Evaluating LLMs for Combinatorial Optimization: One-Phase and Two-Phase Heuristics for 2D Bin-Packing

Anonymous Authors

Abstract

This paper presents an evaluation framework for assessing Large Language Models' (LLMs) capabilities in combinatorial optimization, specifically addressing the 2D bin-packing problem. We introduce a systematic methodology that combines LLMs with evolutionary algorithms to generate and refine heuristic solutions iteratively. Through comprehensive experiments comparing LLM generated heuristics against traditional approaches (Finite First-Fit and Hybrid First-Fit), we demonstrate that LLMs can produce more efficient solutions while requiring fewer computational resources. Our evaluation reveals that GPT-4o achieves optimal solutions within two iterations, reducing average bin usage from 16 to 15 bins while improving space utilization from 0.76-0.78 to 0.83. This work contributes to understanding LLM evaluation in specialized domains and establishes benchmarks for assessing LLM performance in combinatorial optimization tasks.

1 Introduction

The evaluation of Large Language Models (LLMs) extends beyond traditional natural language processing tasks to specialized domains like combinatorial optimization. The 2D bin-packing problem that is placing rectangles into the minimum number of fixed-size bins represents a challenging NP-hard optimization task that serves as an ideal testbed for evaluating LLM capabilities in mathematical reasoning and algorithmic design.

Traditional heuristic approaches like Finite First-Fit (FFF) and Hybrid First-Fit (HFF) provide established baselines, but their performance limitations in scalability and solution quality create opportunities for LLM enhanced approaches. This paper evaluates how effectively LLMs can generate, refine, and optimize heuristic algorithms through an iterative evolutionary framework.

Our evaluation framework addresses key questions: Can LLMs understand complex algorithmic constraints? How do LLM generated solutions compare to established heuristics? What evaluation metrics best capture LLM performance in optimization contexts?

2 Mathematical Formulation

The two-dimensional bin packing problem (2D-BPP), an NP-hard problem, seeks to pack n items of size (w_i, h_i) into the minimum number of bins of size (W, H), where $W > w_i$ and $H > h_i$ for all $i \in \{1, ..., n\}$ [15, 8, 4]. Let the indicator variable $z_{ij} = 1$ when item i is placed in bin j and 0 otherwise; similarly, $u_j = 1$ when bin j is used and 0 otherwise. By the pigeonhole principle, a maximum of n bins is needed [6]. The optimization problem is formulated as follows:

$$\min \sum_{j=1}^{n} u_j$$

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Subject to the following constraints for all $i, j \in \{1, ..., n\}$: $\sum_{j=1}^{n} z_{ij} = 1$; $0 \leq x_{ij} \leq (W - w_i)z_{ij}$; $0 \leq y_{ij} \leq (H - h_i)z_{ij}$; $u_j \geq z_{ij}$; together with standard non-overlap constraints, ensuring that no two items in the same bin overlap [11, 13]. Finally, the total utilization, a common metric to evaluate performance for a given solution is measured as $\rho_{\text{total}} = \frac{\sum_{i \in I} w_i h_i}{\left(\sum_{j=1}^{n} u_j\right) WH}$ [5, 9].

3 Evaluation Framework

Problem Formulation and Constraints: We evaluate LLMs on the 2D bin-packing problem with strict constraints: bin dimensions of 200×100 units, item constraints requiring no overlap and complete containment within bins, the objective to minimize number of bins used, and an evaluation dataset of 50 randomly generated squares (10-50 units) across 20 iterations.

LLM Based Evolutionary Process:

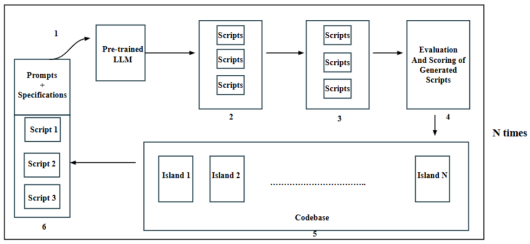

Figure 1: Iterative Evolutionary Framework for Heuristic Generation

Our evaluation methodology employs a six-step iterative process. First, structured prompting designs prompts that clearly specify problem constraints, input/output formats, and success criteria. Second, code generation and correctness validation systematically validates LLM generated candidate solutions against constraint satisfaction. Third, performance scoring evaluates solutions using multiple metrics: number of bins used (primary), space utilization efficiency (secondary), and execution time (tertiary). Fourth, island-based selection clusters high-performing solutions into "islands" to promote diversity. Fifth, iterative refinement uses the top performing solutions to inform subsequent prompts, creating an evolutionary feedback loop.

To implement this framework, each generated script is rigorously validated for syntactic and logical correctness; only solutions that successfully pack all items according to the rules are advanced to the performance evaluation stage. The high performing solutions are clustered into distinct "islands" to preserve strategic diversity and prevent premature convergence on a single type of solution. In the refinement stage, the top three performing solutions one from each of the top three islands are used as "best-shot" examples in the prompt for the next generation cycle. This evolutionary feedback loop instructs the LLM to learn from the most successful strategies, progressively enhancing the quality of the generated heuristics over six full iterations. A detailed breakdown of each component, including full prompt design and baseline implementations, is available in Appendix B.

Baseline Comparisons: We establish baselines using two established heuristics. Finite First-Fit (FFF) places items in the first available position using First-Fit Decreasing Height (FFDH) with time complexity $O(n^2)$. Hybrid First-Fit (HFF) employs a two-phase approach combining strip packing (FFDH) with bin packing (FFD) with time complexity $O(n \log n)$.

4 Experimental Setup:

We conducted experiments using GPT-4o with BPE tokenization on an Intel Core i5-8250U processor with 8GB RAM. The dataset consisted of 20 iterations with 50 randomly generated squares per iteration, and the evaluation protocol used the same dataset for all methods to ensure fair comparison. The LLM evaluation process terminated after demonstrating convergence within 2-6 iterations, indicating rapid solution optimization capability.

5 Results and Discussion

Comparative Performance

Method	Avg Bins	Execution Time (s)	Space Utilization
FFF	16.05	0.002446	0.76
HFF	16.00	0.024438	0.78
LLM	15.00	0.0103	0.83

Table 1: Comparative performance across evaluation metrics

The LLM-generated heuristic demonstrates superior performance across all evaluation metrics, achieving a 6.25% reduction in bin usage compared to baselines, a 6.4% improvement in space utilization over HFF, and competitive execution time despite code generation overhead.

Convergence Analysis The LLM achieved optimal solutions within two iterations, suggesting efficient learning from constraint feedback. This rapid convergence indicates strong pattern recognition capabilities and effective constraint satisfaction learning.

Space Utilization Patterns LLM generated solutions show more consistent space utilization across bins (83% average) compared to traditional heuristics, which exhibit declining utilization in later bins (HFF: 86.83% $\rightarrow$ 63.54%, FFF: 87.50% $\rightarrow$ 68.00%).

LLM Capabilities Assessment Our evaluation reveals several key capabilities. LLMs successfully internalize complex geometric and logical constraints, demonstrating sophisticated constraint understanding. Generated solutions exhibit optimization intuition through sophisticated packing strategies not explicitly programmed. The results show consistent iterative improvement across evolutionary cycles, indicating effective learning mechanisms.

Limitations and Evaluation Challenges Computational constraints limit iteration cycles due to API costs, constraining comprehensive evaluation. LLM non-determinism complicates reproducibility, requiring multiple evaluation runs for statistical validity. The evaluation was limited to moderate problem sizes, and larger instances may reveal different performance characteristics that could affect generalization.

Evaluation Metric Considerations Traditional optimization metrics (bin count, space utilization) prove effective for LLM evaluation, but additional metrics considering code quality, algorithmic sophistication, and constraint satisfaction robustness could provide deeper insights into LLM problem-solving capabilities.

Implications for LLM Evaluation This work contributes to LLM evaluation methodology through domain-specific benchmarking that demonstrates the value of specialized evaluation frameworks for assessing LLM capabilities beyond language tasks. The iterative evaluation protocols show how evolutionary feedback can systematically evaluate LLM learning and adaptation capabilities. Multi-metric assessment establishes that comprehensive LLM evaluation requires performance, efficiency, and solution quality metrics. Finally, baseline establishment provides benchmarks for future LLM evaluation in combinatorial optimization contexts.

6 Related Work

The 2D bin packing problem is a fundamental NP-hard combinatorial optimization challenge where rectangular items must be packed into the minimum number of identical bins without overlapping while respecting bin boundaries [14]. Traditional approaches are broadly categorized into one-phase and two-phase algorithms, each offering distinct advantages for different problem scenarios.

One-phase algorithms pack items directly into bins using strategies such as next-fit, first-fit, and best-fit methods combined with placement heuristics like bottom-left (BL) and bottom-left-fill (BLF) to determine specific item positions within selected bins [14]. These approaches prioritize computational efficiency but may sacrifice solution quality due to their greedy nature.

Two-phase algorithms decompose the packing process into sequential stages, with the most established approach using level-based packing where items are first organized into levels of infinite-height

strips, then stacked into finite bins [1]. Classic implementations include Hybrid First-Fit (HFF) and Finite Best-Strip (FBS), which build upon foundational algorithms like First-Fit Decreasing Height (FFDH) and Best-Fit Decreasing Height (BFDH) [1]. Modern two-phase approaches have evolved to include sophisticated decomposition strategies such as the Positions and Covering (P&C) methodology, which generates valid item positions before using set-covering formulations for optimal configuration selection [2].

Performance analysis reveals significant trade-offs between solution quality and computational efficiency. Ferreira's comparative study of constructive First-Fit Decreasing strategies, local search, simulated annealing, and genetic algorithms demonstrated that while constructive heuristics provide rapid solutions, improvement-based methods offer superior solution quality at increased computational cost [3]. Specific placement strategies like BLF position items iteratively from the lower-left corner, while FFD and BFD algorithms employ different bin selection criteria based on item ordering and space utilization [10].

Recent developments have integrated machine learning techniques with traditional heuristics, including deep reinforcement learning approaches for dynamic scenarios and hierarchical frameworks combining heuristic search with learning-based optimization [7]. However, these approaches remain largely problem specific and have not established systematic evaluation frameworks for assessing algorithmic performance across diverse problem instances. Though the use of LLMs in an evolutionary loop has shown significant promise, for instance, Romera-Paredes et al. [12] introduced FunSearch, a method that pairs an LLM with an evaluator to discover novel, high-performing heuristics for problems such as online bin packing.

Inspired from the work of FunSearch, we contribute to this landscape by introducing a structured evaluation methodology specifically designed for assessing Large Language Model capabilities in generating and optimizing heuristic algorithms for the 2D bin packing problem, addressing the gap in systematic evaluation approaches for AI enhanced combinatorial optimization.

7 Conclusion

This paper presents a systematic framework for evaluating LLMs in combinatorial optimization contexts. Through comprehensive experiments on the 2D bin-packing problem, we demonstrate that LLMs can generate superior heuristic solutions compared to established algorithms while providing efficient performance. The evaluation framework contributes to understanding LLM capabilities in specialized domains and establishes methodological approaches for assessing LLM performance in optimization tasks.

Our results indicate that LLMs possess significant potential for enhancing combinatorial optimization approaches, achieving measurable improvements in solution quality and computational efficiency. These findings support continued research into LLM applications in mathematical and algorithmic domains while highlighting the importance of rigorous evaluation frameworks for assessing such capabilities.

8 Future Work

Several key research directions emerge from this evaluation framework. First, scalability assessment should investigate how these results scale to larger bin-packing instances or different constraint ratios, as the current 200x100 bins with 10-50 unit squares represents a specific problem space that may not generalize to industrial-scale applications. Second, solution interpretability analysis should characterize the specific strategies the LLM discovered that led to improved performance, as understanding the algorithmic innovations behind the 6.25% improvement would inform future heuristic design and provide insights into LLM reasoning capabilities. Third, reproducibility analysis must address how evaluation frameworks should handle LLM non-determinism through protocols for multiple trial runs, confidence interval reporting, and statistical significance testing to ensure robust evaluation methodologies. Finally, prompt engineering sensitivity requires systematic investigation of how results vary with prompt modifications, as the evaluation framework's dependence on prompt design necessitates establishing reproducibility guidelines and optimal prompting strategies for optimization contexts.

References

[1] Christian Blum, Verena Schmid, and Lukas Baumgartner. On solving the oriented two-dimensional bin packing problem under free guillotine cutting: Exploiting the power of probabilistic solution construction. *arXiv preprint arXiv:1209.xxxxx*, 2012.

[2] Néstor M Cid-García and Yasmín A Ríos-Solís. Positions and covering: A two-stage methodology to obtain optimal solutions for the 2d-bin packing problem. *PLoS ONE*, 15(2), 2020.

[3] Duarte Nuno Gonçalves Ferreira. Rectangular bin-packing problem: a computational evaluation of 4 heuristics algorithms. Master's thesis, University of Porto, 2017.

[4] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[5] Manuel Iori, Silvano Martello, and Federico Maffioli. 2dpacklib: A two-dimensional bin packing problem library. *INFORMS Journal on Computing*, 33(1):302–308, 2021.

[6] Richard Johnsonbaugh. *Discrete mathematics*. Pearson, NY, NY, eighth edition edition, 2018.

[7] Beomjoon Lee and Changjoo Nam. A hierarchical bin packing framework with dual manipulators via heuristic search and deep reinforcement learning. *arXiv preprint arXiv:2501.xxxxx*, 2025.

[8] Silvano Martello and Paolo Toth. Knapsack problems: algorithms and computer implementations. *John Wiley & Sons*, 1990.

[9] José F Oliveira, Maria A Carravilla, and Fabio Furini. Two-dimensional cutting and packing problems: a survey. *International Transactions in Operational Research*, 2023.

[10] Camelia-M Pintea, Cristian Pascan, and Mara Hajdu-Macelaru. Comparing several heuristics for a packing problem. *International Journal of Advanced Intelligence Paradigms*, 4(2):164–174, 2012.

[11] David Pisinger and Martin Sigurd. An exact algorithm for large multiple knapsack problems. *INFORMS Journal on Computing*, 19(3):475–482, 2007.

[12] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[13] Andreas Seizinger. Exact approaches for the two-dimensional bin packing problem with side constraints. *European Journal of Operational Research*, 301(3):837–852, 2022.

[14] Wenjie Wu, Changjun Fan, Jin-Yu Huang, Zhong Liu, and Junchi Yan. Machine learning for the multi-dimensional bin packing problem: Literature review and empirical evaluation. *arXiv preprint arXiv:2310.xxxxx*, 2023.

[15] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

A Appendix - Source Code

The code is open source and kept anonymous due to submission policy.

B Appendix - Methodology

B.1 Goal

The goal of this study is to explore how a Large Language Model (LLM) can autonomously generate, evaluate, and refine heuristics for solving the 2D Bin Packing Problem (2D-BPP). This is accomplished through an iterative loop in which the LLM, specifically GPT-4o, writes Python functions based on a well-defined prompt, evaluates their ability to pack items efficiently, and leverages the best-performing scripts to guide the next round of generation.

Through this multi-round learning framework, we aim to determine whether an LLM can discover a packing strategy that rivals or surpasses classical heuristics like Finite First-Fit (FFF) and Hybrid First-Fit (HFF). This methodology not only examines the end results but also provides insight into how the heuristics evolve through contextual learning and prompt refinement.

B.2 Dataset

Each iteration employs a dataset containing 50 randomly generated square items. Each item has a side length randomly chosen between 10 and 50 units. All bins are of fixed dimensions—200 units in width and 100 units in height—and every square must be placed without overlapping and within the confines of the bin. Twenty different datasets were created, each representing a unique and random configuration to simulate varied real-world packing scenarios. By ensuring that each dataset is distinct, the evaluation of heuristic performance remains unbiased and avoids overfitting to any specific pattern of item sizes or arrangements.

B.3 FFF and HFF Scripts

To provide a baseline for performance comparison, two traditional heuristics were implemented: Finite First-Fit (FFF) and Hybrid First-Fit (HFF). In the FFF method, items are sorted by height and packed into the first available bin from bottom to top. If no available position is found within existing bins, a new bin is opened. This greedy strategy is computationally efficient but often results in poor space utilization.

In contrast, the HFF approach operates in two phases. First, it applies the First-Fit Decreasing Height (FFDH) method to create horizontal strip packings by sorting items based on height. Second, these strips are packed into bins using the First-Fit Decreasing (FFD) approach. The combination of strip-level optimization and bin-first fit allocation allows HFF to improve upon the naive nature of FFF, particularly in scenarios involving large numbers of irregular-sized items. Both heuristics were implemented in Python and tested across the same datasets as the LLM-generated heuristics to ensure fair comparison.

B.3.1 Finite First-Fit (FFF) Flow

1. Start
2. Initialize empty bins
3. For each item:
   (a) Check bins one by one
   (b) If fits $\rightarrow$ Place item $\rightarrow$ Next item
   (c) If no bin fits $\rightarrow$ Open new bin $\rightarrow$ Place item
4. All items placed? If Yes $\rightarrow$ End
5. If No $\rightarrow$ Repeat for next item

B.3.2 Hybrid First-Fit (HFF) Flow

1. Start
2. Initialize empty bins
3. For each item:
   (a) Apply heuristic to select bin (First-Fit or alternative)

(b) If bin fits → Place item → Next item

(c) If no bin fits → Open new bin → Place item

4. All items placed? If Yes → End

5. If No → Repeat for next item

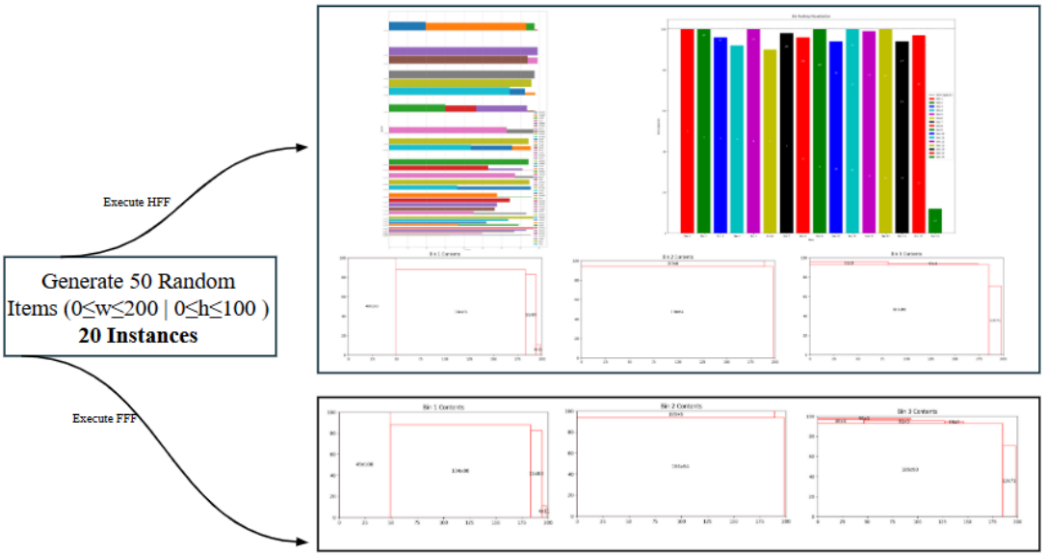

Figure 2: Flowchart of FFF and HFF process steps

B.4 Large Language Model

We utilized the GPT-4o model from OpenAI for the heuristic script generation. The model was provided with a comprehensive prompt, which included a function prototype, input/output specifications, and strict constraints. The function was expected to accept a NumPy array of item dimensions and a tuple representing bin capacities, and return a list of bins with items mapped to specific coordinates. The prompt explicitly instructed the model to ensure non-overlapping placements, respect bin boundaries, and avoid duplication of items across bins. A template function was included to enforce syntactic consistency, making it easier to validate, test, and compare the generated scripts.

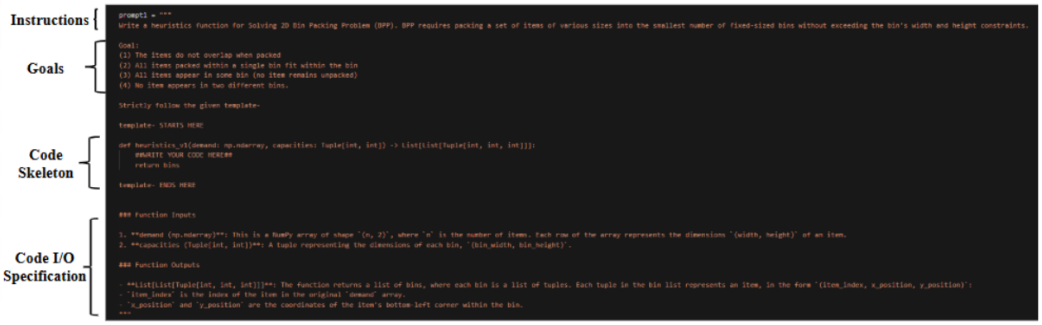

Figure 3: Template used in the LLM prompt

B.5 Prompt Design

The prompt is the primary interface used to communicate the 2D Bin Packing Problem to the LLM. It was carefully crafted to ensure the model understood the objective and constraints of the task. The function needed to handle an array of items and place them into bins in such a way that the constraints were strictly satisfied. To guide the LLM effectively, we included a complete Python

function signature, detailed descriptions of the expected inputs and outputs, and the rules of the problem. We also clarified the goals, such as minimizing the number of bins and ensuring items did not overlap or exceed bin boundaries. The template helped enforce a consistent structure for all generated heuristics.

The prompt is the main interface between the user and the LLM. We designed a clear and structured prompt to instruct the model to write a heuristic function for solving the 2D Bin Packing Problem (2D-BPP). The prompt defined the packing goals and strictly enforced input-output formats. Each function had to pack a list of rectangular items into bins of fixed size while following strict rules: no item could appear in more than one bin, no items could overlap, and all items had to be packed within the bin boundaries. A template format for the Python function was provided to ensure uniformity across all generated scripts.

B.6 Script Generation

The LLM generated multiple scripts based on the initial prompt. Each script was designed to solve the same problem using different logic. A total of 20 scripts were produced during the first round. The variety in the scripts helped cover a wide range of heuristic strategies. These scripts showed significant differences in terms of logic structure, item placement order, and how space within the bins was utilized. Each script was saved for correctness checking and performance scoring.

B.7 Correctness Verification

Once the scripts were generated, they were tested for correctness. Each script had to meet all the packing constraints. The scripts were run on a fixed set of test cases. Outputs were checked to ensure that no item overlapped, every item was placed within bin boundaries, and no item appeared more than once. Incorrect scripts were discarded. Only those that passed all correctness tests were selected for further evaluation.

B.8 Scoring and Evaluation

Scripts that passed the correctness tests were scored using the same metrics applied to traditional heuristics. These included the number of bins used, the total packing density, and the script's runtime. Scripts that used fewer bins and maintained higher densities received higher scores. Execution time was also recorded, though it had a lower weight in score calculation. This method ensured that only efficient and practical scripts moved forward.

B.9 Island Formation

After scoring, high-performing scripts were grouped into islands. Each island contained scripts with similar logic and performance. The term "island" refers to a group of solutions that evolved in parallel but independently. These islands allowed us to preserve diversity among strategies and prevented convergence to a single logic too early in the process.

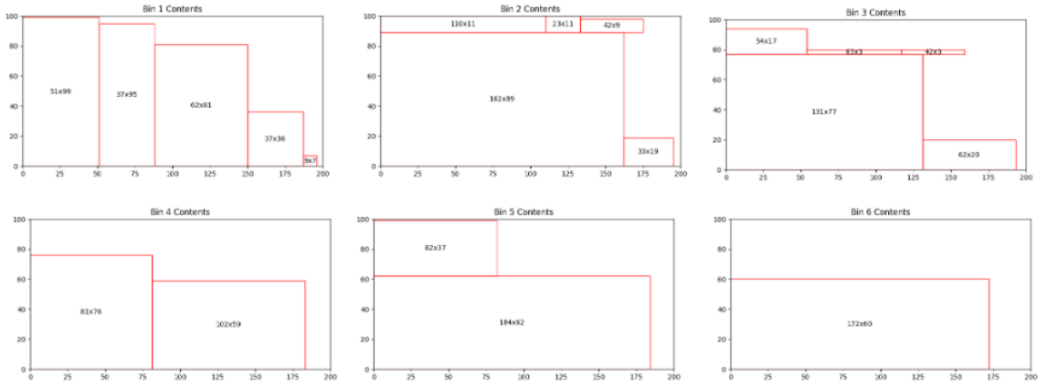

Figure 4: Island distribution after the first iteration, sorted by bin usage

B.10 Iterative Prompt Refinement

The top three performing islands were selected to refine the next round of prompts. One script was randomly chosen from each of these top islands. These scripts were embedded into a new prompt as examples. The prompt instructed the LLM to learn from these three solutions and generate a new heuristic function. This process guided the LLM to focus on effective strategies while still producing novel variations. This approach is known as best-shot learning. It helps the LLM improve script quality without losing diversity.

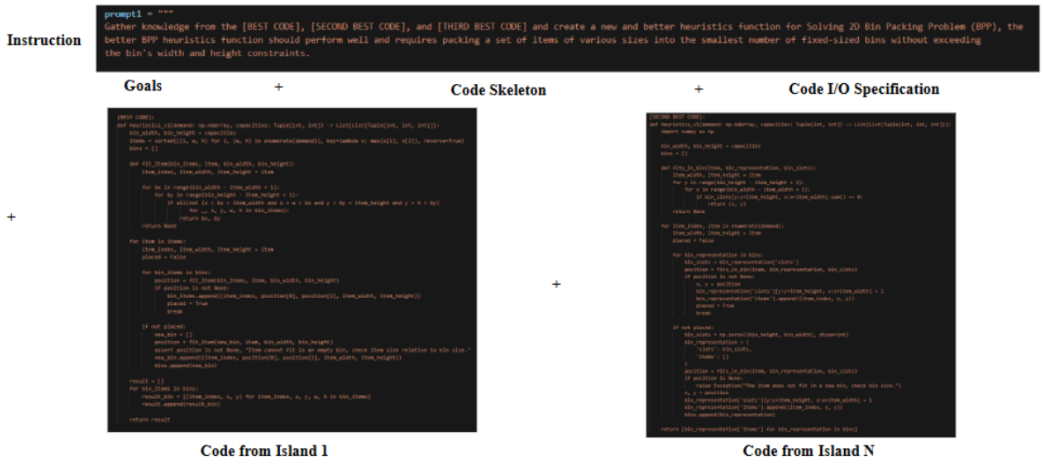

Figure 5: Refined prompt that includes best code samples from top islands

B.11 Iteration and Justification

The process described above was repeated for six iterations. In each round, the best scripts were selected and used to guide the next generation. The goal was to steadily improve solution quality with each iteration. Six rounds were chosen based on resource availability and diminishing returns. Beyond six rounds, improvements were small compared to the extra cost in time and computation. This number of iterations proved effective in reaching high-performing solutions without excessive overhead. The final scripts from the sixth iteration were used in the evaluation and comparison stages.

filename	iteration	total_bins	tal_area_le	time
10_2_generated_code.py	0	16	61781	10.05221415
10_2_generated_code.py	1	15	67672	12.90952539
10_2_generated_code.py	2	16	64725	10.39025068
10_2_generated_code.py	3	15	46447	13.77762437
10_2_generated_code.py	4	14	65474	8.791361332
10_2_generated_code.py	5	13	45930	14.12328315
10_2_generated_code.py	6	19	89100	21.4082129
10_2_generated_code.py	7	15	68457	8.164412975
10_2_generated_code.py	8	16	74585	9.941418171
10_2_generated_code.py	9	16	82946	9.881717682
10_2_generated_code.py	10	12	37590	11.21900916
10_2_generated_code.py	11	14	35427	9.482660294
10_2_generated_code.py	12	17	71446	14.06797814
10_2_generated_code.py	13	17	76079	12.00151372
10_2_generated_code.py	14	14	51363	12.3829267
10_2_generated_code.py	15	12	58295	9.240324259
10_2_generated_code.py	16	16	72219	12.02184606
10_2_generated_code.py	17	16	67927	13.08872342
10_2_generated_code.py	18	11	36646	10.89735484
10_2_generated_code.py	19	15	63898	13.82973814
AVG		**14.95**		**11.88360478**
		≈ 15		

	HFF	FFF
Average Number of Bins	**≈ 16**	**≈ 16**
Average Execution Time	**0.024 sec**	**0.002 sec**

Figure 6: Improvement trend in bin usage over six iterations