms on weak or strongly heterogeneous item problems. To the best of our knowledge, no algorithm has been tested for these situations. The goal of this research is to address both voids in the literature.

This paper introduces a hybrid solution approach for the 3D-BPP/m-CLP with rotation. The proposed method combines Ant Colony Optimization (ACO) and Column Generation (CG) to address the problem with efficacy and efficiency. In this method, ACO quickly explores the solution space, finding good item-packing patterns using a constructive heuristic named Extreme-Point (EP), as introduced by [7]. The integration of EP heuristics within the ACO algorithm enables the exploration of the feasible state space through diversification (exploring new directions) and intensification (delving deeper into promising directions). Subsequently, the CG optimizes the various packing patterns identified by the ACO, aiming to derive an improved solution. The patterns identified by the ACO and the use of several initial solutions allow the generation of integer master problems that commercial solvers can efficiently handle. Computational experiments were carried out for weak and strongly heterogeneous items-based instances, using different combinations of the applied and developed algorithms. The results from our developed algorithms were compared with the best methods described in the literature, considering the spectrum of problems tested. In a global analysis, the developed algorithms presented competitive solutions, simultaneously considering efficiency and effectiveness concerning the best methods described in the literature. The computational result reveals that the developed hybrid approaches, where ant colony optimization and column generations are used simultaneously, obtain the best compromise solutions regarding efficiency and efficacy. In contrast, the ACO algorithm obtained the most efficient solutions but with a loss in the solution quality.

The contributions of our paper are as follows: (i) to propose a heuristic framework for the 3D-BPP that takes the maximum advantages a swarm intelligence metaheuristics and a classical optimization technique can offer; (ii) a flexible approach capable of solving both weak and strongly heterogeneous item problems;

and (iii) to offer very efficient algorithms to solve the problem.

The organization of the manuscript is as follows. The subsequent section provides a concise literature review addressing the 3D-BPP. Section 3 delineates the problem formulation. Section 4 expounds on the hybrid solution method developed for efficient problem resolution. Section 5 details the computational tests and a comparative analysis with state-of-the-art methodologies. Finally, Section 6 states the main findings and outlines future research.

2. Literature review

Given the significance of containerization in logistics, there has been a substantial increase in the literature addressing the 3D-PP in recent decades. This problem exhibits various variants concerning the nature of items to be packed, the quantity, dimensions, types of containers to be utilized, and the planning environment (online and offline). Ali et al. [1] offer an extensive review of 3D-PP, while Zhao et al. [8] provides a comparative analysis of solution methods. This literature review is specifically directed toward offline input minimization 3D-BPP, which is the principal focus of this study.

The problem was initially formalized as a mixed-integer linear programming (MILP) model by [9], considering multiple items and homogeneous bins, but without the inclusion of rotation. The authors addressed small-scale problems using a commercial solver. Martello et al. [10] pioneered an exact branch-and-bound algorithm tailored for the problem, albeit without considering additional real features and rotation. The approach proved effective for instances involving up to 90 items and was subsequently refined in Martello et al. [11], extending its application to a robot packing problem. Nevertheless, due to the complexity of the 3D-BPP problem, exact methods can only be applied to a small number of items.

Several heuristic approaches were then devised to tackle the problem, encompassing local search [12], tabu search [13], and the greedy randomized adaptive search procedure (GRASP) [14, 15]. However, these studies overlooked rotation, limiting their applicability to a subset of large real-world problems [16].

Rotation was introduced to the 3D-BPP by Thapatsuwan et al. [17]. A six-way rotation was considered in the m-CLP, employing artificial immune systems, particle swarm optimization, and genetic algorithms (GA) to address large problems featuring highly heterogeneous items. While these methods generally produced effective solutions, they incurred excessive CPU time, particularly the AIS algorithm. Gonçalves and Resende [4] extended considerations to rotation in 2D and 3D bin problems, developing a biased random key genetic algorithm (BRKGA). This algorithm introduced two new heuristics and an efficient fitness function, yielding competitive results compared to earlier methods without rotation. Feng et al. [18] proposed two hybrid genetic algorithms (HGA) catering to small and large-scale problems, permitting rotation for all items in strongly heterogeneous scenarios. While achieving competitive results, particularly in comparison to existing algorithms, these HGAs exhibited high CPU times, raising concerns about their practical utility in

online problems. Mahvash et al. [5] devised a column generation-based heuristic for 3D-BPP with rotation. The CG approach was used to develop a branch-and-price framework, outperforming previously developed algorithms in instances featuring rotation. More recently, a large neighborhood search algorithm for the m-CLP [6] addressed features such as orientation, stacking, weight limits, handling properties, and stability. However, the method seemed tailored for weakly heterogeneous cases, where items exhibit minimal variety in size.

Recent research has increasingly delved into the multi-pallet loading problem (m-PLP), a variant of the m-CLP. Alonso et al. [19] developed several integer linear models capturing real-world features, such as pallet placement in trucks, weight constraints, and heavy loads, achieving very good solutions for various instances. This work expanded in [15], jointly addressing the m-PLP with truck loading and routing problems through a GRASP algorithm. Despite presenting promising ideas applicable to the broader m-CLP context, challenges persist in handling height considerations. Container heights are a fundamental geometric constraint, distinct from the pallet context's more flexible heights treatment. Additionally, the assumption in m-PLP that items always fit in an integer number of layers does not hold for the m-CLP, introducing uncertainties regarding the transferability of optimization approaches.

Examining the literature underscores a predominant emphasis on solution quality in methods developed for the 3D-BPP. While these methods offer valuable insights, the practical application of the 3D-BPP demands that multiple constraints and integration with vehicle routing and scheduling problems be considered, asking the 3D-BPP be solved very efficiently. Notably, real-time appears to be a new trend, especially when packaging occurs immediately after receiving delivery orders [1]. However, reported solution CPU times are notably high. For instance, the CG method in citepmahvash2017column, applied to eight problem classes with 200 items, resolved only one within the proposed 1,000-second limit. Further, there is a trend in the literature on developing specialized algorithms that consider only weakly or strongly heterogeneous item problems. Although not presented explicitly, the algorithms are developed for only a single type of situation. We did not find a solution method that has been evaluated simultaneously for both heterogeneity of items. This research addresses these gaps in the literature by presenting a heuristic approach that concurrently ensures efficiency and flexibility within reasonable effectiveness in problem resolution. Our work emphasizes not proposing an algorithm with better accuracy than those already developed but a method capable of solving several variants of the classical problem with the greatest possible efficiency. The proposed approach is a foundation for developing algorithms applicable to a large spectrum of 3D-BPP real-world problems incorporating real-world requisites and integration with problems such as vehicle routing and scheduling problems.

3. Model Formulation

The conceptual framework for modeling the 3D-BPP draws upon the contributions of previous work [9, 5]. Each cubic item $i \in N$ is characterized by its dimensions (l_i, w_i, h_i), representing the initial orientation of the item parallel to the XYZ axis. Simultaneously, each container $j \in B$ earmarked for packing is situated with its lower-back-left-bottom corner at the point (0,0,0). The boundaries of a container j along the XYZ axis are denoted by the points L_j, W_j, H_j.

Upon placement within a container j, the coordinates (x_i, y_i, z_i) are decision variables that define the location of the bottom, back, and left vertices of item i. A visual representation of these coordinates is elucidated in Figure 1.

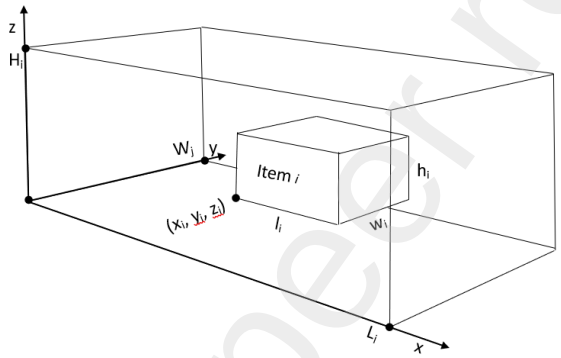


Figure 1: Positioning of items in a bin

Items can assume six orientations following [5], represented by the binary variables $(o_{i1}, o_{i2}, o_{i3}, o_{i4}, o_{i5}, o_{i6})$. These variables assume value one if the item i is in the orientation indicated in Table 1, and 0 otherwise.

Table 1: Six-way rotation

		Projections		
Orientation	Variable	x	y	z
1	o_{i1}	l	w	h
2	o_{i2}	w	l	h
3	o_{i3}	h	w	l
4	o_{i4}	w	h	l
5	o_{i5}	l	h	w
6	o_{i6}	h	l	w

Another crucial representation involves capturing the relative positions between items, which are expressed through binary variables denoted as a_{ik}, b_{ik}, c_{ik}, d_{ik}, e_{ik}, and f_{ik}. These binary variables take on the value of 1 when item i is positioned behind, in front of, on the left side of, on the right side of, below, and above item k, respectively. Further, variables a_{ik}, c_{ik}, and e_{ik} indicate that item i is positioned before item k concerning the XYZ axis, respectively. Conversely, variables b_{ik}, d_{ik}, and f_{iik} indicate that item k is positioned before item i along the XYZ axis, as illustrated in Figure 2.

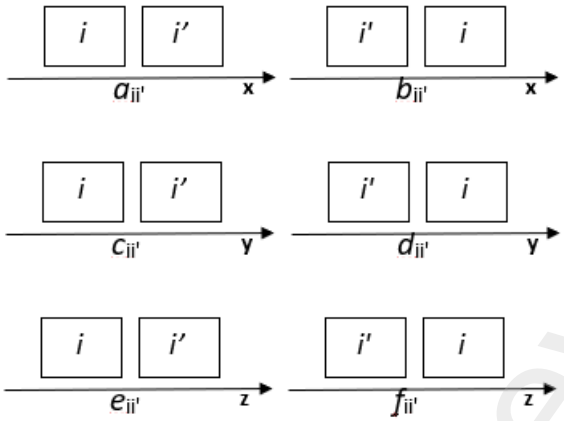


Figure 2: Relative position between items in a bin

Further, we introduce variables s_{ij}, which denotes whether item i is loaded in bin j, and binary variable n_j, with $n_j = 1$ if bin $j \in B$ is used, and $n_j = 0$ otherwise. Based on the above-introduced notation, the 3D-BPP with rotation can be expressed as follows:

$$\min \sum_{j \in B} n_j \tag{1}$$

st

$$x_i + l_i(o_{i1} + o_{i5}) + w_i(o_{i2} + o_{i4}) + h_i(o_{i3} + o_{i6}) \leq L \qquad \forall i \in N \tag{2}$$

$$y_i + l_i(o_{i2} + o_{i6}) + w_i(o_{i1} + o_{i3}) + h_i(o_{i4} + o_{i5}) \leq W \qquad \forall i \in N \tag{3}$$

$$z_i + l_i(o_{i3} + o_{i4}) + w_i(o_{i5} + o_{i6}) + h_i(o_{i1} + o_{i2}) \leq H \qquad \forall i \in N \tag{4}$$

$$x_i + l_i(o_{i1} + o_{i5}) + w_i(o_{i2} + o_{i4}) + h_i(o_{i3} + o_{i6}) \leq x_k + M(1 - a_{ik}) \qquad \forall i,k \in N | i < k \tag{5}$$

$$x_k + l_k(o_{k1} + o_{k5}) + w_k(o_{k2} + o_{k4}) + h_k(o_{k3} + o_{k6}) \leq x_i + M(1 - b_{ik}) \qquad \forall i,k \in N | i < k \tag{6}$$

$$y_i + l_i(o_{i2} + o_{i6}) + w_i(o_{i1} + o_{i3}) + h_i(o_{i4} + o_{i5}) \leq y_k + M(1 - c_{ik}) \qquad \forall i,k \in N | i < k \tag{7}$$

$$y_k + l_k(o_{k2} + o_{k6}) + w_k(o_{k1} + o_{k3}) + h_k(o_{k4} + o_{k5}) \leq y_i + M(1 - d_{ik}) \qquad \forall i,k \in N | i < k \tag{8}$$

$$z_i + l_i(o_{i3} + o_{i4}) + w_i(o_{i5} + o_{i6}) + h_i(o_{i1} + o_{i2}) \leq y_k + M(1 - e_{ik}) \qquad \forall i,k \in N | i < k \tag{9}$$

$$z_k + l_k(o_{k3} + o_{k4}) + w_k(o_{k5} + o_{k6}) + h_k(o_{k1} + o_{k2}) \leq y_i + M(1 - f_{ik}) \qquad \forall i,k \in N | i < k \tag{10}$$

$$a_{ik} + b_{ik} + c_{ik} + d_{ik} + e_{ik} + f_{ik} \geq s_{ij} + s_{kj} - 1 \qquad \forall j \in B, \forall i, k \in N | i < k \qquad (11)$$

$$\sum_{j \in B} s_{ij} = 1 \qquad \forall i \in N \qquad (12)$$

$$\sum_{i \in N} s_{ij} \leq M n_j \qquad \forall j \in B \qquad (13)$$

$$o_{i1} + o_{i2} + o_{i3} + o_{i4} + o_{i5} + o_{i6} = 1 \qquad \forall i \in N \qquad (14)$$

$$x_i, y_i, z_i \geq 0 \qquad \forall i \in N \qquad (15)$$

$$o_{i1}, o_{i2}, o_{i3}, o_{i4}, o_{i5}, o_{i6} \in \{0, 1\} \qquad \forall i \in N \qquad (16)$$

$$a_{ik}, b_{ik}, c_{ik}, d_{ik}, e_{ik}, f_{ik} \in \{0, 1\} \qquad \forall i, k \in N \qquad (17)$$

$$n_j \in \{0, 1\} \qquad \forall j \in B \qquad (18)$$

where M represents a large numerical value.

The objective of the formulation (1) is to minimize the number of bins/containers used. Constraints (2)–(4) ensure that the dimensions of the loaded items do not surpass those of the container. Constraints (5)–(10) prohibit overlapping among loaded items. The constraint set (11) specifies that each item is oriented in only one way. Constraints (12) guarantee that loaded items exhibit permissible relative positions to one another. Constraints (13) guarantee that an item is placed in only one bin. Constraints (14) ensure that containers containing at least one loaded item are utilized. Constraints (15)–(18) establish the feasible range for the decision variables. Given the NP-hard nature of the 3D-BPP, we propose a heuristic method to address the problem efficiently and effectively.

4. Solution Method

The solution framework combines three approaches: extreme point-based (EP) heuristics, ant colony optimization (ACO), and column generation. The first two methods use the EP algorithm to pack items in a bin while searching for near-optimal solutions. ACO and CG can be used in conjunction or individually, but we show that better-quality solutions are obtained when both methods are combined. The next sections will describe the three approaches and how they are combined to find efficient and effective solutions.

4.1. Extreme point-based packing heuristics

A pivotal concern in tackling the 3D-BPP revolves around determining the methodology for efficiently arranging items within a bin. The utilization of loading patterns has been a common approach to packing items into available spaces, aiming to optimize bin utilization. Fanslau and Bortfeldt [20] have categorized loading patterns into distinct rules, such as wall building, stack building, and block building. A comprehensive examination by [1] delves into applying these rules within the context of the 3D-BPP. While these rules

prove valuable for numerous problems, an in-depth literature analysis reveals their optimal use in scenarios characterized by weak heterogeneity, wherein items can be grouped in small sets across multiple dimensions. The Extreme-Point (EP) algorithm [7] appears to be more generic, being capable of handling weakly and strongly heterogeneous problems [5, 21].

The core principle of the EP rule lies in introducing new potential Extreme Points (EPs) upon inserting an item i with dimensions (l_i, w_i, h_i) into a given packing. These EPs, projected onto the orthogonal axes of the containers, include coordinates such as $(x_i + w_i, y_i, z_i)$, $(x_i, y_i + l_i, z_i)$, and $(x_i, y_i, z_i + h_i)$. [7] developed several constructive heuristics grounded in the concept of EPs, employing diverse placement strategies such as first-fit decreasing (FFD) and best-fit decreasing (BFD). The criteria for sorting items varied across dimensions: volume-height, height-volume, area-height, height-area, clustered area-height, and clustered height-area. Notably, these heuristics did not account for item rotation. In contrast, Mahvash et al. [5] introduced a packing heuristic rooted in the EP algorithm, incorporating rotation.

Within our proposed solution framework, we have implemented an EP-based packing heuristic that integrates item rotation. This heuristic aims to generate viable packing patterns for CG and Ant ACO algorithms. Aligned with the foundational concepts of the constructive heuristics outlined by [7], Algorithm 1 follows a similar approach. The notation used in the algorithm description includes EP denoting the set of current extreme points, RS representing the set of all current residual spaces as defined by [7], IP indicating the set of loaded items in a container, and ON standing for an ordered set of items based on a predefined criterion.

The algorithm commences by initializing container j and creating an extreme point, denoted as ep, along with a residual space, referred to as rs, within this container. Subsequently, the feasibility of inserting item $i \in N$ into one of the residual spaces within the set RS is examined using the routine $PutItemInBin$, a modified version derived from the approach developed by [7], which accounts for item rotation. If insertion is deemed feasible, item i is placed in the container corresponding to the residual space rs within set RS with coordinates (x_i, y_i, z_i) at the extreme-point ep corresponding to the respective rs. Item i is then removed from set ON and added to set IP. If none of the available residual spaces accommodates item i, a new container is considered, resulting in the generation of a new extreme-point $ep' = (j, 0, 0, 0)$ and the corresponding residual space $rs = (ep', L, W, H)$. These elements are incorporated into sets EP and RS. In both scenarios, the update of sets EP and ER follows the procedures outlined by [7], which are detailed subsequently.

Set residual space RS can be sorted following seven criteria:

Criterion 1: ordered by smallest volume;

Criterion 2: with the minimum value of z; in the case of a tie, the minimum x is chosen, with the minimum y value used as the last tiebreaker criterion;

Algorithm 1 Extreme point-based packing heuristic

Input: set N, dimensions of the bins (L, W, H)

```
j ← 1
Initialize container j with dimensions LxWxH
ep ← (j, 0, 0, 0), EP ← {rs}
rs ← (ep, L, W, H), RS ← {rs}
IP ← ∅
for each item i ∈ ON do
    for each rs ∈ RS do
        if PutItemInBin(i, RS, EP) then
            Remove item i from ON and set IP ← IP ∪ {i}
        end if
    end for
    if i ∈ ON then
        j ← j + 1
        initialize bin j + 1 with dimensions LxWxH
        ep′ ← (j, 0, 0, 0), EP ← EP ∪ {ep′}
        rs′ ← (ep′, L, W, H), RS ← RS ∪ {rs′}
        PutItemInBin(i, RS, EP)
        Remove item i from N and set IP ← IP ∪ {i}
    end if
    UpdateEP(IP, i, EP)
    UpdateRS(i, EP)
end for
```

Output: IP

Criterion 3: with the minimum value of z; in the case of a tie, the minimum y is chosen, with the minimum x value used as the last tiebreaker criterion;

Criterion 4: with the minimum value of x; in the case of a tie, the minimum y is chosen, with the minimum z value used as the last tiebreaker criterion;

Criterion 5: with the minimum value of x; in the case of a tie, the minimum z is chosen, with the minimum y value used as the last tiebreaker criterion;

Criterion 6: with the minimum value of y; in the case of a tie, the minimum z is chosen, with the minimum x value used as the last tiebreaker criterion;

Criterion 7: with the minimum value of y; in the case of a tie, the minimum x is chosen, with the minimum z value used as the last tiebreaker criterion.

In problems with a cuboid container ($L = W = H$), only criteria 1 and 2 are used. In executing the procedure *PutItemInBin*, the six possible rotations are described in Table 2 based on two moves. The priority move is to rotate an item's longest side (LS) parallel to an axis, followed by moving the second longest side (SLS) parallel to another axis.

Table 2: Rotations rules considered in our study

	Axis		
Rotation rule	X	Y	Z
1	SLS	LS	–
2	–	LS	SLS
3	LS	SLS	–
4	LS	–	SLS
5	–	SLS	LS
6	SLS	–	LS

The rotation rules can be applied in different priority sequences. Each priority sequence,$ps = (r_{i1}, r_{i2}, ..., r_{i6})$ is an ordered list of rotations rules, where the position in the sequence indicates the priority of the application of each rule. Based on experimentation, the following priority sequences are sufficient to give enough diversity to our algorithms based on the rules presented in Table 2, as follows: $ps_1 = (1, 2, 3, 4, 5, 6)$, $ps_2 = (2, 3, 4, 5, 6, 1)$, $ps_3 = (3, 4, 5, 6, 1, 2)$, $ps_4 = (4, 5, 6, 1, 2, 3)$, $ps_5 = (5, 6, 1, 2, 3, 4)$, and $ps_6 = (6, 1, 2, 3, 4, 5)$. Using additional sequences only generates redundant ways of accommodating items in a container. In problems with containers with $L = W = H$, routine $PutItemInBin$ uses only ps_1. If the item cannot be loaded using the current rule, the next rotation rule in ps_1 is used until the item can be accommodated. For other problems, all six priority sequences are used to load an item. The item is accommodated using the priority sequence and the residual space that leads to the container's lowest volumetric use. Algorithm 3 details how these priority sequences and residual criteria are used together to obtain the best container loading.

The procedure $Update - EPs$ initiates by removing the utilized EP, generating six new EPs. The initial three points, considering the utilization of orientation o_{i1}, for example, are as follows: (i) $(x_i + l_i, y_i, z_i)$; (ii) $(x_i, y_i + w_i, z_i)$; and (iii) $(x_i, y_i, z_i + h_i)$. Each initial point is projected in two directions towards the nearest object, either a previously loaded item or a container wall, as illustrated in Figure 3. Any duplicate EPs are eliminated during this process.

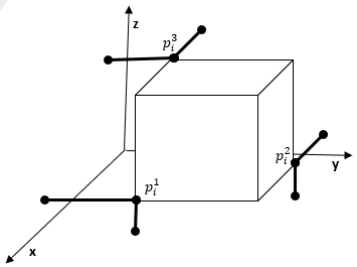


Figure 3: Extreme-Points Projections

Procedure $Update - RS$ deletes used residual space and scrolls through set RS to ensure that any inserted item in a bin occupies some residual space. If so, the residual space is delimited by each item, as demonstrated in Figure 4. For each new extreme point generated in the last execution of $Update - EPs$,

one corresponding residual space is added, equally delimited by the nearest object in each axis.

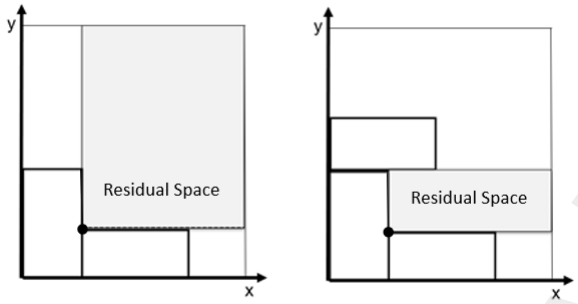


Figure 4: Residual Spaces

4.2. Truncated column generation

Column generation is a widely recognized method for addressing intricate combinatorial optimization problems [22]. In this study, we propose a CG heuristic approach, drawing inspiration from the methodology introduced by [5]. The application of CG is directed towards solving the linear relaxation of the model (19)–(20), denoted as the master problem (MP). The MP encapsulates the classic set-partitioning formulation of the 3D-BPP, where the notation is as follows: let P represent the set of feasible packing patterns, define parameter $a_{ip} = 1$ if item i is loaded in pattern $p \in P$, and $a_{ip} = 0$ otherwise. Additionally, introduce the binary decision variable θ_p, with $\theta_p = 1$ if pattern $p \in P$ is utilized, and $\theta_p = 0$ otherwise.

Master Problem:

$$\min \sum_{p \in P} x_p \tag{19}$$

st

$$\sum_{p \in P} a_{ip}\theta_p = 1 \qquad \forall i \in N \tag{20}$$

$$\theta_p \in \{0, 1\} \qquad \forall p \in P \tag{21}$$

As stated in (19), the primary objective is to minimize the number of patterns encompassing all items. Constraints (18) ensure that each item is incorporated in only one packing pattern.

Given the many patterns involved, CG circumvents exhaustive enumeration by dynamically generating patterns through Dantzig–Wolfe decomposition. This decomposition entails breaking down the Master Problem (MP) into two interconnected sub-problems: a restricted master problem (RMP) and pricing problems. At iteration t, the RMP is a constrained version of the MP, limited to a subset of the variables θ_p. The solution to the RMP yields both a primal and a dual solution. To establish the optimality of the primal solution for the MP, it is necessary to ensure that the reduced costs of as-yet-unconstructed patterns are all

non-negative. The pricing problems are responsible for verifying this condition and, if not met, generating new variables (columns in the RMP) with a negatively reduced cost. In the 3D-BPP, the pricing problem corresponds to determining a feasible packing pattern for a single container. At iteration t, the cost of pattern p is expressed as $1-\sum_{i\in N} a_{ip}\pi_i$, where the changed cost aligns with the reduced cost of the variable θ_p. The dual variable of the RMP, during iteration t, is employed in the pricing problems both to assess the optimality of the solution (ensuring non-negativity of reduced costs for all variables) and to generate new columns (patterns identified in pricing problems with negative reduced costs). This iterative process continues until no additional columns are identified or a predefined number of iterations is reached. If the RMP solution is all integer, a solution is obtained. Otherwise, achieving an integer solution necessitates applying methods such as branch and price or rounding.

The pricing problem can be formulated as a three-dimensional knapsack problem (3D-KP) with rotation to maximize profit. In this formulation, an item i with profit π_i and dimensions (l_i, w_i, h_i) must be packed into a knapsack of size (L, W, H) [5]. With pattern p fixed for each pricing problem, for convenience, we substitute a_{ip} with the binary variable s_i, where $s_i = 1$ if item i is packed in the knapsack, and $s_i = 0$ otherwise. The pricing problem can be formulated as follows:

Pricing Problem:

$$\max \sum_{i\in N} \pi_i s_i \tag{22}$$

st

$$(2)-(10),(14)-(17)$$

$$a_{ik}+b_{ik}+c_{ik}+d_{ik}+e_{ik}+f_{ik} \geq s_i+s_k-1 \qquad \forall i,k\in N|i<k \tag{23}$$

$$s_i \in \{0,1\} \qquad \forall i \in N \tag{24}$$

The objective function (22) maximizes the profit of the pattern. Constraints (23) avoid the overlapping between items.

However, the 3DKP is an NP-hard problem [23]. To efficiently address this situation, we employ a solution strategy akin to that proposed by [5]. The EP-based packing heuristic introduced in Algorithm 3 facilitates packing items into bins. A total of six distinct pricing problems are solved, employing different rules for item sorting: (i) Rule 1 - non-decreasing order of π_i; (ii) Rule 2 - non-decreasing order of $\frac{\pi_i}{l_i w_i h_i}$; (iii) Rule 3 - non-decreasing order of $\frac{\pi_i}{l_i+w_i+h_i}$; (iv) Rule 4 - non-decreasing order of $\frac{\pi_i}{h_i}$; (v) Rule 5 - non-decreasing order of $\frac{\pi_i}{w_i}$; and (vi) Rule 6 - non-decreasing order of $\frac{\pi_i}{l_i}$. These rules constitute set R. The application of these rules expedites the generation of patterns. At each iteration t of the CG approach, all patterns exhibiting negatively reduced costs are incorporated as new columns in the RMP.

Towards mitigating the common stagnation and tailing-off phenomena observed in CG [22], we developed

a truncated CG (TCG) with rounding as delineated in Algorithm 2. The TCG was inspired by the algorithm developed by [24] for the multi-depot vehicle scheduling problem. The algorithm demands the specification of the following parameters: Δ, I, $NStable$, and θ_{min}. The algorithm concludes prematurely if the following conditions are attained: (i) the current solution of the RMP does not change after $Nstable$ iterations, and (ii) in the last I iterations, the solution of the RMP has not decreased by more than Δ. Parameter θ_{min} establishes a minimum level for rounding the variables θ_p.

Algorithm 2 Truncated Column Generation for the 3D-BPP

Step 1 (Initialization): Define parameters I, Δ, $NStable$, and θ_{min}. Set $t \leftarrow 1$, $stable \leftarrow 0$, and $t_0 \leftarrow 1$.

Step 2 (Initial Solution): Sort items in a non-decreasing order of their volumes. Obtain a feasible solution applying Algorithm 1. Set $P \leftarrow S_0$, where S_0 is the solution of Algorithm 1. Generate an RPM with the solution in P.

Step 3 (Restricted master problem): Solve the current RPM with set P as columns, obtaining a primal solution θ_p^t, and cost Z_t^{RMP}. If $(Z_t^{RMP} = Z_{t-1}^{RMP})$ then $stable \leftarrow stable + 1$.

Step 4 (Test for stability): If $(stable > NStable)$, then proceed to Step 8.

Step 5 (Elimination of tailing-off): If $(t - t_0 > I)$ and $\left(\frac{(Z_{t-I}^{RMP} - Z_t^{RMP})}{Z_{t-I}^{RMP}} < \Delta\right)$ then proceed to step 8.

Step 6 (Updating the reduced costs): Update the reduced costs of pattern p using dual variables π_i.

Step 7 (Pricing solution): For each sorting packing rule $r \in R$ do:

Step 7.1 Solve a pricing problem with Algorithm 1 with items sorted by the corresponding Rule r.

Step 7.2 Denote its optimal value by z_{rt}^{sub}

Step 8 (CG termination test): If $(z_{rt}^{sub} \geq 0, \forall r)$ then go to Step 11.

Step 9 (Updating the RMP): For each sorting packing rule $r \in R$ do: If $(z_{rt}^{sub} < 0)$, then set $P \leftarrow P \cup S_r^t$, where S_t^r is the solution vector of the subproblem r at iteration t.

Step 10 (Updating iterations): Set $t \leftarrow t + 1$. Go to Step 3.

Step 11 (Integrality): If $(\theta_p^t \in \{0, 1\})$, it indicates that the solution has been identified. Return the solution θ_p^t. Terminate the process.

Step 12 (Rounding): Conduct a rounding procedure for the non-integer variables in θ_p^t. If $(\theta_p^t \geq \theta_{min}), \forall p \in P$, update $\theta_p^t \leftarrow 1$. In the absence of such variables meeting the criterion, assign one to the highest value of θ_p^t. Set $t \leftarrow t + 1$. Go to Step 3.

The procedure commences in Step 1, initializing the process by fixing parameter values. Step 2 obtains an initial solution for the CG by resolving the EP-based packing heuristic (Algorithm 1). Step 3 solves the current RMP using a commercial MILP solver. Steps 4 and 5 examine whether truncation of the CG is required to prevent stagnation and tailing-off, respectively. Step 5 involves updating the reduced costs of the patterns in the set P (pool of solutions). In Step 7, pricing problems are addressed, considering various item sorting rules. Step 8 checks if the optimal solution of all subproblems is non-negative and if the current RMP solution cannot be improved, and Step 11 is executed.

Conversely, a negative solution for a subproblem generates a new column associated with a specific packing pattern in the current RMP, generating an extended RMP (Step 9). With added columns, the new RMP is solved in Step 3. Step 11 is an integrality test in the current solution, while Step 12 performs rounding in the no integer values of variables θ_p. Certain variables θ_p are set to 1 if their corresponding

fractional values surpass the threshold value θ_{min}. Without such variables, the highest value of θ_p is set to 1. The integer values of θ_p are considered in a new RMP, solved in Step 3.

Rounding can make the master problem unfeasible. However, due to the reliability of the EP-based algorithm, such a situation did not arise in our experiments. Rounding was effective and efficient in addressing the 3D-BPPP.

4.3. Ant colony optimization

Ant Colony Optimization (ACO) is a reinforcement learning algorithm for solving difficult combinatorial problems [25]. ACO involves an iterative process that generates additional feasible solutions based on the pheromone trail left by successful feasible solutions identified during the solution process. The foraging behavior of real ants in search of food inspires the approach. Initially, ants explore the environment randomly, and upon discovering food, they leave a pheromone trail whose intensity depends on the quality of the find. ACO has successfully tackled challenging integer problems, including bin-packing problems [26, 27], but we cannot find any application for the 3D-BPP with rotation.

For our problem, the pheromone trail represents the order in which items are loaded into containers. The set of ants is represented by ANT, with the cardinality determined by the size of items $|N|$ multiplied by the parameter Q ranging between zero and one, as proposed by [28]. The ACO routine applied to the 3D-BPP with rotation is outlined in Algorithm 3, inspired by the work of [29].

The algorithm initializes running the procedure *pheromoneValue*, in which each item $i \in N$ receives an initial value of pheromone τ_i greater than zero. The probability of each item $i \in N$ being packed is computed by

$$p_i = \frac{[\tau_i]^\alpha [\eta(c_i)]^\beta}{\sum_{j \in N} [\tau_j]^\alpha . [\eta(c_j)]^\beta}$$

where the heuristic value, c_i, receives the volume of item i, the parameter α controls the influence of the pheromone, while parameters β and η control the influence of the heuristic value c_i. These values mitigate the weights of the pheromones and the heuristic intuition in the probability computation. Furthermore, $\eta = Q_\eta \max_{i \in N} V_i$, where Q_η is a parameter defined between zero and one and V_i is the volume of item i.

In the *rouletteWheel* procedure, the items are ordered in set ON using a standard roulette wheel selection method, in which the items with the highest probability are more likely to be drawn. The items in ON are packed using the EP algorithm (Algorithm 1) to find a feasible solution, following residual space criteria and priority rotation rules sequences according to the container dimensions.

After ANT iterations are completed, the best iteration solution S_{iter} is selected. This solution is used in routine *updatePheromone*. This procedure comprises two steps: (i) evaporation and (ii) pheromone deposit on the trail. Evaporation allows us to forget trails that were scarcely used. It is obtained by an evaporation coefficient e, and it is calculated as $\tau_i = \tau_i(1 - e)$. The pheromone deposit on the train is calculated using

Algorithm 3 Ant Colony Optimization Algorithm

Input: set N, container dimension (L,W,H), parameter $limitTime$

```
Initialize set ANT, with cardinality |ANT| = Q|N|
totalTime ← 0
initialize pheromoneValue(τ)
S_bs ← ∅
while totalTime ≤ limitTime do
    P ← calculateProbability(τ)
    S_iter ← ∅
    for each element in set ANT do
        IP, O ← ∅
        ON ← rouletteWheel(P)
        if L = H = W then
            for each Rotation Rule in ps_1 do
                for each Residual Space Criteria do
                    Run the EP-based packing algorithm (Algorithm 1).
                    Update set IP.
                    if S_iter = ∅ or IP < S_iter then
                        S_iter ← IP
                    end if
                end for
            end for
        else
            for each Priority Sequence ps_i, i = 1, ..., 6 do
                for each Rotation Rule in ps_i do
                    for each Residual Space Criteria do
                        Run the EP-based packing algorithm (Algorithm 1).
                        Update set IP.
                        if S_iter = ∅ or IP < S_iter then
                            S_iter ← IP
                        end if
                    end for
                end for
            end for
        end if
    end for
    updatePheromone(τ, S_iter)
    updateTotalTime
    if S_bs = ∅ or S_iter < S_bs then
        S_bs ← S_iter
    end if
end while
```

Output: S_{bs}

$$\tau_i = \tau_i + \delta - (ON[i] - 1)d + \frac{L}{S_{iter}}a \tag{25}$$

where δ is the pheromone deposited, parameter d represents the percentage decreases that are inversely proportional to the position of item i on the best current solution, $ON[i]$. Furthermore, the pheromone receives an increase by a term that is computed based on the quality of the best iteration solution, in comparison by the lower bound $L = \frac{\sum_{iin_N} V_i}{V_j}$, where V_j is the volume of container j, mitigated by parameter a. In the best solution of the iteration, the items receive an amount of pheromone inversely proportional to the loading order.

The need to introduce combinations of multiple rotation rule application sequence priorities and residual space utilization criteria arises from the need to provide flexibility to our optimization approach. This combination allows the container space to be better utilized, considering situations of weakly and strongly heterogeneous items. Although these combinations increase the algorithm's complexity, the overall performance is not greatly affected. As several possibilities are tested in each iteration, the algorithm finds good solutions faster, requiring fewer total iterations.

4.4. Hybrid Algorithm

The hybrid algorithm is grounded in the conceptualization that a collection of feasible solutions generated by the Ant Colony Optimization (ACO) algorithm can enhance the efficacy of Algorithm 2. Each solution obtained through ACO represents a feasible pattern adhering to the integrality constraint of variable θ_p and can thus be directly incorporated into the Restricted Master Problem (RMP). Depending on the problem's item count, all patterns generated by the ACO algorithm may be inserted into the RMP. However, for instances with more than 100 items, efficiency considerations led to the inclusion of only the patterns from the best solution of each iteration.

Moreover, we curated a set comprising the best solutions from the ACO to furnish multiple initial feasible solutions to the commercial solver. These solutions, known as MIP starts, serve as suggestions for the solver to identify an incumbent solution. Typically, the solver selects the optimal MIP start as the incumbent solution. The presence of an incumbent solution enables the solver to prune portions of the search tree during the solution process, resulting in more compact branch-and-cut trees. Additionally, the solver can employ advanced neighborhood search and polishing heuristics with an incumbent solution.

Given that the ACO algorithm furnishes robust initial solutions for the Column Generation (CG), we solved the RMP without relaxing variables θ_p. Furthermore, to attain superior bounds, constraints (20) were substituted with

$$\sum_{p \in P} a_{ij}\theta_p \geq 1 \qquad \forall i \in N \tag{26}$$

This change ensures that all solutions by the RPM are feasible during the CG execution. Redundant patterns can be easily eliminated from the final solution to repair this relaxation.

Let γ_i be the dual variables of constraints (26). Define parameters $NStart$ as the number of MIP-starts from the ACO algorithm and $NSol$ as the number of solutions from the ACO algorithm to generate columns in the RMP. Algorithm 4 describes the developed hybrid algorithm.

Algorithm 4 Hybrid ACO+CG algorithm

Step 1 (Initialization): Set $t \leftarrow 1$, the solution pool $P \leftarrow \varnothing$, $t \leftarrow 1$. Define θ_{min}.

Step 2 (ACO solutions): Solve Algorithm 3. Set S_t with the $NSol$ best solutions obtained by the ACO algorithm. Set $P \leftarrow P \cup S_t$.

Step 3 (RMP): Solve the current RPM model (22), (26) with set P as columns and $NStart$ as MIP-starts. A primal solution θ_p^t is obtained.

Step 4 (Updating the reduced costs): Update the reduced costs of pattern p using dual variables γ_i.

Step 5 (Pricing solution): For each sorting packing rule $r \in R$ do:
Solve a pricing problem with Algorithm 1 with items sorted by the corresponding Rule r. Denote its optimal value by z_t^r.

Step 6 (CG termination test): If $(z_t^r \geq 0, \forall r)$ then go to Step 8.

Step 7 (Updating the RMP): For each sorting packing rule $r \in R$ do:
If $(z_{tr}^{sub} < 0)$, then set $P \leftarrow P \cup S_t^r$.
Go to Step 9.

Step 8 (Integrality): If $(\theta_{\mathbf{t}} \in \{0, 1\})$, it indicates that the solution has been identified. Return the solution θ_p^t. Terminate the process.

Step 9 (Rounding): Conduct a rounding procedure for the non-integer variables in θ_p^t. If $(\theta_p^t \geq \theta_{min}), \forall p \in P$, update $\theta_p^t \leftarrow 1$. In the absence of such variables meeting the criterion, assign one to the highest value of θ_p^t. Set $t \leftarrow t + 1$. Go to Step 3.

5. Numerical experiments

To assess the efficacy of our proposed methodology, we carried out tests with strongly heterogeneous problems [10] and weakly heterogeneous problems [30] available in the literature, comparing the results with previously developed algorithms. Our developed algorithms were programmed in C++, using Gurobi to solve the Mixed-Integer Linear Programming (MILP) problems. The computational tests were conducted on an Intel computer with a 2.1GHz Intel processor with 8M RAM. We also developed a visualization program implemented in Python to validate and better communicate the results.

5.1. Strongly heterogeneous instances

We used the benchmark instances introduced by [11] to test our algorithm's performance in strongly heterogeneous instances. The instances are categorized into eight classes, each containing four problems featuring 50, 100, 150, and 200 items, resulting in 320 instances. Table 4 presents the bin dimensions used in each class. The type of items to be packed is generated following uniformly random distributions with parameters either based on the dimensions of the bin or in fixed values. Different combinations of item

types are used in each class. For each problem, 10 instances are randomly generated. [5] comprehensively explains how the dimensions of bins and items are generated for each class. The instances can be generated using a C code available at http://hjemmesider.diku.dk/ pisinger/codes.html.

The developed algorithm needs to define some parameters. For the TCG, we set $Nstable = 50$, $I = 30$, $\Delta = 5\%$, and $\theta_{mim} = 0.7$. Table 3 presents the values of the main ACO parameters. Finally, we set $NStart = 10$ and $NSol = 0.1|N|$, both used in the ACO+CG algorithm. The parameters' values were defined by experimentation using randomly generated instances based on the code on Pisinger's Website cited above.

Table 3: Parameter values for the ACO algorithm

Parameter	Value
Q	0.5
β	3
Q_η	0.5
α	1.5
δ	0.5
a	0.5
e	0.05

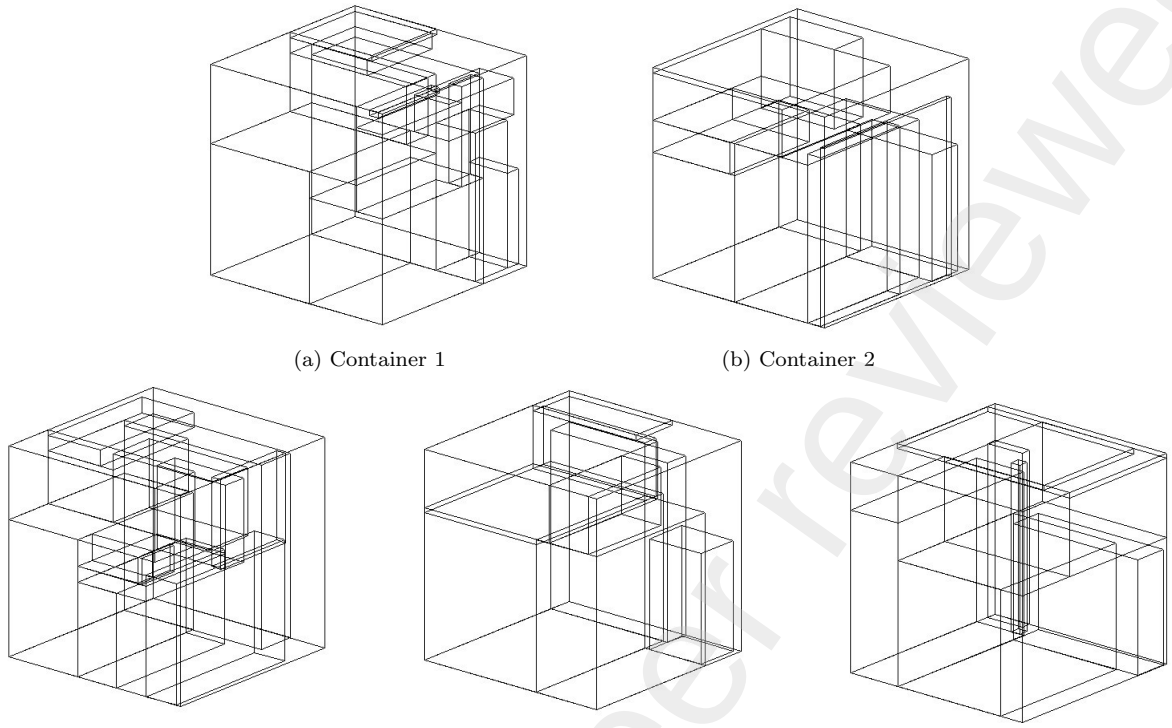
We compared various implementations of our proposed approach, using the results obtained by [5] as a benchmark. The notation employed to represent the implemented algorithms is as follows: (i) TCG: the truncated column generation algorithm (Algorithm 2), (ii) ACO: the ant colony algorithm (Algorithm 3), (iii) ACO+TCG: the hybrid algorithm (Algorithm 4). As benchmarks, we use the column-generation approach by [5] (M-CG), the algorithm BRKGA with six rotations [4] (6rBRKGA), and the hybrid genetic algorithm (HGA-S) introduced in [18].

Table 4 comprehensively compares the developed implementations across the 320 instances, considering solution quality and efficiency. The first three columns present the class, bin dimension, and number of items of each problem. The following columns show the average number of bins (Obj) and the average CPU time (s) obtained for the ten instances of each table line for algorithms TCG, ACO, ACO+TCG, and M-CG, respectively. The last two columns show the average number of bins (Obj) for algorithms 6rBKRGA and HGA-S. CPU times are not reported for these two algorithms. The best solution for each line is identified in bold. Following [5], we discard the solutions of algorithm 6rBRKGA for 200 items in classes 6, 7, and 8. The solutions presented do not respect the lower bounds computed by [31] for the 3D-BPP with rotation. Fig. 5 shows the packing solution, using five containers, of algorithm ACO+CG for class 7 with 50 items.

The results in Table 4 show that algorithm ACO+CG dominates algorithms TCG and ACO, presenting a larger or equal number of bins for all instances. If we add to the fact that the TCG CPU times are almost three times higher, on average than the CPU times required by the ACO+CG algorithm, the TCG algorithm can be excluded from further analysis. ACO is maintained in the analysis due to its high efficiency.

Table 4: Results for the strongly heterogeneous instances

	Number of	TCG		ACO		ACO+CG		M-CG		6rBRKGA	HGA-S
Class	Items	Obj	CPU	Obj	CPU	Obj	CPU	Obj	CPU	Obj	Obj
1	50	11.7	12	11.8	1	**11.6**	5	11.7	10	11.8	11.7
	100	23.2	48	23.8	2	**22.8**	23	22.9	60	23	24.2
	150	32.5	531	33.2	54	**31.3**	85	**31.3**	200	31.7	33.5
	200	44.9	771	44.9	21	43	392	**42.8**	1000	43.4	47.4
2	50	12.2	10	12.1	1	**11.8**	4	**11.8**	10	**11.8**	**11.8**
	100	23	54	23	1	**22.4**	21	**22.4**	70	22.5	23.6
	150	32.4	357	33.5	1	31.5	82	**31.4**	100	**31.5**	32.3
	200	44.3	894	44.1	15	42.3	167	**41.9**	1000	42.5	44.7
3	50	11.9	21	11.9	1	**11.4**	5	11.8	9	11.6	11.9
	100	22.9	38	23.3	1	**22.4**	21	22.5	60	22.6	23.9
	150	33.1	485	33.9	7	32.3	91	**32**	300	32.4	35.3
	200	44.3	876	44.1	13	**42.1**	470	**42.1**	1000	42.3	44.4
4	50	28.9	1	**28.9**	1	**28.9**	1	**28.9**	1	28.9	**28.8**
	100	58.4	1	**58.4**	1	**58.4**	1	**58.4**	3	**58.4**	**58.2**
	150	86.4	1	**86.4**	1	**86.4**	1	**86.4**	4	**86.4**	86.5
	200	118.3	1	**118.3**	1	**118.3**	1	**118.3**	17	**118.3**	**118.3**
5	50	7.7	9	7.5	1	7.5	4	7.5	5	7.5	**7.3**
	100	14.2	38	13.9	1	**13.7**	25	**13.7**	100	**13.7**	13.9
	150	19.6	456	19.1	2	19	89	**18.6**	400	**18.6**	18.8
	200	26.9	639	25.7	3	25.6	316	**25.3**	1000	**25.3**	25.9
6	50	9.6	12	**8.9**	1	**8.9**	5	**8.9**	5	9.4	9.1
	100	18.7	124	18.2	1	18.1	37	**17.9**	100	18.9	18.3
	150	28.8	510	28	2	27.9	108	**27.5**	200	28.2	28.2
	200	38	745	36	1	36	451	**35.5**	1000	33.3	37
7	50	7	7	6.5	1	6.5	4	6.6	16	6.4	**6.3**
	100	12.4	166	11.3	41	**11.2**	25	**11.2**	200	11.3	**11.2**
	150	16.3	822	14.7	2	14.7	56	**14.3**	500	14.6	15
	200	24	902	22.3	5	22.3	246	**21.9**	1000	20.3	22.5
8	50	8.7	12	**8.3**	1	**8.3**	4	**8.3**	10	9.2	8.5
	100	18	42	17.7	1	**17.5**	19	17.7	22	18.2	18.3
	150	23.7	932	22.8	4	22.5	53	**22.1**	500	**22.1**	22.5
	200	30.1	789	28.1	1	28.1	514	**27.7**	1000	24.8	29.4
Sum all classes		9321	10306	9209	190	9047	3326	9013	9902	9009	9287



(a) Container 1 (b) Container 2

(c) Container 3 (d) Container 4 (e) Container 5

Figure 5: Example of a complete solution by algorithm ACO+CG - class 7, 50 items

Table 4 reveals no dominance between ACO+CG, M-CG, 6rBRKGA, and HGA-S algorithms. Algorithm M-CG obtains the best solutions for 24 instances, followed by ACO+CG, 6rBRKGA, and HGA-S, with 17, 9, and 7 best solutions, respectively. Algorithm ACO+CG presents competitive solutions considering all evaluated instances. On the one hand, ACO+CG increases the solutions of the M-CG algorithm by 0.32%, on average. On the other hand, the algorithm reduces the solution values of the 6rBRKGA and HGA-S algorithms by 0.94% and 2.37%, on average, respectively. In about 46.9% of the evaluated problems, ACO+CG presents better or equal results than M-CG. In the remaining ones, the largest differences are by a small margin. Algorithm ACO presents worse results than M-CG, ACO+CG, and 6rBRKGA, with overall gaps of 2.17%, 1.82%, and 0.94%, respectively, but outperforms the HGA-S algorithm, with an average gap of 0.20%.

However, the ACO and ACO+CG algorithms stand out regarding computational efficiency. They are significantly more efficient than M-CG. ACO achieves acceptable solutions about 51 times faster on average than M-CG. ACO does not require more than 60 seconds to obtain solutions with the abovementioned gaps for all problems. Algorithm ACO+CG is much slower than the ACO algorithm (about 55.73% on average), but about three times faster on average than the M-CG algorithm to obtain solutions with average gaps of 0.32%. These results are significant if we think the classical 3D-BPP problem needs to be solved several

times as a subproblem when applied to real problems.

5.2. Weakly heterogeneous instances

The data set IMM by [30] was used to evaluate the developed algorithm for weakly heterogeneous instances. The data set contains 47 different instances. Two to five different item types with a total number of items from 47 to 180 must be packed into a homogeneous container. Each instance has a different container size. The objective is to minimize the number of containers.

Table 5 compares the results by our developed algorithms, ACO and ACO+CG, with the best current results by the literature, reported by the exact algorithms introduced in [32] (KSJSC) and by the large Neighbourhood search (LNS) algorithm by [6] The solutions for the LNS reported in the table were obtained by running the algorithm for 180s for all 47 instances.

Unsurprisingly, KSJSC and LNS obtained the best results from all the methods compared, with a slight advantage over the former, which offers the best solutions for all instances. It seems both algorithms were designed by focusing on weakly heterogeneous instances. KSJSC uses this condition to reduce the complexity of the problem, while LNS solves the problem using wall, layer, and column heuristics that suit weakly heterogeneous instances. Algorithm ACO+CG obtained solutions with average gaps of 2.61% and 1.59% in comparison with KSJSC and LNS, requiring extra 18 and 11 containers compared to KSJSC and LNS to pack all items in the 47 instances. Nevertheless, ACO+CG found equal solutions than KSJSC and LNS in 29 and 36 instances, respectively. Again, algorithm ACO showed a worse performance than ACO+CG. ACO required 23 and 16 extra containers compared to KSJSC and LNS, respectively, leading to average gaps of 3.34% and 2.30%, respectively, in the objective function. ACO's performance was hampered by some instances, mainly IMM23, IMM29, and IMM30, in which the algorithm became stagnant at a local optimum. We could not identify the relation between these bad results and the instances' characteristics. Additional tests are required. ACO found the same solutions for 28 instances compared to KSJSC, and 33 instances compared to LNS. ACO+CG reduced the number of containers of the initial solutions of the algorithm ACO in 9 instances, with significant reductions for instances IMM23, IMM29, and IMM30.

However, our algorithms are more efficient in a global analysis than KSJSC and LNS. KSJSC simultaneously presents the fastest times for some instances and very high times for others, showing a certain lack of robustness concerning efficiency. Repeating the pattern for strongly heterogeneous instances, ACO was extremely fast, reducing the average CPU time to solve the 47 instances in factors 23 and 16 compared with KSJSC and LNS, respectively. ACO was quicker than LNS in all 47 instances. Although ACO was slower than KSJSC in 8 instances, the reduction in the remaining ones was significant. ACO+CG was also more efficient than KSJSC and LNS, but with more modest CPU reduction factors, averaging 5.15 and 1.89 times compared with these two algorithms, respectively.

Table 5: Results for weakly heterogeneous instances

		Item	ACO		ACO+CG		KSJSC		LNS
Instance	# Items	Types	Obj	CPU	Obj	CPU	Obj	CPU	Obj
IMM1	70	2	25	1	25	62	25	0.04	25
IMM2	70	2	10	1	10	120	9	2.66	10
IMM3	180	4	19	1	19	61	19	8.83	19
IMM4	180	4	26	1	26	121	26	8.97	26
IMM5	180	4	51	2	51	121	51	82.09	51
IMM6	103	3	10	1	10	121	10	4.52	10
IMM7	103	3	16	1	16	121	16	3.28	16
IMM8	103	3	4	1	4	60	4	6.55	4
IMM9	110	2	20	1	20	180	19	0.85	19
IMM10	110	2	55	1	55	60	55	0.02	55
IMM11	110	2	17	35	17	120	16	31.82	17
IMM12	95	3	53	1	53	120	53	3.88	53
IMM13	95	3	25	1	25	120	25	1.3	25
IMM14	95	3	28	1	28	118	27	6.52	27
IMM15	95	3	11	26	11	120	11	7.28	11
IMM16	95	3	26	1	26	63	26	0.76	26
IMM17	95	3	8	1	8	60	7	6.22	7
IMM18	47	3	2	1	2	60	2	14.69	2
IMM19	47	3	3	1	3	60	3	10.66	3
IMM20	47	3	5	1	5	60	5	49.9	5
IMM21	95	5	21	1	20	121	20	3217.77	20
IMM22	95	5	9	1	9	60	8	15	8
IMM23	95	5	21	1	20	138	19	930.95	20
IMM24	72	4	6	1	6	60	5	11.5	5
IMM25	72	4	5	1	5	60	5	3600	5
IMM26	72	4	3	1	3	60	3	15	3
IMM27	95	3	5	1	5	66	4	6.71	5
IMM28	95	3	10	12	10	64	9	5.27	10
IMM29	118	4	19	2	17	180	16	1116.65	17
IMM30	118	4	24	1	23	180	22	3600	22
IMM31	118	4	14	2	14	180	13	3600	13
IMM32	90	3	4	1	4	60	4	10.98	4
IMM33	90	3	5	1	5	62	4	8.91	4
IMM34	90	3	9	1	9	160	8	26.41	8
IMM35	84	2	2	13	2	70	2	5.21	2
IMM36	84	2	14	1	14	120	14	0.39	14
IMM37	102	3	23	1	23	120	23	3.14	23
IMM38	102	3	45	1	45	60	45	0.77	45
IMM39	102	3	15	1	15	120	15	2139.96	15
IMM40	85	4	9	42	9	120	8	104.49	8
IMM41	85	4	15	1	15	120	15	27.96	15
IMM42	90	3	5	1	5	64	4	12	4
IMM43	90	3	3	1	3	60	3	15	3
IMM44	90	3	4	1	4	60	3	747.4	4
IMM45	99	4	3	1	3	62	3	3600	3
IMM46	99	4	2	1	2	66	2	15	2
IMM47	99	4	3	30	3	61	3	15	3
Total Sum			712	202	707	4482	689	23102.31	696

5.3. Discussion

The developed algorithms did not present the best results for all tested instances compared with previous solution methods from the literature. The best results were obtained for problems with strong heterogeneity in the items, obtaining the smallest gaps about the benchmarks considered. The ACO+CG algorithm found the best-known solutions for some instances of this class. For the remaining problems, our algorithms were less effective. However, the results demonstrate our algorithm's versatility in solving different class problems. This feature was achieved by calling the EP algorithm by combining several rotation rules and different criteria for using the residual space, totaling forty-two combinations. This fact helps to optimize container space by effectively managing weakly and strongly heterogeneous items, considering different sizes of containers.

A relevant result that needs further exploration is the ACO algorithm's better relative performance for weakly heterogeneous problems. For these problems, solutions are presented closer to the ACO+CG algorithm than the strongly heterogeneous ones. Considering the ACO's speed in solving the problem, this can be useful when solving online problems.

Further, another important attribute of our developed algorithms seems to be the good compromise between the quality and efficiency of the solution process. The ACO algorithm presented good results very quickly, reducing the solution time for both problem variants compared to previously reported results. Some of the reductions are quite significant. These results indicate great potential to improve the performance of the ACO algorithm in terms of effectiveness as long as there is no significant loss of efficiency. Some recent studies report the possibility of improving the solution quality obtained by ACO by aggregating local search, algorithm refinements, and careful parallel implementation [33, 34]. In turn, the ACO+CG algorithm presented solutions that were slightly inferior to some benchmarks but more efficient, with an overall average CPU utilization reduction factor of around two. This algorithm can also improve its performance by introducing improvements in the ACO-based algorithm.

This flexibility in solving different classes of the problem and the significant reductions in CPU time, with relatively small losses in the objective function, are important when thinking that the 3D-BPP is a classic problem that needs to be customized by introducing real features or by integrating it with other problems, such as routing and scheduling vehicles carrying containers or bins, to be useful in practice. The two most relevant algorithms, ACO and ACO+CG, can play important roles in developing efficient and effective solution methods for these two contexts for several variants of the 3D-BPP.

6. Conclusions

This paper introduces a hybrid approach combining ant colony optimization and column generation for solving the 3D bin packing problem with rotation. The algorithms were extensively tested, using strongly

and weakly heterogeneous problems. To the best of our knowledge, this is the first time that algorithms have been tested for both of these problems for 3DBPP with rotation. The developed heuristic approach offered very competitive results, especially ACO and ACO+CG variants, for both classes of problems. Conversely, variant ACO+CG offered the best compromise solutions regarding quality and efficiency. This variant can be a good option in contexts where efficiency and solution quality are simultaneously relevant criteria. Variant ACO found solutions very quickly but with some loss of efficacy. We recommend its application in online packing problems, where the 3D-BPP is solved several times as a subproblem, and solution quality is an important but not essential criterion. Nevertheless, the final selection among these two variants is case-dependent and will depend on further experimentation.

The research is proceeding to integrate real features in our developed approach, such as the stability of items within the bin and heterogeneous containers. Incorporating these features will increase the complexity of the problem, requiring changes in our algorithms to find good solutions within an acceptable time. Moreover, we intend to incorporate a large search (LS) algorithm into our hybrid approach to improve the quality of the solution. Levine and Ducatelle [26] have observed that large search has a great potential to improve the performance of ACO-based algorithms. With this improvements, we plan to handle the online 3D-BPP and large 3D-BPPs with more than 1,000 items to be packed.

Funding: This work was supported by the Brazilian National Council for Scientific and Technological Development [grant number 303941/2023-5]

References

[1] S. Ali, A. G. Ramos, M. A. Carravilla, J. F. Oliveira, On-line three-dimensional packing problems: a review of off-line and on-line solution approaches, Computers & Industrial Engineering (2022) 108122.

[2] G. Wäscher, H. Haußner, H. Schumann, An improved typology of cutting and packing problems, European Journal of Operational Research 183 (2007) 1109–1130.

[3] D. Pisinger, Heuristics for the container loading problem, European Journal of Operational Research 141 (2002) 382–392.

[4] J. F. Gonçalves, M. G. Resende, A biased random key genetic algorithm for 2D and 3D bin packing problems, International Journal of Production Economics 145 (2013) 500–510.

[5] B. Mahvash, A. Awasthi, S. Chauhan, A column generation-based heuristic for the three-dimensional bin packing problem with rotation, Journal of the Operational Research Society (2017) 1–13.

[6] Ö. Şafak, G. Erdoğan, A large neighbourhood search algorithm for solving container loading problems, Computers & Operations Research 154 (2023) 106199.

[7] T. G. Crainic, G. Perboli, R. Tadei, Extreme point-based heuristics for three-dimensional bin packing, INFORMS Journal on Computing 20 (2008) 368–384.

[8] X. Zhao, J. A. Bennell, T. Bektaş, K. Dowsland, A comparative review of 3D container loading algorithms, International Transactions in Operational Research 23 (2016) 287–320.

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

[10] S. Martello, D. Pisinger, D. Vigo, The three-dimensional bin packing problem, Operations Research 48 (2000) 256–267.

[11] S. Martello, D. Pisinger, D. Vigo, E. D. Boef, J. Korst, Algorithm 864: General and robot-packable variants of the three-dimensional bin packing problem, ACM Transactions on Mathematical Software 33 (2007) 7:1–7:12.

[12] O. Faroe, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin-packing problem, Informs Journal on Computing 15 (2003) 267–283.

[13] A. Lodi, S. Martello, D. Vigo, Heuristic algorithms for the three-dimensional bin packing problem, European Journal of Operational Research 141 (2002) 410–420.

[14] F. Parreño, R. Alvarez-Valdés, J. F. Oliveira, J. M. Tamarit, A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing, Annals of Operations Research 179 (2010) 203–220.

[15] M. T. Alonso, R. Alvarez-Valdés, F. Parreño, A GRASP algorithm for multi-container loading problems with practical constraints, 4OR 18 (2020) 49–72.

[16] O. X. do Nascimento, T. A. de Queiroz, L. Junqueira, Practical constraints in the container loading problem: Comprehensive formulations and exact algorithm, Computers & Operations Research 128 (2021) 105186.

[17] P. Thapatsuwan, P. Pongcharoen, C. Hicks, W. Chainate, Development of a stochastic optimisation tool for solving the multiple container packing problems, International Journal of Production Economics 140 (2012) 737–748.

[18] X. Feng, I. Moon, J. Shin, Hybrid genetic algorithms for the three-dimensional multiple container packing problem, Flexible Services and Manufacturing Journal 27 (2015) 451–477.

[19] M. T. Alonso, R. Alvarez-Valdés, M. Iori, F. Parreño, Mathematical models for multi container loading problems with practical constraints, Computers & Industrial Engineering 127 (2019) 722–733.

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

[21] T. G. Crainic, G. Perboli, R. Tadei, TS2PACK: A two-level tabu search for the three-dimensional bin packing problem, European Journal of Operational Research 195 (2009) 744–760.

[22] M. E. Lübbecke, J. Desrosiers, Selected topics in column generation, Operations Research 53 (2005) 1007–1023.

[23] J. Egeblad, D. Pisinger, Heuristic approaches for the two-and three-dimensional knapsack packing problem, Computers & Operations Research 36 (2009) 1026–1049.

[24] A.-S. Pepin, G. Desaulniers, A. Hertz, D. Huisman, A comparison of five heuristics for the multiple depot vehicle scheduling problem, Journal of Scheduling 12 (2009) 17–30.

[25] Y. Liu, B. Cao, H. Li, Improving ant colony optimization algorithm with epsilon greedy and levy flight, Complex & Intelligent Systems 7 (2021) 1711–1722.

[26] F. Ducatelle, J. Levine, Ant colony optimisation for bin packing and cutting stock problems, in: UK Workshop on Computational Intelligence (UKCI-01), Edinburgh, 2001.

[27] M. E. Silveira, S. M. Vieira, J. M. Da Costa Sousa, An aco algorithm for the 3d bin packing problem in the steel industry, in: Recent Trends in Applied Artificial Intelligence: 26th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2013, Amsterdam, The Netherlands, June 17-21, 2013. Proceedings 26, Springer, 2013, pp. 535–544.

[28] H. Duan, G. Ma, S. Liu, Experimental study of the adjustable parameters in basic ant colony optimization algorithm, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore, IEEE, 2007, pp. 149–156. URL: https://doi.org/10.1109/CEC.2007.4424466. doi:10.1109/CEC.2007.4424466.

[29] M. Dorigo, C. Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243–278. URL: https://www.sciencedirect.com/science/article/pii/S0304397505003798. doi:https://doi.org/10.1016/j.tcs.2005.05.020.

[30] N. J. Ivancic, An integer programming-based heuristic approach to the three-dimensional packing problem, Ph.D. thesis,

Case Western Reserve University, 1988.

[31] M. A. Boschetti, New lower bounds for the three-dimensional finite bin packing problem, Discrete Applied Mathematics 140 (2004) 241–258.

[32] D. V. Kurpel, C. T. Scarpin, J. E. P. Junior, C. M. Schenekemberg, L. C. Coelho, The exact solutions of several types of container loading problems, European Journal of Operational Research 284 (2020) 87–107.

[33] R. Skinderowicz, Improving Ant Colony Optimization efficiency for solving large TSP instances, Applied Soft Computing 120 (2022) 108653.

[34] Y. Zhang, A hybrid algorithm combining ant colony optimization and large neighborhood search based on reinforcement learning for integrated container-truck scheduling problem, in: 2023 8th International Conference on Intelligent Computing and Signal Processing (ICSP), 2023, pp. 1354–1358. doi:10.1109/ICSP58490.2023.10248613.