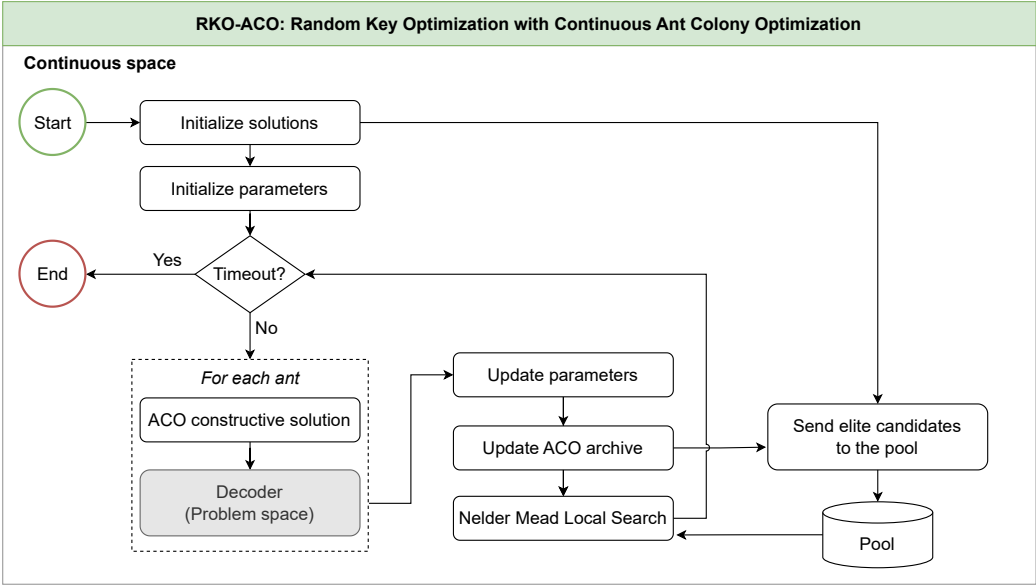


Graphical Abstract

Random-Key Metaheuristic and Linearization for the Quadratic Multiple Constraints Variable-Sized Bin Packing Problem

Natalia Alves Santos, Marlon Jeske, Antônio Augusto Chaves



Highlights

Random-Key Metaheuristic and Linearization for the Quadratic Multiple Constraints Variable-Sized Bin Packing Problem

Natalia Alves Santos, Marlon Jeske, Antônio Augusto Chaves

- Advances the state-of-the-art for the QMC-VSBPP with exact and metaheuristic methods.
- Proposes a new linearized formulation, eliminating complex quadratic terms.
- Introduces lower bounds via exact solutions of the linearized model using Gurobi.
- Develops the RKO-ACO: an adaptation of continuous ACO for random-key optimization.
- Establishes updated best-known solutions for a set of benchmark instances.

Random-Key Metaheuristic and Linearization for the Quadratic Multiple Constraints Variable-Sized Bin Packing Problem

Natalia Alves Santos^{a,*}, Marlon Jeske^{a,b}, Antônio Augusto Chaves^a

^a*Federal University of São Paulo, Av. Cesare Mansueto Giulio Lattes, 1201, São José dos Campos, 12247-014, SP, Brazil*

^b*Aeronautics Institute of Technology, Praça Marechal Eduardo Gomes, 50, São José dos Campos, 12228-900, SP, Brazil*

Abstract

This paper addresses the *Quadratic Multiple Constraints Variable-Sized Bin Packing Problem* (QMC-VSBPP), a challenging combinatorial optimization problem that generalizes the classical bin packing by incorporating multiple capacity dimensions, heterogeneous bin types, and quadratic interaction costs between items. We propose two complementary methods that advance the current state-of-the-art. First, a linearized mathematical formulation is introduced to eliminate quadratic terms, enabling the use of exact solvers such as Gurobi to compute strong lower bounds—reported here for the first time for this problem. Second, we develop RKO-ACO, a continuous-domain Ant Colony Optimization algorithm within the Random-Key Optimization framework, enhanced with adaptive Q-learning parameter control and efficient local search. Extensive computational experiments on benchmark instances show that the proposed linearized model produces significantly tighter lower bounds than the original quadratic formulation, while RKO-ACO consistently matches or improves upon all best-known solutions in the literature, establishing new upper bounds for large-scale instances. These results provide new reference values for future studies and demonstrate the effectiveness of evolutionary and random-key metaheuristic approaches for

*Corresponding author.

Email addresses: `natalia.santos26@unifesp.br` (Natalia Alves Santos), `jeske@unifesp.br` (Marlon Jeske), `antonio.chaves@unifesp.br` (Antônio Augusto Chaves)

solving complex quadratic packing problems.

Keywords:

Bin Packing Problem, Random-Key Optimization, Ant Colony Optimization, Metaheuristics, Mathematical Programming

1. Introduction

The bin packing problem (BPP) is a well-studied combinatorial optimization problem that has been extensively researched since its introduction in 1979 and has been applied in various fields, such as scheduling and cutting stock problems. The primary objective of the BPP is to minimize the packing cost of a set of items into a finite number of bins, each with a fixed capacity, while considering the weights of the items.

The variable-sized bin packing problem (VSBPP), one of many variants of BPP, allows packing combinations in different types of bins, each with its own cost and capacity. Besides, BPPs and VSBPPs with conflicts often penalize when certain pairs of items are assigned to the same bin (Ekici, 2023), or offer rewards for items of the same category packed together (Santos et al., 2019). Recently, a novel variant of the VSBPP was introduced, known as the Quadratic Multiple Constraint VSBPP (QMC-VSBPP) (Meng et al., 2022). This variant considers multiple dimensions for bin capacities and item weights, and penalizes pairs of items packed in different bins. As the VSBPP, the QMC-VSBPP is classified as NP-hard, indicating its computational complexity.

In this work, we address the QMC-VSBPP, aiming to improve both solution quality and the quality of lower bounds. To this end, we formulate three core research questions: (1) how to linearize the model of the QMC-VSBPP to achieve improved lower bounds and enhance the solution quality; (2) to explore whether the RKO framework, incorporating a continuous-domain ACO algorithm, can successfully generate high-quality solutions for this problem; and (3) to compare the efficiency of the solutions obtained through the proposed approach against existing benchmarks. Based on these questions, we hypothesize that the proposed linearization of the QMC-VSBPP model will reduce the solution gap and yield tighter lower bounds, consequently increasing the performance of standard solvers like Gurobi. Furthermore, we tailored the RKO framework with the continuous-domain ACO, which will produce superior solutions compared to traditional methods found in the

literature, leading to the establishment of new best-known solutions for the QMC-VSBPP.

The main contributions of this work are:

- Advances the state-of-the-art for the QMC-VSBPP with exact and metaheuristic methods.
- Proposes a new linearized formulation, eliminating complex quadratic terms.
- Introduces lower bounds via exact solutions of the linearized model using Gurobi.
- Develops the RKO-ACO: an adaptation of continuous ACO for random-key optimization.
- Establishes updated best-known solutions for a set of benchmark instances.

This paper is organized as follows. Section 2 reviews the literature on the bin packing problem with conflicts, bio-inspired heuristics, and RKO. Section 3 presents the model formulation, and the proposed linearization. Section 4 provides the theoretical background on random-key optimization and ACO for continuous domains and introduces the developed RKO-ACO algorithm. Section 5 details the implementation of RKO-ACO tailored to the QMC-VSBPP, with a focus on the solution decoder. Section 6 presents computational experiments and analyzes the results obtained from exact and metaheuristic approaches. Section 7 concludes with the main contributions, addresses research questions and validates the hypotheses.

2. Literature Review

The bin packing problem and its variants, including the variable-sized bin packing problem, have been extensively studied over the past decades. This review is organized into three subjects: BPP with conflicts, bio-inspired metaheuristics for BPP, and the RKO framework and its applications.

Several extensions of BPP incorporate conflict constraints, where specific pairs of items cannot be packed together or incur penalties when placed in

separate bins. Early studies on the classical BPP with conflicts (BPC) proposed the first-fit decrease (FFD) heuristic adaptations and clique computations on the conflict graph (Gendreau et al., 2004). Later works improved the results using set covering formulations with column generation and tabu search (Muritiba et al., 2010). Online variants, where items are partially or completely unknown a priori, were also explored (Epstein et al., 2011). The min-conflict packing problem (Khanafer et al., 2012) introduced a bi-objective formulation minimizing both violated conflicts and the number of bins. Overall, the main challenge lies in resolving the conflict graph rather than the packing constraints themselves (Khanafer et al., 2012).

Variants that consider item fragmentation (BPPC-IF) allow items to be partially packed while respecting conflicts. Heuristics for sequential packing based on item degree in the conflict graph were proposed (Ekici, 2021), and later improved by refining item selection criteria (Fleszar, 2022). The density of the conflict graph strongly influences the solution gap, with higher density typically leading to worse outcomes. Other related variants include packing compatible categories (Santos et al., 2019) and Open-End BPC, where the last item may exceed bin capacity (Balık et al., 2025). These studies commonly employed Variable Neighborhood Search (VNS) with initial solutions generated by modified FFD heuristics.

Recent developments include the VSBPP with conflicts (VSBPPC) and item fragmentation (Ekici, 2022), where maximal clique calculations provide lower bounds, and heuristics generate independent subsets of compatible items to guide packing. In Ekici (2023), the VSBPPC was formulated as a mixed-integer linear programming (MILP) problem and solved using a Large Neighborhood Search (LNS) metaheuristic. Firstly, initial solutions are constructed using a greedy heuristic by sequentially packing items into existing or new bins while minimizing cost. Then, the LNS procedure iteratively destroys a subset of bins and reconstructs them using the initial solution strategy combined with local search operations, such as moving or swapping items, and changing bin types to smaller costs as much as possible.

In Meng et al. (2022), the VSBPPC was extended to the Quadratic Multiple-Constraint VSBPP (QMC-VSBPP), introducing 3 to 5 attribute dimensions, where each item has a d -dimensional weight vector and each bin has corresponding d -dimensional capacity limits. This formulation models scenarios such as cloud computing resource allocation, where dimensions represent attributes like CPU and RAM. In addition to capacity constraints, the conflict graph imposes extra costs when specific item pairs are placed

in different bins. The authors formulated the problem as an MILP and proposed a VNS algorithm, with initial solutions generated via hierarchical clustering. The VNS employs neighborhood strategies including bin swapping, bin deletion, and shortest-path-based item permutation. Performance was evaluated against adapted VNS and Genetic Algorithm methods from the VSBPP state-of-the-art literature (Haouari and Serairi, 2009; Hemmelmayr et al., 2012) as well as the CPLEX solver. Across 96 newly generated benchmark instances, the proposed VNS consistently found better solutions in less computational time than the other evaluated approaches.

Bio-inspired optimization methods have been widely applied to BPP and its variants. Approaches include squirrel search algorithms (El-Ashmawi and Elminaam, 2019), evolutionary heuristics (Stawowy, 2008), and genetic algorithms using grouping strategies and parallel islands (Kucukyilmaz and Kiziloz, 2018). Particle Swarm Optimization (PSO) has been explored for multi-objective BPPs (Liu et al., 2008), while Ant Colony Optimization (ACO) has been applied in classic form with local search (Levine and Ducatelle, 2004), as well as enhanced with differential pheromone strategies (Ali et al., 2024). Moreover, Socha and Dorigo (2008) demonstrated that ACO, when adapted for continuous domains, achieves superior performance across a range of combinatorial problems, highlighting its potential for mixed-variable optimization challenges.

The concept of random keys was introduced by Bean (1994) to facilitate search over continuous spaces in combinatorial problems. As a precursor to the RKO framework, Chaves et al. (2024) adapted GRASP for random-key optimization, demonstrating effectiveness across several NP-hard problems. More recently, the RKO framework has been successfully applied to classical optimization problems, including the Traveling Salesman Problem, Set Covering, Vehicle Routing, and Node Capacitated Graph Partitioning (Chaves et al., 2025), as well as real-world applications such as robot motion planning (Schuetz et al., 2022), operating room scheduling (Vieira et al., 2025), and cutting stock problems (Silva et al., 2025).

The reviewed studies demonstrate the versatility of bin packing extensions, the effectiveness of bio-inspired metaheuristics, and the adaptability of the RKO framework to complex combinatorial problems. However, to the best of our knowledge, the QMC-VSBPP has not been explored in the literature beyond its formulation and instance generation. This gap provides the foundation for our work, which seeks to advance exact and metaheuristic approaches for this challenging variant.

3. Problem Definition

The QMC-VSBPP proposed by Meng et al. (2022) involves a set of items $I = \{1, 2, \dots, n\}$, a collection of bin types $M = \{1, 2, \dots, m\}$, and multiple attribute dimensions $D = \{1, 2, \dots, d\}$.

Each item $i \in I$ is characterized by a weight vector w_{ir} in d dimensions, where $w_{ir} > 0$ represents the weight of item i in dimension $r \in D$. For any distinct pair of items $i, s \in I$, a symmetric non-negative cost c_{is} is incurred if the items are placed in separate bins.

Each bin type m is associated with a fixed cost c_m and a capacity vector Q_{md} , where $Q_{md} > 0$ denotes the capacity in dimension d . Bins can be instantiated in unlimited numbers, with their types chosen from M .

Consider $B = \{1, 2, \dots, n\}$ as the set of opened bins, with n as the maximum number of bins considering the assignment of one item per bin. The binary variable x_{ij} controls if the item i is assigned to bin j , and y_{jm} determines if bin j is assigned to the type m .

The problem objective is to allocate all items into bins satisfying the conditions: the total weight of items in any bin does not exceed its capacity in any dimension; each opened bin must be assigned to a specific bin type; and both incurring costs, the cost per bin type utilized and the penalty cost for packing certain items in separate bins, must be minimized.

The mathematical model for the QMC-VSBPP is expressed as follows (Meng et al., 2022):

$$\min \sum_{j \in B} \sum_{m \in M} c_m y_{jm} + \sum_{j \in B} \sum_{i \in I} \sum_{s \in I} c_{is} x_{ij} (1 - x_{sj}) \quad (1)$$

$$\sum_{j \in B} x_{ij} = 1 \quad \forall i \in I \quad (2)$$

$$\sum_{m \in M} y_{jm} \leq 1 \quad \forall j \in B \quad (3)$$

$$\sum_{i \in I} w_{id} x_{ij} \leq \sum_{m \in M} Q_{md} y_{jm} \quad \forall j \in B, d \in D \quad (4)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in B \quad (5)$$

$$y_{jm} \in \{0, 1\} \quad \forall j \in B, m \in M \quad (6)$$

The first term in the objective function (1) sums the costs of the bins utilized, while the second term represents the penalty costs by placing certain items in separate bins. Constraint (2) ensures that each item is assigned to only one bin. Constraint (3) sets each bin to a single type. Constraint (4) ensures that the total weight of items assigned to a bin does not exceed its capacity in any dimension. Finally, constraints (5) and (6) set the decision variables' binary domain.

3.1. Model Linearization

Linearization is a common approach used to simplify optimization problems with quadratic terms, reducing computational complexity and improving the efficiency of solvers in finding optimal solutions.

For the QMC-VSBPP model linearization, a new set of binary variables, z_{ijs} , is introduced to capture the interaction between items i and s when they are placed in different bins j , in substitution of the objective function's quadratic term $x_{ij}(1 - x_{sj})$. The linearized model is defined as follows:

$$\min \sum_{j \in B} \sum_{m \in M} c_m y_{jm} + \sum_{j \in B} \sum_{i \in I} \sum_{s \in I} c_{is} z_{ijs} \quad (7)$$

$$\sum_{j \in B} x_{ij} = 1 \quad \forall i \in I \quad (8)$$

$$\sum_{m \in M} y_{jm} \leq 1 \quad \forall j \in B \quad (9)$$

$$\sum_{i \in I} w_{id} x_{ij} \leq \sum_{m \in M} Q_{md} y_{jm} \quad \forall j \in B, d \in D \quad (10)$$

$$z_{ijs} \leq x_{ij} \quad \forall i \in I, j \in B, s \in I \quad (11)$$

$$z_{ijs} \leq 1 - x_{sj} \quad \forall i \in I, j \in B, s \in I \quad (12)$$

$$z_{ijs} \geq x_{ij} - x_{sj} \quad \forall i \in I, j \in B, s \in I \quad (13)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in B \quad (14)$$

$$y_{jm} \in \{0, 1\} \quad \forall j \in B, m \in M \quad (15)$$

$$z_{ijs} \in \{0, 1\} \quad \forall i \in I, j \in B, s \in I \quad (16)$$

The new constraints involving z_{ijs} ensure that these variables correctly represent the interaction between items. Constraints (11)-(13) refer to the model linearization. Constraint (16) defines the z_{ijs} as binary.

4. Random Key Optimization (RKO)

A typical combinatorial optimization problem consists of a finite ground set $E = \{1, \dots, n\}$, an objective function $f : 2^E \rightarrow \mathbb{R}$, and a set of feasible solutions $F \subseteq 2^E$. The goal is to find an optimal solution $S^* \in F$ such that $f(S^*) \leq f(S) \quad \forall S \in F$ for minimization problems. Within the random keys approach, a problem solution can be encoded as a vector χ of real numbers, where $\chi = \{x_1, x_2, \dots, x_n\}$ and $x_i \in [0, 1)$. Thus, χ is a point in the \mathbb{R}^n hypercube, allowing the search process to operate over a continuous domain instead of the discrete solution space originally defined by the problem. To convert a random-key as a problem solution, a decoder function G is employed to map the vector χ into a problem-specific solution $S \in F$, it is $S = G(\chi)$.

4.1. RKO Framework

The RKO framework, proposed by Chaves et al. (2025)¹, provides a modular environment in which metaheuristics search directly in the continuous random-key space, while a dedicated decoding function maps each random-key vector into a feasible discrete solution. This clear separation of search and problem-specific logic makes algorithms built within the RKO framework largely problem-independent: metaheuristic operators explore the continuous space, and the decoding function handles all problem-specific constraints and requirements.

In summary, each iteration follows a standardized cycle:

1. Random key generation: initialize an individual or population of random keys.
2. Solution decoding: map each random key to a problem solution, evaluate it, and optionally apply local search in the problem domain.
3. Continuous-space exploration: apply continuous-space search methods, such as the Nelder–Mead algorithm (Chaves et al., 2024), to generate new random keys.

To enhance performance, the framework integrates several reusable, problem-independent strategies. Multi-threading allows each processing thread to run an independent metaheuristic instance, whether multiple runs of the same

¹<https://github.com/RKO-solver>

algorithm or different algorithms in parallel. A shared solution pool collects candidate solutions from all threads, creating a set of elite solutions. A restart mechanism helps escape stagnation by fully reinitializing both the population and the solution pool. Finally, an online parameter tuning component, based on Q-learning, dynamically adjusts search parameters to balance intensification and diversification over time.

These strategies, along with the continuous random-key search design, make the framework compatible with any metaheuristic able to operate in continuous space (Chaves et al., 2025). In this work, the RKO is extended with a continuous-domain ACO metaheuristic, as presented in the next sections.

4.2. ACO for Continuous Domains

Ant Colony Optimization is a well-established metaheuristic inspired by the foraging behavior of ants, traditionally applied to discrete combinatorial problems. To address continuous optimization challenges, the ACO paradigm has been extended to operate in continuous domains (Socha and Dorigo, 2008). In this variant, the pheromone model is represented by an archive of elite solutions, each encoded as a vector in the continuous search space.

In the continuous ACO approach, each solution in the archive is assigned a weight reflecting its quality and rank. New candidate solutions are generated by probabilistically selecting an archive member according to these weights and then sampling each variable from a Gaussian distribution centered at the corresponding value in the chosen solution. The standard deviation of this Gaussian is adaptively determined by the dispersion of the archive in each dimension, scaled by a parameter ξ .

Formally, let k denote the archive size, q the selection pressure parameter, and ξ the exploration scaling parameter. The weight ω_l for the l -th ranked solution is computed as:

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} \exp\left(-\frac{(l-1)^2}{2q^2k^2}\right) \quad (17)$$

where $l = 1$ corresponds to the best solution. The probability of selecting a solution is proportional to its weight.

For each variable i in the random-key vector, the standard deviation σ_l^i for sampling is given by:

$$\sigma_l^i = \xi \frac{1}{k-1} \sum_{e \neq l} |s_e^i - s_l^i| \quad (18)$$

where s_l^i is the i -th variable of the selected solution, and the sum is over all other archive members. In cases where the computed σ is numerically close to zero (i.e., negligible dispersion), a large default σ (e.g., 0.9999) is used to preserve exploration.

Therefore, new solutions are created as follows:

1. Compute weights for all archive solutions and derive their selection probabilities.
2. Select an archive solution according to these probabilities.
3. For each variable, sample a new value from a Gaussian distribution centered at the selected solution's value, with standard deviation as above. Rejection sampling is used to ensure that variables remain within the feasible range $[0, 1)$.

This process is iterated for a specified number of ants (number of candidate solutions) per generation. The archive is then updated by merging the new solutions with the existing archive and retaining only the top k solutions according to fitness.

4.3. RKO-ACO

The proposed RKO-ACO metaheuristic integrates a continuous Ant Colony Optimization approach, as detailed in the previous section, with Q-learning for dynamic parameter adaptation and the Nelder–Mead algorithm for local search. The algorithm maintains an archive of elite solutions that guides the search and a global solution pool shared across multi-threaded executions of the RKO-ACO metaheuristic.

The step-by-step procedure of the algorithm is illustrated in Figure 1. In summary, the RKO-ACO operates as follows:

1. **Archive Initialization:** The solution archive is initialized as a randomized random-key, or using a constructive heuristic tailored to the problem. Each solution is encoded as a random-key vector, and the best initial solution is submitted to the global solution pool.

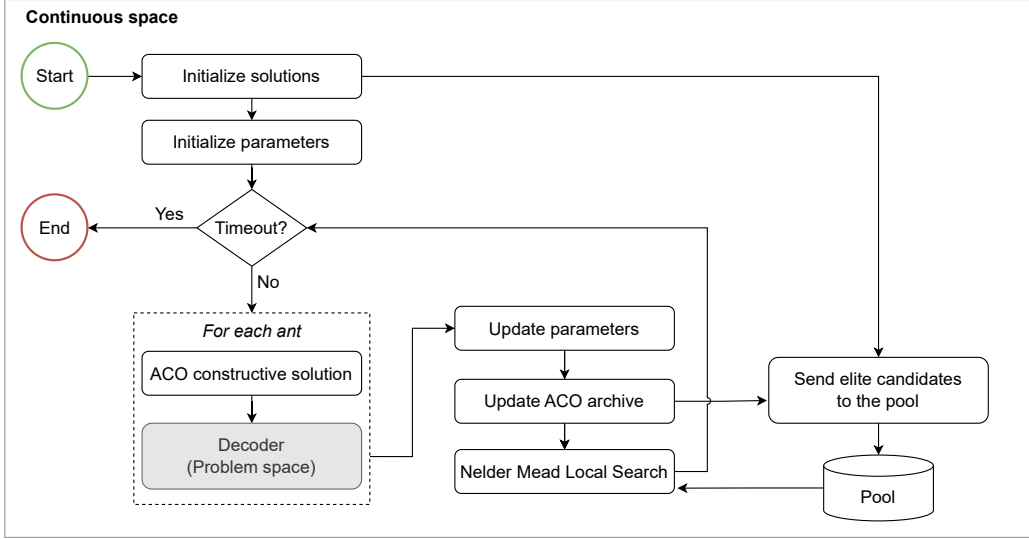


Figure 1: RKO-ACO flowchart

2. **Parameter Initialization:** The algorithm initializes key ACO parameters within predefined value ranges, delimiting the space explored by the Q-learning agent during the optimization process.

3. ACO Generations Loop:

- *ACO Constructive Solution:* Each ant constructs a new solution, as described in Section 4.2, resulting in a random-key. After being decoded into a problem solution, its objective value (fitness) is evaluated.
- *Local Search:* The best solution of the current generation undergoes local improvement using the Nelder–Mead algorithm.
- *Archive Update:* The archive is updated with the newly generated solutions and pruned to retain only the best individuals.
- *Solution Pool Update:* When a new best solution is found, it is added to the shared solution pool, which retains the global best solutions across all threads.
- *Parameter Update:* The Q-learning agent receives a reward based on the relative improvement in solution quality. This feedback guides the selection of parameter values and reinforces combinations that lead to better performance.

4. **Termination and Restart:** The process repeats until a predefined computational time limit is reached. If a restart condition is triggered, the solution pool is cleared and new solutions are initialized, restarting the ACO generation process to escape stagnation.

A detailed description of RKO-ACO’s online parameter tuning and a novel caching system for the solution decode step are presented in the following subsections.

4.3.1. *Online Parameter Tuning*

To enhance metaheuristic performance, ACO parameters are dynamically adapted during execution using a Q-learning approach. The parameters archive size, number of ants, q , and ξ are randomly initialized within predefined ranges. These parameters are subsequently updated at each generation based on a reward function, presented in Chaves et al. (2025), that evaluates improvements in solution quality, favoring configurations that lead to better solutions.

For the archive size, changes between generations require careful management. When the archive decreases, the worst-performing solutions, ranked by objective function value, are removed. Conversely, when the archive size increases, additional solutions are created by sampling new random-key vectors with values uniformly distributed in $[0, 1]$. Each vector is decoded into a problem solution, and this process continues until the new archive capacity is reached.

This adaptive mechanism manages the trade-off between diversity and efficiency: larger archives enhance exploration by maintaining a more diverse population, whereas frequent or substantial expansions increase computational cost due to the evaluation of newly generated solutions.

4.3.2. *Caching*

As a contribution to the RKO framework, we implemented a fixed-size queue-based caching system to avoid recalculating solutions that have already been evaluated on the Decode step. The cache stores previously evaluated solutions as key-value pairs, where the key is a string representation of the solution and the value is its objective function. The cache maintains a maximum size ($Q_{\max} = 1000$); when the limit is reached, the oldest entry (the one inserted earliest) is removed to accommodate new solutions. The caching process is presented in Algorithm 1.

Algorithm 1: Queue-Based Caching

Input: Decoded solution key k , objective value v (optional), cache \mathcal{Q} of max size Q_{\max}

```
1 if  $k$  exists in  $\mathcal{Q}$  then
2   | return cached value for  $k$ 
3 else
4   | if size of  $\mathcal{Q} = Q_{\max}$  then
5     | Remove the entry at the front of the queue from  $\mathcal{Q}$ ;
6     | Insert  $(k, v)$  at the back of the queue from  $\mathcal{Q}$ ;
7   | return  $v$ 
```

Although the cache may introduce additional overhead at the beginning of the metaheuristic execution, when most solutions are new, it helps reduce computation time during convergence by skipping repeated evaluations of known solutions.

5. RKO-ACO for QMC-VSBPP

In this section, the implementation of the QMC-VSBPP using the RKO-ACO metaheuristic is described, including the generation of initial solutions, random-key decoding, post-processing, local search within the problem space, and objective function tie breakers.

5.1. Notations

Throughout this work, we use the following notations, which are fundamental to the description of our algorithms and analysis.

The adjacency matrix $L = [\ell_{ij}]$, also called the *links*, encodes the pairwise penalties for separating items. Specifically, $\ell_{ij} = c_{ij}$ if items i and j can feasibly be packed together in at least one bin type (i.e., their combined weights do not exceed the bin's capacity in any dimension).

The *aggregate neighbor penalty* of an item i is denoted by ℓ_i^+ and defined as follows:

$$\ell_i^+ = \sum_{j=1}^N \ell_{ij} \quad \forall i \in N$$

where N is the number of items and ℓ_{ij} is the penalty (link cost) for separating items i and j .

The *aggregate neighbor weight* of an item i is denoted by w_i^+ and defined as follows:

$$w_i^+ = \sum_{j=1}^N \mathbb{I}[\ell_{ij} > 0] \left(\sum_{d=1}^D w_{jd} \right) \quad \forall i \in N$$

where w_{jd} is the weight of item j in dimension d , and $\mathbb{I}[\ell_{ij} > 0]$ is 1 if items i and j are linked, and 0 otherwise.

The *largest bin type*, denoted m^* , is defined as the bin type with the largest aggregated capacity:

$$m^* = \arg \max_{m \in M} \left(\sum_{d=1}^D Q_{md} \right)$$

Lastly, consider that the bin types are sorted in ascending order of their cost.

5.2. Generating Initial Solutions

Initial solutions are constructed using a semi-greedy randomized procedure that prioritizes the assignment of items with high aggregate neighbor weight w_i^+ while maintaining feasibility. At each iteration, a randomized candidate list (RCL) of unassigned items with links is formed, and the item with the highest w_i^+ in the RCL is selected as the starting point for a new bin of the default type m^* . Additional items are greedily added to the bin based on the highest penalty ℓ_{ij} to the start item, case they fit within the bin's capacity constraints. This process repeats until all items with links are assigned. Any remaining unassigned items are placed in the existing bin that minimizes the link cost and can accommodate the item, or in a new bin if necessary. The complete procedure, including post-processing steps, is detailed in Algorithm 2.

5.3. Decoding and Item to Bin Assignment

Each solution is represented as a random-key vector of length $n+3$, where n is the number of items. The first n elements determine the allocation order: items are sorted in ascending order of their random-key values, and this order is used for assignment. The last three elements of the vector encode algorithmic parameters: (i) the allocation strategy, (ii) the number of initial bins to open, and (iii) the probability of applying item relocation during local search.

Algorithm 2: Semi-Greedy Initial Solution Generation

```
1 Initialize all items as unassigned;
2 Initialize an empty set of opened bins  $B$ ;
3 Randomly select RCL size  $r$  uniformly between 3 and 5% of  $N$ ;
4 while there are unassigned items with  $\ell_i^+ > 0$  do
5   Build RCL: take up to  $r$  unassigned items in input order;
6    $i^* \leftarrow \max(w_i^+)$  in RCL;
7   Open a new bin  $b \in B$  of type  $m^*$ . Assign  $i^*$  to  $b$ ;
8   Mark  $i^*$  as assigned;
9   while true do
10    Among all unassigned items  $j$ , find  $j^*$  with the highest link cost  $\ell_{i^*j}$  to
11     $i^*$  such that  $j^*$  fits in the current bin  $b$ ;
12    if no such  $j^*$  exists then
13       $\perp$  break
14    Assign  $j^*$  to  $b$  and mark as assigned;
15 for each remaining unassigned item  $i$  do
16   Assign  $i$  to a bin in  $B$  that minimizes link cost  $\ell_{ij}$  and fits, or open a new
17   bin (Algorithm 3);
18 Apply post-processing (bin type replacement, bin merging);
19 Apply local search with probability  $p = 1$ ;
20 Encode the final allocation as a random-key vector  $\chi$ ;
21 Compute the objective value;
22 return  $rk$ 
```

Algorithm 3: Item to Best Bin Assignment

```
Input: Opened bins  $B$ , unassigned item  $i$ 
1  $B^* \leftarrow \{ b \in B : i \text{ fits in } b \}$ ;
2 if  $B^*$  is not empty then
3    $b^* \leftarrow \arg \min_{b \in B^*} \sum_{j \in b} \ell_{ij}$ ;
4   Assign  $i$  to  $b^*$ ;
5 else
6   Open new bin  $b$  of type  $m^*$ . Assign  $i$  to  $b$ ;
7 return Item assignment
```

We propose three item-to-bin assignment strategies. The first reuses the semi-greedy procedure for building initial solutions, with the order of unassigned items following the decoded random-key vector. The second and third strategies are adapted from the random-key decoding method for the node-capacitated graph partitioning problem (NCGPP) (Chaves et al., 2025). In

these, items are processed in the decoded order: each is placed in the first bin that can accommodate it without exceeding capacity, or in a new bin if none are suitable. The second strategy uses a random-key parameter to define the number of bins to pre-open, whereas the third starts with no bins opened a priori. The Algorithm 4 describes the assignment process.

Algorithm 4: Random-key Decode and Item Assignment

Input: Random-key vector χ of length $n + 3$

- 1 Sort the first N elements of χ in ascending order to obtain item order σ ;
- 2 Extract allocation strategy, number of initial bins b_0 , and item relocation probability p from the last three elements of χ ;
- 3 **if** $strategy = 1$ **then**
- 4 Apply the semi-greedy construction (Algorithm 2) using σ ;
- 5 **else**
- 6 **if** $strategy = 2$ and $b_0 > 0$ **then**
- 7 Open b_0 bins of type m^* ;
- 8 Assign the first b_0 items in σ to these bins (one per bin);
- 9 **for** each unassigned item i in σ with $\ell_i^+ > 0$ **do**
- 10 Assign i to the best feasible bin (Algorithm 3);
- 11 **for** each remaining unassigned item i in σ **do**
- 12 Assign i to the best feasible bin (Algorithm 3);
- 13 Apply post-processing (bin type replacement, bin merging);
- 14 Apply local search with probability p ;
- 15 **return** Set of opened bins B with its assigned items

5.4. Post-processing

Post-processing is applied both to initial solutions and to every decoded solution to improve cost efficiency while maintaining feasibility. Two main procedures are performed.

The *bin type replacement* checks if any bin can be replaced with a smallest-cost bin type that remains feasible for its assigned items.

After, the *bin merging* verifies for every pair of opened bins if the combined contents of two bins fit within a feasible bin type whose cost is lower than the total cost of the original bins. Thus, the items are reassigned to that bin, and the redundant bin is removed.

The following algorithms describe these procedures in detail.

Algorithm 5: Bin Type Replacement

Input: Set of opened bins B

```
1 for each bin  $b \in B$  do
2   for each bin type  $m$  inferior than current type of  $b$  do
3     if all items in  $b$  fit in bin type  $m$  then
4       Update  $b$  to type  $m$ ;
5       break;
6 return Updated  $B$ 
```

Algorithm 6: Bin Merging

Input: Set of opened bins B

```
1 Initialize set  $T$  to track merged bins;
2 for each bin  $b_a \in B$  do
3   if  $b_a \in T$  then
4     continue
5   for each bin  $b_b \in B$  do
6     if  $b_b \in T$  then
7       continue
8      $c' \leftarrow$  cost of  $b_a + b_b$  bin types;
9     for each bin type  $m$  with cost  $c_m \leq c'$  do
10      if all items in  $b_a \cup b_b$  fit in bin type  $m$  then
11        Create new bin  $b_{\text{new}}$  in  $B$  of type  $m$ ;
12        Move items from  $b_a$  and  $b_b$  to  $b_{\text{new}}$ ;
13        Add  $b_a$  and  $b_b$  to  $T$ ;
14        break (only merge each pair once);
15      if  $b_a$  is in  $T$  then
16        break
17 return Updated  $B$ 
```

5.5. Local Search on the Problem Domain

After the initial solution construction and post-processing, an item relocation local search is applied to further improve solution quality. This procedure iteratively attempts to move items between bins to reduce the total cost, considering both bin costs and link penalties. In each iteration, candidate items for relocation are selected with a given probability. These candidates are sorted and prioritized by their external link cost (the sum of penalties for being separated from linked items in other bins), the cost of

their current bin, and the bin's size. This prioritization focuses the search on items whose relocation is most likely to yield a cost reduction. For each candidate item, the algorithm evaluates all other opened bins to find a feasible destination and simulate a move. If it results in a lower total cost, the move is accepted and the solution is updated. This process continues until no further improvements are found or a maximum number of iterations is reached. After the relocation phase, the bin merging procedure is reapplied to further reduce costs. The Algorithm 7 describes the item relocation process.

Algorithm 7: Item Relocation Local Search

Input: Set of opened bins B , probability p , max iterations t_{max}

```

1 for  $t = 1$  to  $t_{max}$  and improved is True do
2   Set improved  $\leftarrow$  false;  $t++$ ;
3   Initialize candidate list  $\mathcal{C}$ ;
4   for each bin  $b$  in  $B$  do
5     Draw  $r \sim \text{Uniform}(0, 1)$ ;
6     if  $r \leq p$  then
7       for each item  $i$  in  $b$  do
8         Add  $(i, b)$  to  $\mathcal{C}$ ;
9   Sort  $\mathcal{C}$  descending by items  $w_i^+$ , then bin type cost, then ascending by
    number of items in  $b$ ;
10  for each candidate  $(i, b)$  in  $\mathcal{C}$  do
11    for each other bin  $b'$  in  $B$  do
12      if  $i$  fits in  $b'$  then
13        Simulate moving the item  $i$  from bin  $b$  to  $b'$ ;
14        Apply bin type reduction to  $b$  and  $b'$ ;
15        Compute  $\Delta\text{objective}$  with new bins cost and link changes;
16        if  $\Delta\text{objective} < 0$  then
17          Accept move: update  $B$ ;
18          Set improved  $\leftarrow$  true;
19          break (first-improvement);
20    if improved then
21      break
22  if not improved then
23    break
24  Apply bin merging (Algorithm 6);
25 return Updated  $B$ 

```

5.6. Objective Function Tie Breakers

To guide the search toward more desirable solutions when objective function values are equal, three tie-breaking rules are applied in sequence. First, solutions that use bins with lower relative capacity utilization are preferred, promoting flexibility for accommodating future items. Second, among bins of equal utilization, those offering a better cost-to-capacity ratio are favored to reduce overall costs. Finally, solutions that resolve a greater link cost, thereby placing more highly connected items together, are prioritized, as this improves both feasibility and cost-effectiveness in the long term.

6. Computational Experiments

The set of 96 benchmark instances for the QMC-VSBPP used in this study were proposed by Meng et al. (2022). Each instance defines a set of items with multidimensional weights, inter-item costs, and a collection of bin types with distinct capacities and fixed costs. The instances differ in the number of items ($n = 25, 50, 100, 200$), bin types ($m = 10, 20, 50$), weight and capacity dimensionality ($d = 3, 5$), and cost function (B1 - linear, B2 - convex, B3 - concave, and B4 - mixed). In this study, we adopt the same instances to ensure fair comparison and reproducibility.

The algorithms were implemented in C++ and executed on an Intel Core Ultra 9 185H 5.1 GHz processor with 64 GB of RAM.

Exact solutions for the QMC-VSBPP, using both the original and linearized models, were obtained with the Gurobi 12.0.2 solver (Gurobi Optimization, LLC, 2024), with a time limit of 3600 seconds per instance.

For the approximated approach using the RKO-ACO, each instance was run 30 times with the following configuration: best improvement local search, Q-learning for online parameter tuning, and 16 threads running the ACO metaheuristic. Time limits were set to 200 seconds for instances with $n = \{25, 50\}$, 400 seconds for $n = 100$, and 600 seconds for $n = 200$, with a restart triggered at half of the maximum execution time. The solution pool size was set to 10.

ACO parameters were tuned in preliminary experiments where the archive size was tested with values $\{30, 40, 50, 60, 80, 100\}$. During execution, parameters were dynamically adjusted within the following ranges: archive size $\{25, 30\}$ for $n = \{25, 50\}$ and $\{55, 60\}$ for $n = \{100, 200\}$, number of ants $\{2, 3\}$, $q \in \{0.0001, 0.001, 0.1, 0.3\}$, and $\xi \in \{0.80, 0.85\}$.

To evaluate the quality of solutions, three commonly used metrics are employed. The Best Relative Percentage Deviation (BRPD) measures the deviation of the best solution obtained (Best) from the Best Known Solution (BKS) and is calculated as:

$$\text{BRPD} = \frac{\text{Best} - \text{BKS}}{\text{Best}} \times 100\% \quad (19)$$

where the BKS is the minimum best solution reported either by Gurobi or in the literature.

The Average Relative Percentage Deviation (ARPD) captures the average deviation across N independent runs and measures the performance consistency of the algorithm:

$$\text{ARPD} = \frac{1}{N} \sum_{i=0}^N \frac{S_i - \text{BKS}}{S_i} \times 100\% \quad (20)$$

The Gap quantifies the difference between the best solution obtained and a lower bound (LB) found by the Gurobi using the proposed models, offering insight into the solution’s proximity to optimality:

$$\text{Gap} = \frac{\text{Best} - \text{LB}}{\text{Best}} \times 100\% \quad (21)$$

Detailed results are reported in the GitHub repository², containing the instance settings, the best solution found by Gurobi and RKO, the time to the best solution, lower bound, Gap, ARPD, and BRPD values.

6.1. Exact Results

Both the original and linearized formulations of the QMC-VSBPP were solved using Gurobi. Figure 2 shows the resulting gap values.

The results show that the original model is effective at finding high-quality solutions, particularly for larger instances, achieving superior solutions in 62% of all cases. In contrast, the linearized model excels at providing tighter lower bounds and finding optimality in smaller instances (4 optimal solutions in $n = 25$), outperforming the original model in 67% of the instances in terms of lower bound quality.

²<https://github.com/nataliaalves03/RKO-ACO>

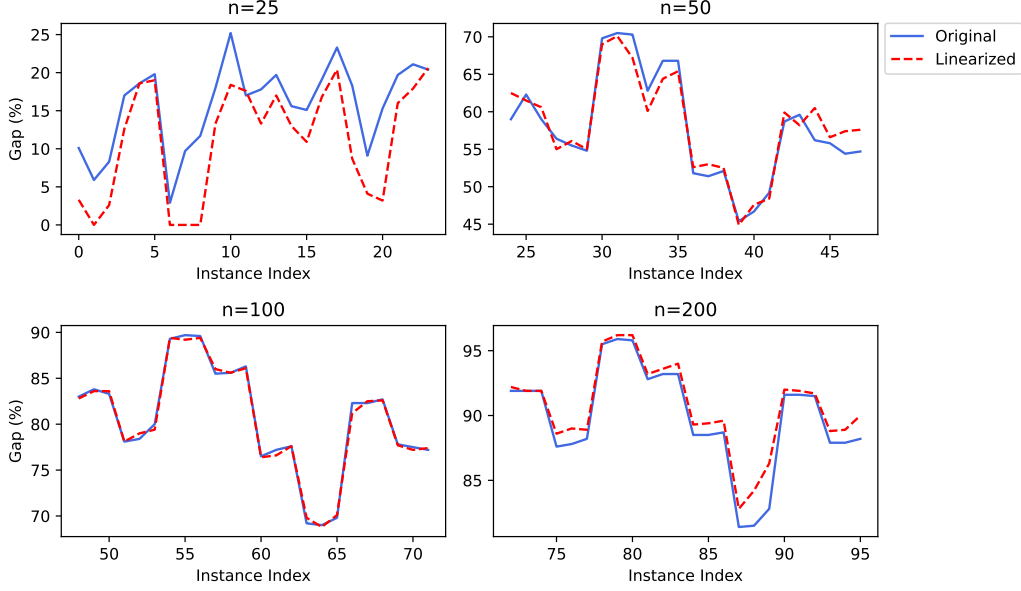


Figure 2: Percentage gap from Gurobi results with original and linearized models

Nevertheless, for larger instances, both models exhibit substantial optimality gaps, averaging around 90% with a one-hour execution limit, highlighting the computational difficulty of the QMC-VSBPP and motivating the adoption of advanced metaheuristics.

6.2. RKO-ACO Results

The RKO-ACO solutions, also referred to as RKO, were compared with the VNS results reported by Meng et al. (2022) and with the best solutions obtained by Gurobi for both the original and linearized models.

Figure 3 presents the relative gaps of RKO, VNS, and Gurobi to the best-known gap obtained by either VNS or Gurobi. For each method h and instance i , the relative gap ($\Delta\text{Gap}_{h,i}$) is defined as:

$$\Delta\text{Gap}_{h,i} = \text{Gap}_{h,i} - \text{BKG}_i \quad (22)$$

where BKG_i represents the best-known gap obtained by VNS or Gurobi on instance i . The BKG value is taken as the reference and normalized to zero in the Figure 3. Positive $\Delta\text{Gap}_{h,i}$ values indicate that a method performed worse than the best of these two baselines, while negative values denote that the method outperformed both.

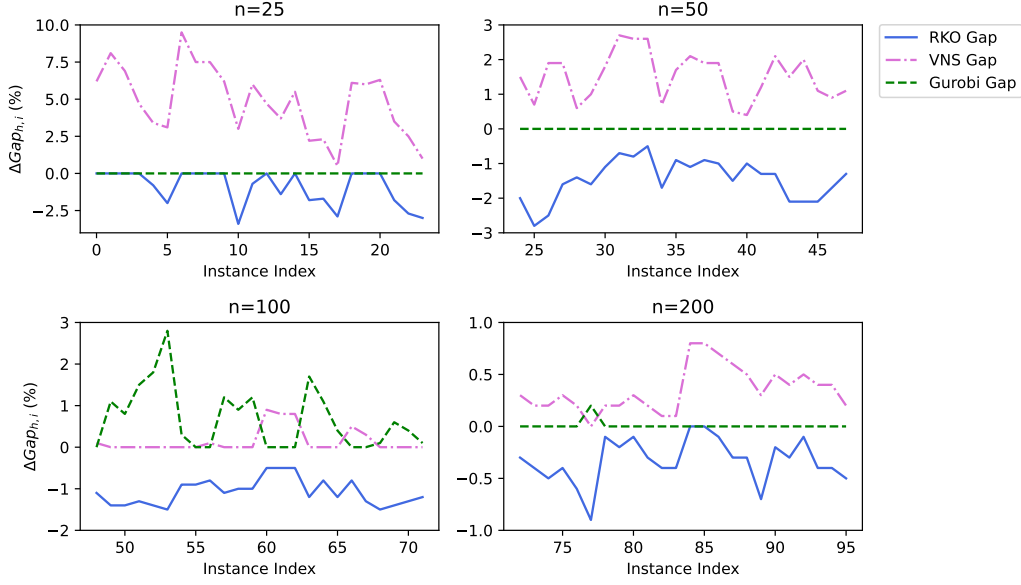


Figure 3: Relative gaps of RKO, VNS, and Gurobi to the best-known gap obtained by either VNS or Gurobi

According to Figure 3, the VNS shows the weakest performance, with relative gaps of up to 9.5% in $n = 25$ instances. This method achieved the BKS in only 19% of instances, most of them with $n = 100$.

Besides, Gurobi achieves the BKS in 100% of the instances with $n = \{25, 50\}$, in 33% of the $n = 100$ instances, and in 96% of the $n = 200$ instances, surpassing VNS at the cost of significantly greater computational time.

The proposed RKO demonstrates strong performance across all instance sizes, consistently matching or improving upon the best-known solutions. For $n = 25$, it attains the BKS in 13 out of 24 instances and establishes new best results in the remaining 11. For $n = 50$ and $n = 100$, RKO achieves new best solutions in all 48 instances, surpassing both exact and heuristic methods from the literature. In the largest case, $n = 200$, it matches the BKS in 2 instances and improves upon it in 22, further confirming its robustness.

Figure 4 presents boxplots comparing the VNS's time to reach its reported best solution with the time required by RKO, using a single thread, to reach that same solution value. To ensure a fair comparison, the VNS times were adjusted by a factor of 2.2 to compensate for the performance difference

relative to the hardware used for the RKO runs. For small instances ($n = \{25, 50\}$), both methods reach the target in only a few seconds. However, for instances with $n = 100$, VNS requires approximately 7 s, 2.3 times longer than RKO. For $n = 200$, RKO reaches the VNS best solution in an average of 0.7 s, whereas VNS ranges between 33 s and 92 s.

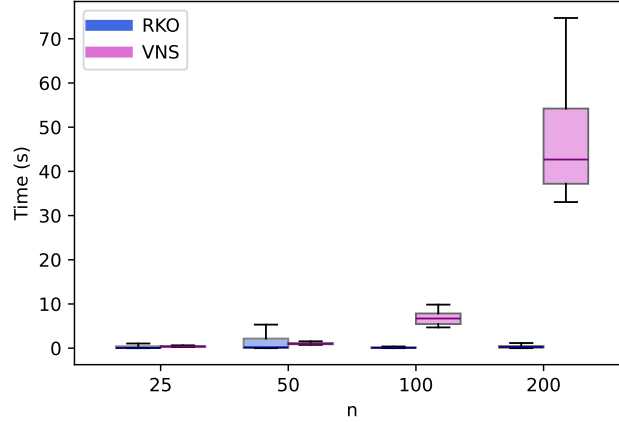


Figure 4: Time required by RKO to reach the best solution reported by VNS

In Figure 5, the RKO’s ARPD and BRPD values are reported for each instance set. A larger difference between ARPD and BRPD reflects greater variance across runs, while negative values indicate improvements over the BKS.

As shown in Figure 5, the average difference between ARPD and BRPD is 0.8% with a standard deviation of 0.6%. The largest differences occur for instances with $n = 50$, where the ARPD exceeds the BRPD by up to 3% and 1.2% on average. In contrast, for instances with $n = 200$, the maximum difference is 1% and the average is 0.5%. Overall, the ARPD remains below or equal to zero in 93% of the instances, and the BRPD remains below or equal to zero in 100% of the cases, confirming that the algorithm consistently matches or improves upon the BKS.

The time analysis of RKO per thread configuration, where each thread runs an independent instance of the ACO algorithm, is presented in Figure 6. For each value of n , the best-performing instance was selected to measure the computational time (in seconds) required to reach solution targets within 2% to 5% of the best RKO result for that instance. With a single thread, the time required exceeds 11 s. Using four threads leads to a substantial reduction,

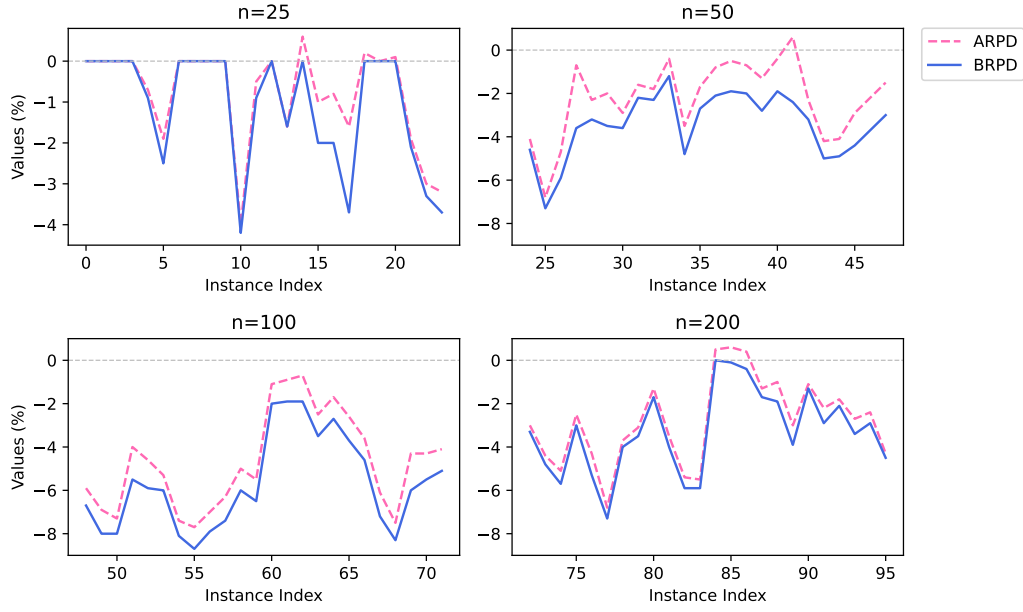


Figure 5: RKO's ARPD and BRPD values

with an average time of 5 s. With eight threads, the target solutions are obtained in 7.5 s for instances with $n = 50$, and approximately 2 s for the remaining instances.

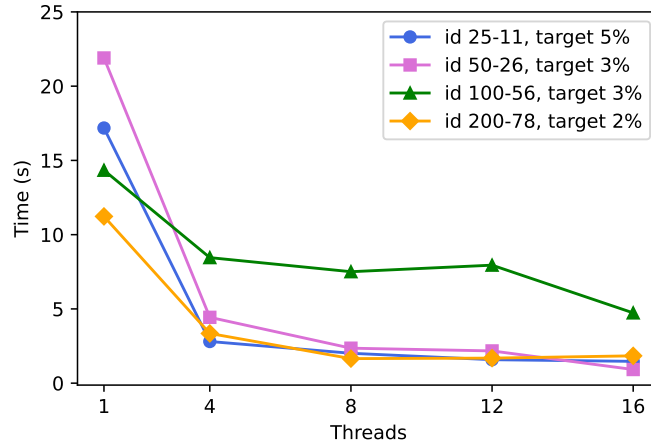


Figure 6: RKO's time to target solution by number of threads

In addition, we evaluate the RKO’s ability to find high-quality solutions by verifying the average gap per instance setting: n (number of items), m (number of bins), d (weight and capacity dimensions), and c (cost function: B1 - linear, B2 - concave, B3 - convex, or B4 - mixed), as presented in Figure 7.

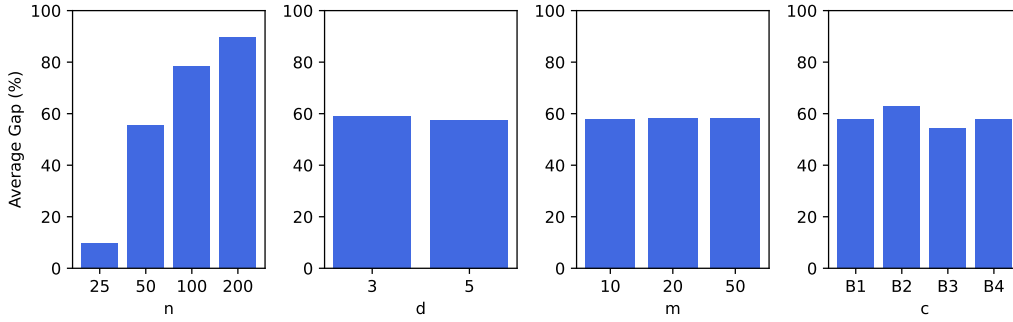


Figure 7: Average gap from RKO by instance settings: n , d , m , and c

As shown in Figure 7, the main factor contributing to the solution gap is the number of items, indicating that the gap is directly proportional to n . This is supported by a Pearson correlation of 0.83 between gap and n values. Although the cost function B2 (concave) appears to impact the gap, an ANOVA test indicates that this difference is not statistically significant (p-value 0.83).

6.3. Statistical Analysis

Since the Shapiro–Wilk test indicated non-normality for all paired differences (p-value $< 10^{-5}$), nonparametric tests were applied. The Friedman test detected significant performance differences among the methods (p-value 5.4×10^{-34}). Pairwise Wilcoxon tests confirmed that RKO significantly outperforms both VNS (p-value $= 8.8 \times 10^{-18}$) and Gurobi (p-value $= 2.7 \times 10^{-15}$).

7. Conclusion

This study advances the state-of-the-art for the Quadratic Multiple Constraints Variable-Sized Bin Packing Problem (QMC-VSBPP) by integrating exact and metaheuristic solution approaches.

First, we proposed a linearization of the original formulation, which contained quadratic terms. This linearized model facilitated the use of exact solvers and produced tighter lower bounds, providing the first reported lower-bound results for the QMC-VSBPP. Although optimality gaps remain significant for larger instances, the linearized model establishes a stronger foundation for exact optimization.

The proposed RKO-ACO metaheuristic demonstrated robust performance across all benchmark instances, consistently achieving or surpassing best-known solutions from both exact and heuristic methods from the literature. Notably, the RKO-ACO improved 95 of 96 benchmark instances, confirming its effectiveness in handling the combinatorial complexity of the problem. Analysis of ARPD and BRPD metrics further validated the stability of the approach across independent runs.

The results of this study confirm the research hypotheses introduced in Section 1, as detailed below.

(1) Effectiveness of linearization. The linearized formulation of the QMC-VSBPP consistently resulted in tighter lower bounds compared to the original quadratic model. Also, this simplified formulation could obtain four optimal solutions in small instances. However, both formulations exhibited substantial optimality gaps for larger instances, reflecting the intrinsic combinatorial complexity of the problem and motivating the need for complementary metaheuristic strategies.

(2) Performance of RKO-ACO. The RKO-ACO algorithm demonstrated robust performance across all tested instances, outperforming best-known solutions from both exact and traditional methods from the literature. Analysis of ARPD and BRPD metrics across independent runs confirmed the reliability and stability of the approach.

(3) Comparison with benchmarks. The exact Gurobi solutions already outperform the VNS results reported by Meng et al. (2022) in most of the instances. However, the proposed RKO-ACO further improves upon both, achieving the best-known solution or establishing new upper bounds for all $n = 25$ to $n = 100$ cases, and 92% of instances with $n = 200$. These results establish updated benchmark values for future studies.

Overall, the combination of exact and metaheuristic methods provides complementary benefits: the model linearization improves lower bound estimation and simplifies exact optimization approaches, while RKO-ACO effectively explores the solution space, obtaining high-quality solutions with computational efficiency. These findings demonstrate that adaptive meta-

heuristics, particularly when integrated with continuous-domain representations, are highly effective for solving complex quadratic packing problems.

Future work may explore hybrid exact–heuristic methods and exploit problem-specific structures to further enhance scalability.

Author contributions: CRediT

Natalia Alves Santos: Conceptualization, Methodology, Software, Visualization, Writing – Original Draft. **Marlon Jeske:** Methodology, Validation, Writing – Review and Editing, Funding acquisition. **Antônio Augusto Chaves:** Conceptualization, Supervision, Resources, Software, Validation, Writing – Review and Editing, Funding acquisition.

Funding sources

This study was financed in part by the Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES) - Finance Code 001. Marlon Jeske was supported by the São Paulo Research Foundation (FAPESP) under grant 2025/00386-3. Antonio Chaves was supported by the São Paulo Research Foundation (FAPESP) under grants 2022/05803-3 and 2024/08848-3, and the National Council for Scientific and Technological Development (CNPq) under grant 305557/2024-68.

Data and Code Availability

All datasets used in this study are publicly available. The source code for the RKO-ACO algorithm, along with instances and scripts to reproduce the experiments, is available at <https://github.com/nataliaalves03/RKO-ACO>.

References

- Ali, A.I., Keedwell, E., Helal, A., 2024. A differential pheromone grouping ant colony optimization algorithm for the 1-D bin packing problem, in: Proc. Genet. Evol. Comput. Conf., Association for Computing Machinery, New York, NY, USA. p. 1463–1469. doi:10.1145/3638529.3654074.
- Bahk, E.N., Ekici, A., Elyasi, M., Örsan Özener, O., 2025. Open-end bin packing problem with conflicts. Comput. Ind. Eng. 208, 111293. doi:<https://doi.org/10.1016/j.cie.2025.111293>.

- Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Comput.* 6, 154–160.
- Chaves, A.A., Resende, M.G.C., Schuetz, M.J.A., Brubaker, J.K., Katzgraber, H.G., de Arruda, E.F., Silva, R.M.A., 2025. A random-key optimizer for combinatorial optimization. *J. Heuristics* 31. doi:[10.1007/s10732-025-09568-z](https://doi.org/10.1007/s10732-025-09568-z).
- Chaves, A.A., Resende, M.G.C., Silva, R.M.A., 2024. A random-key GRASP for combinatorial optimization. *J. Nonlinear Var. Anal.* 8, 855–881. doi:[10.23952/jnva.8.2024.6.03](https://doi.org/10.23952/jnva.8.2024.6.03).
- Ekici, A., 2021. Bin packing problem with conflicts and item fragmentation. *Comput. Oper. Res.* 126, 105113. doi:<https://doi.org/10.1016/j.cor.2020.105113>.
- Ekici, A., 2022. Variable-sized bin packing problem with conflicts and item fragmentation. *Comput. Ind. Eng.* 163, 107844. doi:<https://doi.org/10.1016/j.cie.2021.107844>.
- Ekici, A., 2023. A large neighborhood search algorithm and lower bounds for the variable-sized bin packing problem with conflicts. *Eur. J. Oper. Res.* 308, 1007–1020. doi:<https://doi.org/10.1016/j.ejor.2022.12.042>.
- El-Ashmawi, W.H., Elminaam, D.S.A., 2019. A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem. *Appl. Soft Comput.* 82, 105565. doi:<https://doi.org/10.1016/j.asoc.2019.105565>.
- Epstein, L., Favrholdt, L.M., Levin, A., 2011. Online variable-sized bin packing with conflicts. *Discrete Optim.* 8, 333–343. doi:<https://doi.org/10.1016/j.disopt.2010.11.001>.
- Fleszar, K., 2022. A MILP model and two heuristics for the bin packing problem with conflicts and item fragmentation. *Eur. J. Oper. Res.* 303, 37–53. doi:<https://doi.org/10.1016/j.ejor.2022.02.014>.
- Gendreau, M., Laporte, G., Semet, F., 2004. Heuristics and lower bounds for the bin packing problem with conflicts. *Comput. Oper. Res.* 31, 347–358. doi:[https://doi.org/10.1016/S0305-0548\(02\)00195-8](https://doi.org/10.1016/S0305-0548(02)00195-8).

- Gurobi Optimization, LLC, 2024. Gurobi Optimizer Reference Manual. URL: <https://www.gurobi.com>.
- Haouari, M., Serairi, M., 2009. Heuristics for the variable sized bin-packing problem. *Comput. Oper. Res.* 36, 2877–2884. doi:<https://doi.org/10.1016/j.cor.2008.12.016>.
- Hemmelmayr, V., Schmid, V., Blum, C., 2012. Variable neighbourhood search for the variable sized bin packing problem. *Comput. Oper. Res.* 39, 1097–1108. doi:<https://doi.org/10.1016/j.cor.2011.07.003>.
- Khanafer, A., Clautiaux, F., Hanafi, S., Talbi, E.G., 2012. The min-conflict packing problem. *Comput. Oper. Res.* 39, 2122–2132. doi:<https://doi.org/10.1016/j.cor.2011.10.021>.
- Kucukyilmaz, T., Kiziloğlu, H.E., 2018. Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Comput. Ind. Eng.* 125, 157–170. doi:<https://doi.org/10.1016/j.cie.2018.08.021>.
- Levine, J., Ducatelle, F., 2004. Ant colony optimization and local search for bin packing and cutting stock problems. *J. Oper. Res. Soc.* 55, 705–716. doi:[10.1057/palgrave.jors.2601771](https://doi.org/10.1057/palgrave.jors.2601771).
- Liu, D., Tan, K., Huang, S., Goh, C., Ho, W., 2008. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *Eur. J. Oper. Res.* 190, 357–382. doi:<https://doi.org/10.1016/j.ejor.2007.06.032>.
- Meng, F., Cao, B., Chu, D., Ji, Q., Zhou, X., 2022. Variable neighborhood search for quadratic multiple constraint variable sized bin-packing problem. *Comput. Oper. Res.* 143, 105803. doi:<https://doi.org/10.1016/j.cor.2022.105803>.
- Muritiba, A.E.F., Iori, M., Malaguti, E., Toth, P., 2010. Algorithms for the bin packing problem with conflicts. *J. Comput.* 22, 401–415. doi:[10.1287/ijoc.1090.0355](https://doi.org/10.1287/ijoc.1090.0355).
- Santos, L.F.M., Iwayama, R.S., Cavalcanti, L.B., Turi, L.M., de Souza Morais, F.E., Mormilho, G., Cunha, C.B., 2019. A

- variable neighborhood search algorithm for the bin packing problem with compatible categories. *Expert Syst. Appl.* 124, 209–225. doi:<https://doi.org/10.1016/j.eswa.2019.01.052>.
- Schuetz, M.J., Brubaker, J.K., Montagu, H., van Dijk, Y., Klepsch, J., Ross, P., Luckow, A., Resende, M.G., Katzgraber, H.G., 2022. Optimization of robot-trajectory planning with nature-inspired and hybrid quantum algorithms. *Phys. Rev. Appl.* 18, 054045. doi:[10.1103/PhysRevApplied.18.054045](https://doi.org/10.1103/PhysRevApplied.18.054045).
- Silva, E.M., Chaves, A.A., de Araujo, S.A., Jans, R., 2025. Random-key optimizer with reinforcement learning for the capacitated multi-period cutting stock problem with setup cost. *Comput. Oper. Res.* 183, 107159. doi:<https://doi.org/10.1016/j.cor.2025.107159>.
- Socha, K., Dorigo, M., 2008. Ant colony optimization for continuous domains. *Eur. J. Oper. Res.* 185, 1155–1173. doi:<https://doi.org/10.1016/j.ejor.2006.06.046>.
- Stawowy, A., 2008. Evolutionary based heuristic for bin packing problem. *Comput. Ind. Eng.* 55, 465–474. doi:<https://doi.org/10.1016/j.cie.2008.01.007>.
- Vieira, B.S., Silva, E.M., Chaves, A.A., 2025. Random-key algorithms for optimizing integrated operating room scheduling. *Appl. Soft Comput.* 180, 113368. doi:<https://doi.org/10.1016/j.asoc.2025.113368>.