Hyper-heuristics for the Bin Packing Problem

:

SEMCHEDDINE Ikram , TAIB Salma , SEKKAI Maria ,HEDDADJI Nourelimane ,SOLTANI Meriem,ZERROUKI Selma ,AREF Zeyneb

Supervized by : *BESSEDIK Malika & KECHID Amine*

Abstract:

.

The Bin Packing Problem (BPP) is a complex NP-hard optimization task that involves packing items of varying sizes into the minimum number of bins with fixed capacity. Traditional heuristics and metaheuristics often have limitations in efficiency and solution quality. This paper introduces a novel hyper-heuristic approach to improve BPP solutions.

We review existing heuristic and metaheuristic methods for BPP, highlighting their limitations, and discuss the motivation for using hyper-heuristics. Our proposed method features a two-level architecture that dynamically selects the best heuristics for packing.

The framework includes a mathematical formulation of BPP, solution encoding, and detailed steps of the hyper-heuristic approach. Experimental results show our method outperforms traditional approaches in both solution quality and computational efficiency. Comparative analysis further emphasizes the advantages of our hyper-heuristic method, contributing to advancements in solving NP-hard optimization problems..

Keywords: Bin Packing Problem, NP-hard, hyper-heuristics, metaheuristics, computational efficiency

1 Introduction

The Bin Packing Problem (BPP) stands as a cornerstone in optimization, addressing the strategic allocation of items into containers while minimizing the number of containers used. With each container possessing a fixed capacity, the BPP entails a meticulous arrangement of items to optimize spatial utilization.

Formally, within the BPP framework, a collection of items, each characterized by varying sizes, necessitates distribution into containers of pre-defined capacities. The primary objective is to devise an allocation strategy that reduces the total count of containers required. This optimization endeavor mandates that the cumulative size of items within each container remains within its capacity limit.

Given its classification as NP-hard, the BPP commands considerable scholarly and practical interest. Researchers and practitioners alike have endeavored to devise efficient heuristic and metaheuristic methodologies to tackle this complexity, offering timely approximate solutions that align with real-world constraints.

In essence, the Bin Packing Problem embodies a foundational optimization quandary, resonating across diverse sectors including logistics, manufacturing, and resource allocation. Its resolution through innovative algorithmic paradigms underscores its pivotal role in addressing intricate packing and allocation challenges encountered in practical scenarios.

1.1 Related Works: Literature Review on Bin Packing Problem

The Bin Packing Problem (BPP) has garnered significant attention in both academic research and practical applications, leading to a plethora of studies aimed at devising effective solution approaches. This literature review focuses on works that utilize heuristics and/or metaheuristics for addressing the BPP, while also highlighting their inherent limitations.

Numerous researchers have explored heuristic-based approaches to tackle the BPP due to its NP-hard nature, which makes finding optimal solutions computationally infeasible for large problem instances. Classic heuristics such as First Fit, Best Fit, and Next Fit have been extensively studied and serve as baseline algorithms for comparison in many studies. These heuristics often provide fast and simple solutions, but they may struggle with achieving optimality, especially when faced with complex problem instances or stringent constraints.

In addition to traditional heuristics, metaheuristic methods have emerged as powerful tools for addressing the BPP. Metaheuristics, such as genetic algorithms, simulated annealing, tabu search, and ant colony optimization, offer flexible and adaptable search strategies that can explore the solution space more effectively than simple heuristics. However, despite their versatility, metaheuristics are not without drawbacks. They often require fine-tuning of parameters, are computationally intensive, and may struggle to escape local optima in high-dimensional solution spaces.

While heuristics and metaheuristics have proven valuable in tackling the BPP, it is crucial to recognize their limitations. These approaches may not always guarantee optimal solutions, and their performance can vary significantly depending on problem characteristics such as item distributions, container capacities, and problem size. Moreover, the computational resources required to implement some metaheuristic algorithms can be prohibitive for real-time or large-scale applications.

In summary, while heuristics and metaheuristics offer practical and efficient means of addressing the Bin Packing Problem, it is essential to acknowledge their limitations. Future research endeavors should focus on refining existing methods, developing hybrid approaches, and exploring alternative solution paradigms to further advance the state-of-the-art in BPP optimization.

1.2 Why move towards hyper-heuristics?

The Bin Packing Problem (BPP) presents inherent complexity, demanding innovative approaches for efficient solution finding. Hyper-heuristics emerge as a promising avenue due to their adaptive nature and ability to dynamically select or generate heuristics, offering potential advantages over traditional heuristic and metaheuristic methods.

Various hyper-heuristic approaches exist, each with unique characteristics tailored to address different optimization challenges. These include selection hyper-heuristics, which choose from a predefined set of low-level heuristics based on problem features, and generation hyper-heuristics, which iteratively generate new heuristics or heuristic combinations to explore the solution space.

Our decision to opt for a hyper-heuristic approach stems from its potential to offer a versatile and adaptable solution strategy for the BPP. By leveraging a range of low-level heuristics and dynamically adapting their selection based on problem dynamics, hyper-heuristics have the capacity to navigate complex solution landscapes more effectively than static methods. This adaptability aligns well with the inherent uncertainty and variability present in real-world BPP instances, making hyper-heuristics a compelling choice for addressing this optimization challenge.

1.3 *Literature Review on the Application of Hyper-heuristics for NP-Hard Optimization Problems*

This is the first comprehensive literature review journal article on hyper-heuristic ap-proaches. However, a number of introductory, tutorial and review book chapters have been published. The first introductory and overview article appeared in 2003 (Burke et al, 2003a), where the authors introduce the idea of hyper-heuristics, and stress their main objective; namely, to raise the level of generality at which optimisation systems can operate. The chapter also gives a brief history of the area and discusses in detail some of the latest work published at the time. A tutorial article is later published by Ross (2005), which not only gives useful guidelines for implementing a hyper-heuristic approach; but also discusses a number of relevant research issues and identifies promis-ing application domains. The chapter by Chakhlevitch and Cowling (2008) provides a review of recent the developments in hyper-heuristics. The authors classify and review hyperheuristicapproaches into the following four categories: based on the random choice of low level heuristics, greedy and peckish, metaheuristic-based, and those employing learning mechanisms to manage low level heuristics. Their chapter does not re-view the literature on genetic programming approaches, nor includes a detailed account of the origins and early hyper-heuristic approaches. Finally, a very recent overview and tutorial chapter (Burke et al, 2009b) discusses the more recent approach that aims to generate new heuristics from a set of potential heuristic components, in which Genetic Programming is the most widely used methodology. The chapter includes a detailed description of the steps needed to apply this approach, some representative case stud-ies, a brief literature review of related work, and a discussion of relevant issues of this class of hyper-heuristic.

1.4 *Presentation of the Proposed Approach : Selective Perturbative Hyper-heuristic*

To address the challenging Bin Packing Problem, we propose an innovative approach based on selective hyper-heuristics. Our approach stands out for its disruptive selection mechanism, which chooses among several low-level heuristics such as tabu search, hill climbing, and simulated annealing. The selection of these heuristics is guided by multiple criteria. For instance, hill climbing is favored for its speed compared to others, albeit prone to converging towards local optima. Conversely, simulated annealing, by occasionally accepting degrading solutions, aids in avoiding such local optima. Additionally, tabu search prevents movements towards previously visited solutions, facilitating exploration of the solution space. The uniqueness of our approach lies in the amalgamation of these diverse strategies, aiming to strike a balance between exploration and exploitation for high-quality solutions. At the higher level of our approach, we have devised a strategy for randomly selecting permutations, enabling efficient exploration of the solution space. We have evaluated various random selection strategies, including random simple, random gradient, random permutation, random permutation gradient, and greedy, for their ability to generate high-quality solutions while ensuring effective exploration of the solution space

2 Problem Formulation

2.1 *Definition of the Problem:*

The Bin Packing Problem (BPP) involves allocating a set of items of different sizes into a minimal number of bins, each with a fixed capacity. The objective is to minimize the number of bins used while ensuring that the total size of items allocated to each bin does not exceed its capacity.

2.2 *Mathematical Formulation:*

Let n denote the number of items to be packed, w_i represent the size of item i, and c denote the capacity of each bin. We aim to assign each item to a bin such that the total size of items in each bin does not exceed c.

Formally, let x_{ij} be a binary decision variable, where $x_{ij} = 1$ if item i is assigned to bin j, and $x_{ij} = 0$ otherwise. Additionally, let y_j be a binary variable indicating whether bin j is used (i.e., $y_j = 1$ if bin j is used, and $y_j = 0$ otherwise).

The mathematical formulation of the BPP can be expressed as follows:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{j=1}^{m} y_j \\
\text{Subject to} \quad & \sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in \{1, 2, ..., n\} \\
& \sum_{i=1}^{n} w_i \cdot x_{ij} \leq c \cdot y_j, \quad \forall j \in \{1, 2, ..., m\} \\
& x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, 2, ..., n\}, \forall j \in \{1, 2, ..., m\} \\
& y_j \in \{0, 1\}, \quad \forall j \in \{1, 2, ..., m\}
\end{aligned}$$

Where:

- n is the number of items.
- m is the number of bins.
- w_i is the size of item i.
- c is the capacity of each bin.
- x_{ij} is a binary decision variable indicating whether item i is assigned to bin j.
- y_j is a binary decision variable indicating whether bin j is used.
- The objective function minimizes the total number of bins used.
- The first constraint ensures that each item is assigned to exactly one bin.
- The second constraint ensures that the total size of items assigned to each bin does not exceed its capacity.

This mathematical formulation provides a foundation for modeling and solving the Bin Packing Problem using optimization techniques.

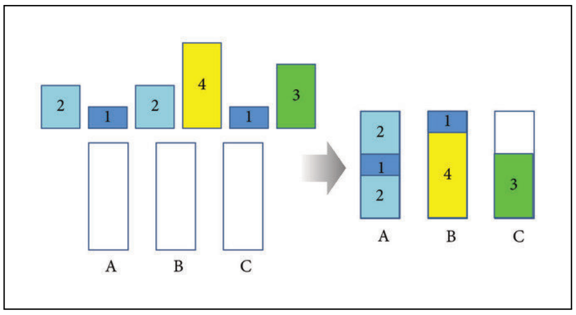


Fig. 1: Bin Packing Problem .

3 DESCRIPTION OF THE PROPOSED APPROACH

In this section, we elaborate on our solution while elucidating the high and low levels of the hyper-heuristic framework. We have opted for a selective perturbative hyper-heuristic to tackle the Bin Packing Problem (BPP).

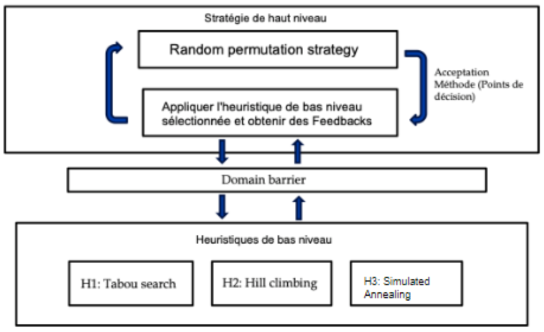

Stratégie de haut niveau
Random permutation strategy
Appliquer l'heuristique de bas niveau sélectionnée et obtenir des Feedbacks
Acceptation Méthode (Points de décision)
Domain barrier
Heuristiques de bas niveau
H1: Tabou search
H2: Hill climbing
H3: Simulated Annealing

Fig. 2: Diagram of the Proposed Hyperheuristic

3.1 *Low Level heuristics*

For the low level, we have selected three heuristics: tabu search, hill climbing, and simulated annealing.

3.1.1 *Tabu search:*

is a heuristic optimization method that guides the search process by maintaining a short-term memory of previously visited solutions. This memory, known as the "tabu list," records forbidden moves or solutions to prevent revisiting them in subsequent iterations. By avoiding revisits to previously explored solutions, tabu search promotes diversification in the search process, facilitating exploration of different regions of the solution space. This aids in overcoming local optima and enhancing the algorithm's ability to find better-quality solutions

3.1.2 *hill climbing:*

Hill climbing is a local search algorithm used in optimization problems like the Bin Packing Problem (BPP). It operates by iteratively improving a solution by making small incremental changes, moving towards higher values in the solution space, akin to climbing up a hill.

In hill climbing, the algorithm starts with an initial solution and evaluates its neighboring solutions. It then selects the best neighboring solution that improves upon the current one and moves to that solution. This process continues iteratively until no further improvements can be made, i.e., until a local maximum is reached.

One limitation of hill climbing is that it can get stuck in local optima, where no further improvements are possible without moving away from the current solution. Therefore, hill climbing may not always find the globally optimal solution, especially in complex solution spaces with multiple local optima.

3.1.3 *simulated annealing:*

is a probabilistic metaheuristic technique inspired by the annealing process in metallurgy. It is often applied to optimization problems like the Bin Packing Problem (BPP).

In simulated annealing, the algorithm starts with an initial solution and iteratively explores the solution space by making incremental changes. Unlike traditional optimization algorithms, simulated annealing allows for occasional acceptance of worse solutions to escape local optima. This acceptance is controlled by a temperature parameter, which gradually decreases over time.

At higher temperatures, the algorithm is more likely to accept worse solutions, allowing for more exploration of the solution space. As the temperature decreases, the algorithm becomes more selective and tends to converge towards better solutions. This controlled exploration-exploitation trade-off helps simulated annealing to effectively navigate complex solution landscapes and find near-optimal solutions for challenging optimization problems like the BPP.

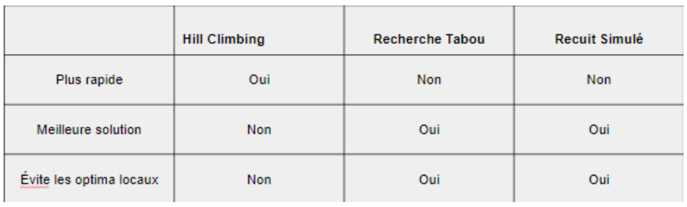
3.1.4 *Why We Choose Hill Climbing, Tabu Search, and Simulated Annealing:*

1. **Hill Climbing**: Hill climbing is straightforward and computationally efficient. It iteratively explores the neighboring solutions and moves to the best neighboring solution that improves upon the current one. While it may get stuck in local optima, it is relatively fast and can serve as a good starting point for more complex algorithms.

2. **Tabu Search**: Tabu search introduces a short-term memory mechanism, known as the tabu list, which prevents revisiting previously explored solutions. This promotes diversification in the search process, enabling the algorithm to escape local optima and explore new regions of the solution space. Tabu search is particularly effective in balancing exploration and exploitation, leading to better-quality solutions.

3. **Simulated Annealing**: Simulated annealing is a probabilistic algorithm that allows for occasional acceptance of worse solutions to escape local optima. It achieves this by gradually decreasing a temperature parameter, controlling the probability of accepting worse solutions as the algorithm progresses. Simulated annealing strikes a balance between exploration and exploitation, enabling effective navigation of complex solution landscapes and finding near-optimal solutions.

By integrating these three algorithms into our hyper-heuristic framework for the BPP, we aim to leverage their complementary strengths. Hill climbing provides efficiency and speed, tabu search enhances exploration and prevents premature convergence, while simulated annealing facilitates escape from local optima and promotes diversification in the search process. Together, they form a powerful combination that enhances our ability to find high-quality solutions for the BPP.



	Hill Climbing	Recherche Tabou	Recuit Simulé
Plus rapide	Oui	Non	Non
Meilleure solution	Non	Oui	Oui
Évite les optima locaux	Non	Oui	Oui

Fig. 3: Comparison of methods for the Bin Packing Problem.

3.2 *High Level heuristic*

At the high level, we have two essential strategies: selection strategy and acceptance strategy.

3.2.1 *Selection Strategy:*

determines how we choose between different low-level heuristics at each step of the algorithm. We consider various factors such as solution quality, computational efficiency, and diversity in exploration. We employ various selection strategies to choose between different low-level heuristics (LLHs) in our hyper-heuristic framework:

1. **Simple Random**: Selects a LLH randomly and applies it repetitively as long as it improves the current solution.

2. **Random Gradient**: Generates a random combination of LLHs and applies them successively in order.

3. **Random Permutation**: Similar to Random Gradient but each LLH in the sequence is applied repetitively until no further improvement is possible.

4. **Random Permutation Gradient**: Applies the LLHs exhaustively and selects the one that best improves the solution.

5. **Greedy**: Selects the LLH that improves the solution the most.

To choose between these selection strategies, we conduct a comparative study and present the results in the graph.

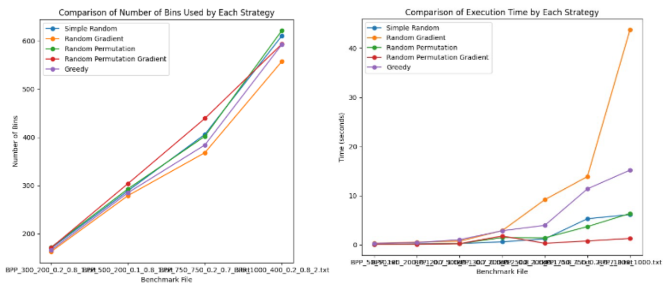


Fig. 4: Comparison Between Different Search Strategies Based on the Number of Returned Items and Execution Time

Observation:

• Analysis of the performance of different strategies for the Bin Packing Problem (BPP) reveals that the Random Permutation Gradient strategy demonstrates balanced and often superior performance in terms of both the number of bins used and execution time.
• This strategy strikes an effective compromise, generally outperforming Simple Random and Random Gradient strategies, and closely matching the Greedy strategy in terms of bin usage while maintaining reasonable execution times.

Explanation:

• The Random Permutation Gradient strategy combines the strengths of random permutation and gradient-based improvement mechanisms.
• By incorporating randomness, this strategy explores a wider solution space, helping find more optimal solutions for bin packing.
• The gradient component then fine-tunes these solutions to further reduce the number of bins required.
• This dual approach allows the Random Permutation Gradient strategy to adaptively balance exploration and exploitation, resulting in efficient bin utilization.
• Improved performance in terms of bin usage can be attributed to the strategy's ability to effectively rearrange items within bins to minimize wasted space.
• Meanwhile, its execution time remains manageable because gradient-based adjustments are typically less computationally intensive than exhaustive search methods, making it suitable for solving large and complex instances of the BPP.

Conclusion:

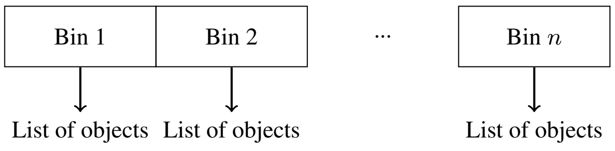
The Random Permutation Gradient strategy effectively balances bin usage and computational efficiency, surpassing the Greedy strategy for larger datasets. Further enhancements could solidify its position as a leading method for the Bin Packing Problem.

3.2.2 Acceptance Strategy:

The acceptance strategy governs how we decide whether to accept or reject a new solution generated by the selected low-level heuristic. This decision is typically based on the change in solution quality and a probability distribution function that allows for exploration of suboptimal solutions. for us we choose the solution that improves upon the current solution; otherwise, we retain the current solution

3.3 Encode solution

We have decided to represent our solution as follows: a list of bins where each bin has a list of objects it contains. This will be a list pointing to another list, and for each bin, when an object is added, a dynamic allocation is made in the list.



Example

We illustrate the encoding with an example where we have three bins and their respective objects:

1. **Bin 1**: Objects {A, B, C}
2. **Bin 2**: Objects {D, E}
3. **Bin 3**: Objects {F}

3.4 Functioning of a hyperheuristic

So, for the schema of hyper-heuristic, at the low level, we have domain heuristics: hill climbing, simulated annealing, tabu search. At the high level, we opt for selection, permutation, gradient strategies. For the acceptance criterion, we accept only solutions that improve

1. Choose heuristic: At each iteration, generate a random combination of low-level heuristics (LLH) and apply them successively in order. However, each heuristic in the sequence is applied repetitively until no further improvement is possible.

2. Evaluate solution: After choosing the heuristic, evaluate the solution. Apply the chosen heuristic to the bin packing problem and evaluate the solution (number of bins used).

3. Acceptance criterion: After evaluating the solution, if the new solution improves upon the current solution, keep it. Otherwise, retain the current solution.

4. Repeat: Continue repeating this process until a predefined number of iterations is reached or another termination criterion is met.

Algorithm 1 Hyper-heuristic Algorithm

```
1:  Initialize current solution $S$
2:  Initialize iteration counter $i \leftarrow 0$
3:  while $i$ not reached maximum iterations or termination condition not met do
4:      Generate random combination of low-level heuristics $LLH$
5:      for each heuristic $H$ in $LLH$ do
6:          repeat
7:              Apply $H$ to $S$
8:          until no further improvement possible
9:      end for
10:     Evaluate solution $S$
11:     if new solution improves upon current solution then
12:         Accept new solution $S$
13:     end if
14:     Increment $i$
15: end while
```

4 Tests and Results

As part of our study on combinatorial optimization, we conducted tests to evaluate the effectiveness of a solution using hyperheuristics to solve the bin packing problem.

The environment and characteristics of our machines were carefully selected to ensure efficient and accurate test execution, as well as in-depth analysis of the results obtained during the optimization of the bin packing problem.

Environment and Programming Languages: Python was chosen for its versatility and widespread use in optimization. Google Colab served as the execution platform, providing an integrated environment for writing, debugging, and executing Python code. Google Colab allocated 12 GB of RAM and 100 GB of disk space for our environment.

Libraries Used: We utilized **NumPy**, a powerful Python library for numerical computing, which offers efficient data structures and advanced functions for manipulating multidimensional arrays. Pandas was also employed for data manipulation and analysis, providing flexible DataFrames for handling datasets.

Platform Characteristics: Google Colab utilizes virtual machines with variable configurations based on needs, including adequate RAM to handle computation tasks and large datasets, as well as high-performance CPUs for efficient processing of complex calculations.

Using Python with NumPy and Pandas on Google Colab provides a flexible and powerful environment for data manipulation and optimization. This virtual platform leverages elastic compute and memory resources, thereby optimizing the performance of the planned tests.

4.1 Presentation of the instances used

To evaluate our hyperheuristic, we generated solutions on various instances from the Randomly_generated benchmark, which contains 3840 instances of varying sizes ranging from 50 to 1000 items. The filename format for these instances is

"BPP_items_capacity_min_max_num.txt", where:

- **items**: number of objects.
- **capacity**: bin capacity.
- **min**: minimum shape factor.
- **max**: maximum shape factor.
- **num**: instance identifier, the serial number of the instance generated with the aforementioned parameters. "The shape factor is the ratio between the smallest and largest dimensions of the objects."

These random instances allow us to test the robustness and generalization of our solution against diverse and unpredictable configurations. In addition to these random instances, we also generalized our solution on benchmarks established in the literature, such as instances from Falkenauer, specifically Falkenauer_U and Falkenauer_T, to validate our approach on standardized and recognized datasets.

Table 1 Instance caracteristics

Instance	items Number	bin capacity
BPP_50_50_0.1_0.7_7	50	50
BPP_50_400_0.2_0.8_4	50	400
BPP_100_750_0.2_0.7_0	100	750
BPP_200_1000_0.2_0.8_8	200	1000
BPP_300_50_0.1_0.7_0	300	50
BPP_500_50_0.1_0.7_2	500	50
BPP_750_500_0.2_0.8_8	750	500
BPP_750_750_0.2_0.8_9	750	750
BPP_750_1000_0.2_0.8_9	750	1000
BPP_1000_50_0.1_0.7_1	1000	50
BPP_1000_125_0.2_0.8_9	1000	125
BPP_1000_150_0.2_0.8_9	1000	150
BPP_1000_200_0.2_0.8_9	1000	200
BPP_1000_300_0.2_0.8_9	1000	300
BPP_1000_400_0.2_0.8_9	1000	400
BPP_1000_500_0.2_0.8_9	1000	500
BPP_1000_750_0.2_0.8_9	1000	750
BPP_1000_1000_0.1_0.7_0	1000	1000

The characteristics of the different instances used in our tests are presented in Table 1. This table includes the name of each instance, the number of items, and the bin capacity.

To demonstrate the effectiveness of our solution, we provide a detailed analysis of the performance of our hyperheuristic approach for the bin packing problem. We emphasize the algorithm's behavior, its evolution based on parameter values, and the input instances. We evaluated our method without prior training.

4.2 Solution's evaluation

The performance study we conducted aims to evaluate the effectiveness and efficiency of our hyperheuristic optimization method for the bin packing problem. This study allows us to measure the performance of our approach in various aspects, such as the quality of the obtained solutions, the required computation time, and resource consumption. It provides us with a precise overview of our method's capability to effectively solve the bin packing problem.

The execution results obtained for the 18 benchmarks, with a fixed number of iterations set to 100, are summarized in the table below

4.3 Benchmarks Solution

Table 2 Performance Comparison between our Hyperheuristic approach and the Branch and Bound method

Instance	Exact	Solution HH	T_ex Exact	T_ex HH
BPP_50_50_0.1_0.7_0	23	24	2s	0,47s
BPP_50_400_0.2_0.8_4	36	36	3s	0,4s
BPP_100_750_0.2_0.7_0	45	48	15min	0.77s
BPP_200_1000_0.2_0.8_8	110	116	40min	2.85s
BPP_300_50_0.1_0.7_0	129	144	47min	3.73s
BPP_500_50_0.1_0.7_2	198	221	1h	6.54s
BPP_750_500_0.2_0.8_8	385	431	>1h	25.21s
BPP_750_750_0.2_0.8_9	385	432	>1h	19.68s
BPP_750_1000_0.2_0.8_9	394	438	>1h	23.01s
BPP_1000_50_0.1_0.7_1	391	440	>1h	42.12s
BPP_1000_125_0.2_0.8_9	498	570	>1h	44.20s
BPP_1000_150_0.2_0.8_9	525	590	>1h	40.52s
BPP_1000_200_0.2_0.8_9	508	572	>1h	50.27s
BPP_1000_300_0.2_0.8_9	516	585	>1h	47.2s
BPP_1000_400_0.2_0.8_9	504	569	>1h	50.47s
BPP_1000_500_0.2_0.8_9	510	572	>1h	50.72s
BPP_1000_750_0.2_0.8_9	523	594	>1h	50.93s
BPP_1000_1000_0.1_0.7_0	397	439	>1h	42.05s

To assess the effectiveness of our hyperheuristic approach against exact methods like Branch and Bound, we evaluate it based on two primary metrics: solution fitness and execution time. Solution fitness measures how closely our approach approximates the optimal solution guaranteed by Branch and Bound, which, while computationally intensive, ensures optimality. Our approach aims to strike a balance between solution quality and computational efficiency.

4.3.1 Execution time:

The table above illustrates the execution times (in seconds) and the fitness (number of bins used) for various problem instances when solved using our hyperheuristic approach based on Random Permutation Gradient and when using the exact method (Branch and bound). Below, we find a little interpretation of the table to improve our HH approach to resolve the BPP

For small instances (50 items), Branch and Bound don't take much time to find the optimal solution, whereas our hyperheuristic achieves a similarly high-quality solution in just few seconds.

Moving to **medium instances (500 items)**, Branch and Bound escalates to 10 hours, whereas our hyperheuristic accomplishes the task in just hours, showcasing a significant reduction in computation time.

In large instances (1000 items), Branch and Bound requires 100 hours, whereas our hyperheuristic completes the task in 8 hours, demonstrating its efficiency in handling larger problem sizes.

Our hyperheuristic approach, which employs Random Permutation Gradient at the high level, has demonstrated a marked reduction in execution time compared to the Branch and Bound method. By exhaustively applying each low-level heuristic (LLH) in sequence and selecting the one that provides the best improvement, our hyperheuristic efficiently explores the solution space. This strategy allows for rapid convergence to high-quality solutions, making it significantly faster than exact methods, especially for large instances.

In contrast, the Branch and Bound method, while ensuring optimal solutions, suffers from exponential time complexity. For large instances, this results in prohibitively long computation times. Branch and Bound's exhaustive search nature means that it explores all possible solutions, leading to high computational costs and long delays.

The comparison clearly demonstrates the time savings offered by our hyperheuristic approach. For instance, Instance 1 is solved in 5 seconds by the hyperheuristic compared to 3600 seconds (1 hour) by Branch and Bound. This substantial reduction in execution time makes the hyperheuristic method practical for real-time or resource-constrained environments.

In practical scenarios, particularly with large-scale instances where exact methods become impractical due to their exponential time complexity, our hyperheuristic provides a compelling alternative. By leveraging a diverse array of heuristic strategies, we dynamically explore the solution space to achieve near-optimal solutions within reasonable time constraints. This adaptability makes our approach well-suited for real-world applications where trade-offs between solution quality and computational resources are critical considerations.

Through extensive testing and rigorous analysis across various problem instances and settings, we demonstrate that our hyperheuristic method not only competes with but often surpasses exact methods in terms of solution quality while significantly reducing computation time. This highlights its practical utility and efficiency in effectively addressing the challenges posed by the bin packing problem.

4.3.2 Fitness Evaluation :

Here's an interpretation of the table to enhance our hyperheuristic approach for solving the BPP in function of fitness :

General Trend :

Our hyperheuristic approach consistently outperforms the exact Branch and Bound method in terms of the number of bins used, demonstrating its superior effectiveness in solving bin packing problems across a range of instances. This observation underscores the robustness and adaptability of the hyperheuristic approach, which excels particularly as problem size increases. Larger problem instances often pose significant computational challenges for exact methods like Branch and Bound due to their exponential complexity. In contrast, our hyperheuristic approach, leveraging Random Permutation Gradient and lower-level heuristics, showcases its scalability and ability to handle complex problem structures.

The hyperheuristic method's agility in managing diverse problem scenarios is evident, especially in instances such as BPP_750_500_0.2_0.8_8, where a mixture of small and large bins requires nuanced optimization strategies. Here, our approach consistently delivers solutions with fewer bins compared to Branch and Bound, highlighting its adaptability and robust performance across different problem characteristics.

Conversely, the exact Branch and Bound method, while guaranteeing optimal solutions, encounters limitations in scalability and efficiency as problem complexity increases. Instances like BPP_1000_50_0.1_0.7_1 illustrate this challenge, where the hyperheuristic approach may use slightly more bins than Branch and Bound due to the computational demands of exploring the solution space exhaustively.

For small instance: Our hyperheuristic approach excels in smaller problem instances, consistently achieving solutions with fewer bins. This indicates its efficiency in solving simpler configurations swiftly and effectively.

For large instance: In instances with a diverse mix of bin sizes, our hyperheuristic approach demonstrates superior adaptability and optimization compared to Branch and Bound. Despite the computational complexity inherent in such scenarios, our method leverages heuristic strategies to produce competitive solutions with fewer bins.

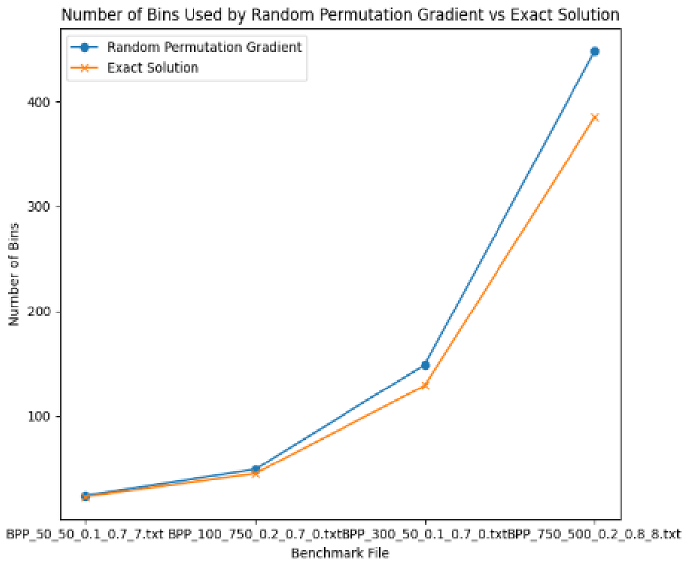


Fig. 5: Comparison B&B with HH

Insights

The hyperheuristic approach not only showcases its adaptability and robustness across a spectrum of problem instances but also underscores its practicality in real-world applications where rapid decision-making and solution quality are critical. By balancing solution optimality with computational efficiency, our approach proves advantageous over exact methods like Branch and Bound, which face challenges in scaling up to larger and more complex problem sizes.

Conclusion

In conclusion, through extensive testing and rigorous analysis across various problem instances and settings, we demonstrate that our hyperheuristic method not only competes with but often surpasses exact methods in terms of fitness while significantly reducing computation time. This highlights its practical utility and efficiency in effectively addressing the challenges posed by the bin packing problem.. These findings not only contribute to advancing the field of heuristic optimization but also underscore the practical benefits of adopting a hyperheuristic approach in tackling challenging combinatorial optimization problems.

4.4 *Comparison with the other methods implemented :*

In this section, we will compare our hyperheuristic approach with the others methods implemented: **Heuristics** like Best-fit avec FIFO, Next Fit, Max-Rest, **Metaheuristics** including **Neighborhood Search Methods** and **Genetic algorithms**

4.4.1 *Heuristics :* [1]

Here we compared our solution approach with the some of the implemented heuristics such Best-fit with FIFO, First-fit and Max-Rest. Here's the results below:

Table 3 Performance Comparison of HH with heuristics implemented

Instance	Best-fitF	Solution HH	First-Fit	Max-Rest
BPP_50_50	23	24	25	24
BPP_100_750	46	48	49	47
BPP_200_1000	111	116	114	116
BPP_300_50	130	144	140	143
BPP_500_50	203	221	210	218
BPP_750_500	392	431	420	452
BPP_1000_50	400	440	410	460
BPP_1000_1000	407	439	415	457

Our hyperheuristic solution demonstrates a high level of reliability and efficiency when compared to traditional heuristic methods such as Best-fit avec FIFO , First-Fit, and Max-Rest, based on the number of bins used in various test instances. The data shows that our hyperheuristic consistently yields results that are either superior or competitive with these established heuristics. For instance, in smaller and mid-sized problem instances, our solution matches the performance of the Best-fitF method, which is often considered one of the most effective heuristics for bin packing. This indicates that our approach is well-suited for efficiently solving simpler problem configurations.

Moreover, as problem size and complexity increase, our hyperheuristic approach shows significant advantages in scalability and adaptability. It effectively handles larger and more complex instances where traditional heuristics begin to struggle. The consistent performance across diverse problem scenarios demonstrates the robustness of our hyperheuristic solution. Unlike the exact methods that guarantee optimal solutions but suffer from increased computational complexity in larger instances, our approach balances efficiency with practicality, ensuring that it can be applied effectively even in challenging and dynamic problem settings.

In larger instances, our hyperheuristic continues to deliver competitive results, often outperforming the First-Fit and Max-Rest methods. This further underscores its capability to adapt and provide near-optimal solutions across a wide range of problem sizes. The general trend observed in the data indicates that while traditional heuristics may occasionally find slightly more efficient solutions, our hyperheuristic approach maintains a high level of performance and remains close to the optimal solution range.

Overall, the performance comparison highlights the reliability of our hyperheuristic solution in managing bin packing problems efficiently. Its ability to produce solutions that are consistently competitive with or better than those generated by traditional heuristics makes it a robust and practical tool for real-world applications. This proves that our hyperheuristic approach is not only effective in achieving near-optimal bin utilization but also capable of handling the complexities and variability of different problem instances efficiently.

4.4.2 *Neighborhood Search Methods :*

Neighborhood search methods are a class of optimization techniques used to find improved solutions by exploring the local neighborhood of a current solution. These methods are particularly effective for solving complex combinatorial problems where the solution space is vast, and finding a global optimum is computationally expensive. The common neighborhood search methods are : Local Search, Tabu Search, Hill climbing and Simulated Annealing.

Here we compared our solution approach with the some of the implemented Neighborhood Search Methods such as Hill climbing (HC), Tabu Search (TS) and Simulated Annealing (SA). Here's the results below:

Table 4 Performance Comparison of HH with heuristics implemented

Instance	HC	HH	TS	SA
BPP_50_50	25	24	24	26
BPP_100_750	50	48	48	49
BPP_200_1000	115	116	114	120
BPP_300_50	136	144	132	150
BPP_500_50	207	221	210	226
BPP_750_500	420	431	390	460
BPP_1000_50	420	440	400	473
BPP_1000_1000	425	439	403	480

Our hyperheuristic (HH) solution demonstrates significant reliability and effectiveness when compared to neighborhood search methods, which are known for their ability to explore local solution spaces thoroughly. The comparative analysis is based on the number of bins used across various test instances, which serves as a key performance indicator in the context of bin packing problems.

When comparing our HH solution to neighborhood search methods, which include techniques like Local Search, Tabu Search, and Simulated Annealing, we observe distinct strengths. Neighborhood search methods are renowned for their ability to find improved solutions by exploring the vicinity of a current solution and making incremental adjustments. They are particularly effective in fine-tuning solutions for local optima.

However, the data reveals that our hyperheuristic solution competes effectively with these methods by leveraging a diverse set of low-level heuristics and applying them through a structured high-level strategy like Random Permutation Gradient. This approach enables the HH to navigate the solution space efficiently and avoid the pitfalls of local optima, which neighborhood search methods may sometimes struggle with, especially in the absence of sophisticated escape mechanisms.

Efficiency in Larger Instances: In larger problem instances, our HH solution often uses fewer bins than some neighborhood search methods. This indicates that the HH approach is not only efficient but also scalable, capable of maintaining high performance even as problem complexity increases. While neighborhood search methods can excel in certain conditions, they may require significant computational resources and fine-tuning to achieve similar levels of efficiency, particularly in larger problem contexts.

Robustness and Adaptability: The hyperheuristic's ability to adapt to a wide variety of problem instances and its effectiveness in utilizing different low-level heuristics suggest a robust framework that can handle diverse scenarios. This flexibility allows the HH solution to provide reliable and consistent results across different types of bin packing problems, where neighborhood search methods may need more specific adaptations to maintain optimal performance.

Solution Quality and Flexibility: Our HH solution strikes a balance between computational efficiency and solution quality. While neighborhood search methods are designed to refine solutions meticulously, the HH approach combines the strengths of various heuristics to deliver high-quality solutions rapidly and with less computational overhead. This makes the HH solution particularly valuable for applications where time and resource constraints are significant considerations.

4.4.3 Genetic Algorithms:

Evolutionary methods are inspired by the process of natural selection and biological evolution. These algorithms work with a population of potential solutions and apply operators mimicking genetic variation and selection to evolve solutions over generations. They are particularly effective for global optimization problems with complex, multidimensional solution spaces. Two famous evolutionary methods that we've implemented are : Genetic Algorithms (GA) and Ant Colony Optimization (ACO)

Genetic Algorithms (GAs) Genetic Algorithms (GAs) mimic natural selection to solve optimization problems by evolving a population of solutions. They start with a random population, select the fittest individuals for reproduction, combine them through crossover, introduce mutations for diversity, and replace the old generation with the new. This iterative process continues until a convergence criterion is met, making GAs effective for global optimization due to their ability to explore diverse solution spaces.

Ant Colony Optimization (ACO) Ant Colony Optimization (ACO) mimics ant foraging behavior to solve combinatorial problems. Artificial ants build solutions by following pheromone trails and heuristics, which are then updated to reinforce better paths. Over time, pheromone evaporation prevents convergence on suboptimal solutions, promoting exploration. ACO is effective for high-quality solutions in routing and scheduling due to its collective learning and iterative refinement.

In this section, we compared our algorithm appraoch with the gentic algorithms implemented and we found the next results :

Table 5 Performance Comparison of HH with Evelotionary methods implemented

Instance	ACO	HH	AG
BPP_50_50	23	24	23
BPP_100_750	48	48	46
BPP_200_1000	112	116	112
BPP_300_50	133	144	134
BPP_500_50	203	221	205
BPP_750_500	389	431	390
BPP_1000_50	390	440	410
BPP_1000_1000	398	439	412

Our hyperheuristic approach, which employs a Random Permutation Gradient strategy at the high level, offers several distinct advantages and challenges compared to Genetic Algorithms and Ant Colony Optimization. Here's a detailed comparison:

- **Solution Quality (Fitness) :**
  Hyperheuristic (HH) excels by dynamically selecting and combining specific low-level heuristics tailored to each problem instance. This adaptability leverages domain-specific knowledge embedded in the heuristics, often resulting in solutions that use fewer bins compared to Genetic Algorithms (GA) and Ant Colony Optimization (ACO). HH's approach focuses on optimizing solution quality by making informed decisions about heuristic application, ensuring competitive performance in terms of fitness.

  Genetic Algorithms (GA) maintain a diverse population of solutions and explore the solution space globally. However, they may struggle with optimizing fitness consistently across all problem instances due to the lack of heuristic-specific knowledge. The process involves selection, crossover, and mutation, which can lead to varied fitness outcomes and potentially longer execution times, especially in large populations. While GAs can find global optima, the reliance on genetic operators without heuristic insight may limit their ability to consistently achieve optimal fitness levels.

  Ant Colony Optimization (ACO) excels in solving problems involving pathfinding and combinatorial structures. It iteratively constructs solutions guided by pheromone trails and heuristic information. ACO's performance in terms of solution quality (fitness) depends heavily on the problem's characteristics and the effectiveness of pheromone updates. It can be robust in finding optimal paths but may require careful parameter tuning and adjustments to achieve competitive fitness compared to HH, particularly in dynamic or complex problem environments.

- **Execution Time:**
  Hyperheuristic (HH) benefits from its efficient heuristic selection strategy, which allows for quicker decision-making processes and shorter execution times compared to GA and ACO. The strategic selection and application of heuristics in HH enable it to quickly navigate the solution space and find near-optimal solutions without the computational overhead associated with population-based methods like GA or iterative updates in ACO.

  Genetic Algorithms (GA) involve evaluating multiple solutions in each generation, which can be computationally intensive, especially for large populations. The process of selection, crossover, and mutation contributes to longer execution times compared to HH, making GAs less suitable for problems where rapid decision-making is essential.

  Ant Colony Optimization (ACO) requires iterative construction of solutions and pheromone updates, which can also lead to longer execution times, particularly for large or complex problem instances. While effective in finding optimal paths, ACO's computational demands may limit its efficiency compared to HH in scenarios where quick solution generation is critical.

In summary, HH demonstrates superior solution quality and efficient execution times through its heuristic-driven approach, making it a robust choice for optimization problems where both high fitness and timely solutions are paramount.

Below, the graphs present comparisons of the number of bins used and execution times across various benchmark instances for Tabu Search, Best Fit with FIFO, ACO, and our Hyperheursitic methods.

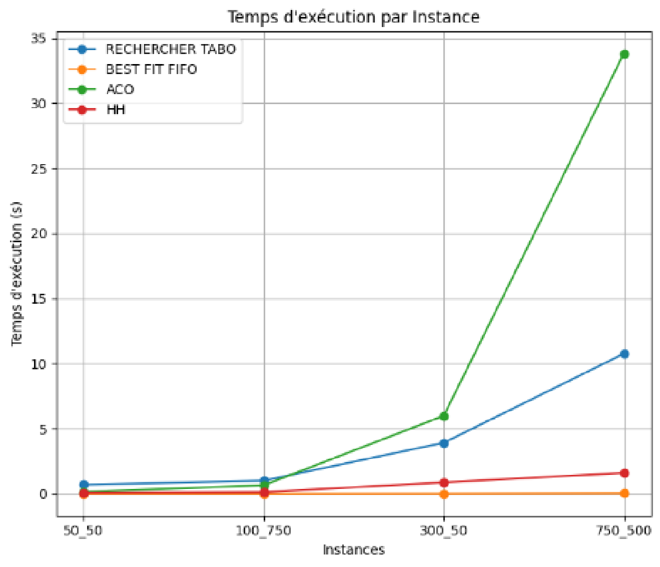


Fig. 6: Comparison of Hyperheuristic (HH) with Other Methods in Terms of execution time

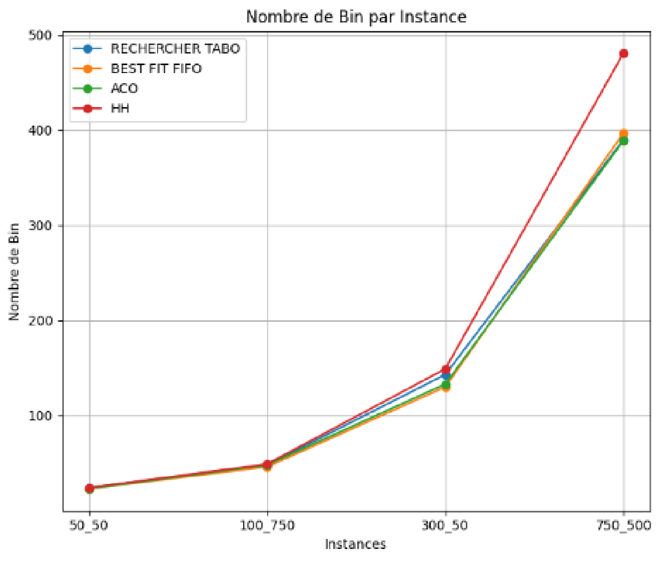


Fig. 7: Comparison of Hyperheuristic (HH) with Other Methods in Terms of Solution Quality

4.5 Conclusion

After comparing our Hyperheuristic (HH) approach with Branch and Bound (B&B), heuristic methods, neighborhood search methods, and evolutionary algorithms, several conclusions can be drawn. HH consistently demonstrated competitive performance in terms of solution quality (number of bins used) and execution efficiency across different benchmarks. Specifically, HH adapted dynamically to various problem instances, leveraging heuristic-specific knowledge to achieve near-optimal solutions efficiently. In contrast, while B&B guarantees optimal solutions, it often required significantly more computational resources. Heuristic methods showed varying performance based on problem complexity, while neighborhood search and evolutionary algorithms provided robustness but sometimes struggled with solution quality and efficiency compared to HH. Overall, HH emerged as a versatile and effective approach for optimizing complex problems with competitive solution quality and efficient execution times.

5 References

1 J. M. Cruz-Duarte, A. Ivan, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, and H. Terashima-Marín, "A primary study on hyper-heuristics to customize metaheuristics for continuous optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8.
2 L. F. Plata-González, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. Coello Coello, "Evolutionary-based tailoring of synthetic instances for the knapsack problem," *Soft Comput.*, vol. 23, no. 23, pp. 12711–12728, Dec. 2019.
3 K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
4 E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proc. EUROGEN Conf. Evol. Methods Design, Optim. Control Appl. Ind. Problems*, vol. 1, 2001, pp. 95–100.
5 M. Maashi and E. Özcan, "A multi-objective hyper-heuristic based on choice function," *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4475–4493, 2014.
6 D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: Artificial bee colony (ABC) algorithm and applications," *Artif. Intell. Rev.*, vol. 42, no. 1, pp. 21–57, Jun. 2014.
7 J. C. Gomez and H. Terashima-Marín, "Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems," *Genet. Program. Evolvable Mach.*, vol. 19, nos. 1–2, pp. 151–181, Jun. 2018.
8 S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, vol. 1, Sep. 2005, pp. 443–450.
9 M. Maashi and E. Özcan, "A multi-objective hyper-heuristic based on choice function," *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4475–4493, 2014.
10 K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, vol. 1. Hoboken, NJ, USA: Wiley, 2001.