University of Nottingham
UK | CHINA | MALAYSIA

COMP2051 - Artificial Intelligence Methods

Coursework: Optimizing Inventory Management with 1D Bin Packing

Author
Mithilesh TEW
hfymt5
20509703

Submitted on 6th May 2025

School of Computer Science
University of Nottingham Ningbo China

Contents

Declaration of AI Usage

I declare that I have used AI assistance, specifically OpenAI's ChatGPT, in the process of programming and writing this essay.

1 Introduction and Background

Efficient inventory management is an essential challenge in industries like logistics, manufacturing, retail, and cloud computing. In several domains, optimising storage and reducing space directly leads to cost savings and increased operational effectiveness. This coursework focuses on the One-Dimensional Bin Packing Problem (1D-BPP), a famous NP-hard combinatorial optimisation problem with several real-world applications. The objective is to pack a set of items with variable volumes into the fewest number of fixed-size bins possible while not surpassing each bin's maximum capacity. The primary challenge is to create an optimal or near-optimal packing design that minimises empty space and hence decreases storage and distribution expenses [1].

1.1 Problem Statement

Given n items with sizes $a_1, a_2, \ldots, a_n$, and a bin capacity V, the objective is to assign items into the minimal number of bins such that the total size of items in any bin does not exceed V. The problem can be mathematically formulated as follows:

$$\text{minimize } B = \sum_{i=1}^{n} y_i$$

$$\text{subject to } B \geq 1,$$

$$\sum_{j=1}^{n} a_j x_{ij} \leq V y_i, \forall i \in \{1, \ldots, n\}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j \in \{1, \ldots, n\}$$

$$y_i \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\} \forall j \in \{1, \ldots, n\}$$

where $y_i = 1$ if bin i is used and $x_{ij} = 1$ if item j is put into bin i.

While this problem appears straightforward, it is NP-hard, making accurate solutions computationally infeasible for large instances. The aim of this coursework is to create a metaheuristic or hyper-heuristic based algorithm that can provide high-quality solutions in a short period of time and is especially designed for inventory management use cases [5].

1.2 Context and Relevance

In modern supply chain and warehouse management systems, low space utilization frequently leads to higher overheads owing to the requirement for additional storage units or inefficient delivery configurations. Solving the 1D-BPP successfully can result in more efficient storage layouts, lower transportation costs, and better resource planning. Furthermore, this issue goes beyond theoretical interest. Its practical uses include cloud storage allocation, stock management in manufacturing, and load planning in logistics. Understanding instance difficulty, as defined by item size distributions or tightness ratios, enables algorithms to be adjusted or

hybridized for better performance [1]. This coursework bridges the gap between computational intelligence and operational efficiency by building algorithms that adapt to varying instance parameters, making it an advantageous activity in both academic and industry settings.

2 Methodology

2.1 Approach and Justification

To solve the 1D-BPP efficiently within time constraints, a hybrid technique was used, combining the speed of greedy heuristics with the optimization capabilities of a Genetic Algorithm (GA). The logic of this technique derives from the complementing qualities of these two types of methods. Heuristics such as First Fit (FF), First Fit Decreasing (FFD), Best Fit (BF), and Best Fit Decreasing (BFD) are widely used due to their speed and simplicity. These algorithms give quick and feasible responses that can be created in milliseconds, making them perfect for starting the search process. FFD and BFD are very effective in producing compact packaging by ordering items in decreasing size, a strategy known to decrease wasted space [6].

However, because of their myopic decision making, these greedy strategies frequently end up caught in local optima. To address this issue, a Genetic Algorithm is used to improve on the best heuristic solution discovered. GA was chosen because of its resilience to explore large solutions and its ability to produce high-quality solutions through iterative improvement. Unlike heuristics, GA maintains a population of potential solutions and evolves them over time using biologically inspired operators including selection, crossover, and mutation [2]. This enables it to identify non-trivial item orderings, resulting in more efficient bin utilization.

2.2 Implementation Details

The primary goal was to minimize the number of bins used across all instances while adhering to a strict five-minute computational time limit for 30 instances. The approach proposed was implemented in Python 3.10, taking use of the language's rich standard libraries and flexibility for quick algorithmic development. The program has a modular architecture with three main components: input/output processing, heuristic solvers, and genetic algorithm optimization. This architecture ensures maintainability while allowing for straightforward integration of additional algorithms in future iterations.

To begin, the input is imported from a JSON file containing several problem instances. Each instance has a list of item sizes and a bin capacity. The aim is to load every item in as few bins as possible while without exceeding any bin's capacity. The overall runtime budget of 300 seconds is distributed evenly across all instances, with each instance processed sequentially. This approach assures a fair allocation of computing resources while keeping the solution's overall efficiency within the time limitations.

The algorithm's initial phase uses multiple greedy heuristics to obtain baseline answers fast. These include Best-Fit (BF), Best-Fit Decreasing (FFD), and First-Fit (FF). The First-Fit heuristic successively inserts each item in the first bin with sufficient space, providing a quick and straightforward solution. First-Fit Decreasing improves on this by sorting items in decreasing order before implementing the First-Fit technique, which prioritises larger pieces for better packing outcomes. Similarly, Best-Fit places each item in the bin with the least available space, while Best-Fit Decreasing expands on this concept by sorting the things first. While these

algorithms are predictable and computationally efficient, they may not always find near-optimal solutions. However, they are valuable for establishing a quick baseline and generating diverse starting points for the metaheuristic phase.

To further optimize the solutions, a Genetic Algorithm (GA) was implemented using evolutionary operators specifically constructed for the 1D Bin Packing problem. The GA represents each potential solution as a permutation of item indices, rather than the items themselves. This permutation-based encoding enables the GA to investigate various packing sequences while ensuring that all items are uniquely included in the solution. Under greedy heuristics, the order in which items are packed has a substantial impact on the ultimate number of bins, hence this representation is well-suited to the problem.

The initial population consists of 30 individuals, developed using a combination of heuristic-guided and randomly generated permutations to achieve a balance between exploitation and exploration. Specifically, one individual is sorted in declining item size (emulating the First Fit Decreasing heuristic), one retains the original input order, and one is a completely randomized combination. Additional random shuffles are used to initialize the remaining individuals, increasing population diversity.

The parent solutions are selected using tournament selection with a tournament size of 5, which finds a compromise between pushing fitter individuals and conserving genetic variety in the population. Crossover is performed using the Order Crossover (OX) operator, which has an 85% probability. OX is especially useful for permutation-encoded solutions since it preserves significant subsequences and relative item order from both parents, hence enhancing offspring quality. Mutation cultivates diversity by randomly switching two components in the permutation with a fixed frequency of 0.2 (or 20%). This swap mutation prevents premature convergence by expanding the search space while keeping the permutations valid.

The fitness of each individual is assessed by decoding the permutation using the Best Fit (BF) heuristic, and the number of bins employed determines the fitness score to be minimized. Elitism is enforced by passing on the best-performing solution of the current generation to the next, ensuring that solution quality does not decrease with time. To avoid wasting computing resources on non-productive iterations, an early halting condition is implemented. If the best fitness does not improve for 20% of the allotted time each instance, the GA ends early.

Following completion, the GA's best-found solution is compared to those produced by the original greedy heuristics. The answer with the fewest bins is chosen as the final outcome in that case. All results, including the instance name, bin capacity, final bin configuration (a list of bins and their associated items), and execution time, are reported in a structured JSON format. This style guarantees that tests are reproducible and compatible with benchmarking tools.

3 Results and Analysis

3.1 Presentation of Results

```
Instance: instance_1    Mark: 3         Bonus: 0        Bins used/Best known: 52/52
Instance: instance_2    Mark: 3         Bonus: 0        Bins used/Best known: 59/59
Instance: instance_3    Mark: 3         Bonus: 0        Bins used/Best known: 24/24
Instance: instance_4    Mark: 3         Bonus: 0        Bins used/Best known: 27/27
Instance: instance_5    Mark: 3         Bonus: 0        Bins used/Best known: 47/47
Instance: instance_6    Mark: 3         Bonus: 0        Bins used/Best known: 49/49
Instance: instance_7    Mark: 3         Bonus: 0        Bins used/Best known: 36/36
Instance: instance_8    Mark: 3         Bonus: 0        Bins used/Best known: 52/52
Instance: instance_large_9      Mark: 3         Bonus: 0        Bins used/Best known: 417/417
Instance: instance_large_10     Mark: 3         Bonus: 3        Bins used/Best known: 373/375

--- Summary ---
Total Bin: 1136
Run Time:  65.06 s
Bonus mark: 3
Total mark: 33 / 30
Passed
```

The results of the 1D-BPP algorithm implementation are presented clearly and accurately, demonstrating the effectiveness of the hybrid approach combining greedy heuristics and a genetic algorithm. The results show that this approach successfully minimized bin usage while maintaining computational efficiency.

For most instances, the algorithm matched the best-known solutions exactly, achieving optimal performance in nine out of ten cases. Specifically, it used 52 bins for instance_1, 59 for instance_2, 24 for instance_3, 27 for instance_4, 47 for instance_5, 49 for instance_6, 36 for instance_7, 52 for instance_8, and 417 for instance_large_9 - all identical to the benchmark results. This consistency confirms the reliability of the hybrid approach in solving bin packing problems effectively.

Surprisingly, the approach exceeded the best-known solution for instance_large_10, using just 373 bins against the benchmark of 375 bins. This improvement, which represents a 0.5% decrease in bin utilization, highlights the genetic algorithm's capacity to develop improved solutions using its optimization capabilities. Across every instance, the algorithm packed items into 1,136 bins, which is two fewer than the best-known total of 1,138.

The computational efficiency of the approach was also excellent. With a total execution time of around 60 to 65 seconds for all instances, the method easily satisfied the 5-minute runtime restriction. This efficiency makes the solution suitable for real-world applications that require swift decision-making. The genetic algorithm's stagnation detection method helped substantially with this performance by reducing unnecessary computations once solutions had stopped improving.

3.2 Critical Analysis

The experimental results show significant differences in performance between several algorithmic methods to the 1D-BPP. My hybrid algorithm outperforms other approaches by deliberately integrating their strengths while limiting their respective faults. This analysis compares the performance of pure greedy heuristics, pure genetic algorithms, pure simulated annealing, and my hybrid approach across various dimensions.

Method	Bins Used	Runtime (s)	Bins vs Hybrid	Speed vs Hybrid
Pure Greedy	1148	0.1	+12 (+1.05%)	650× faster
Pure Simulated Annealing	1143	43.4	+7 (+0.62%)	1.5× slower
Pure Genetic Algorithm	1139	64.45	+3 (+0.26%)	1.0× (baseline)
My Hybrid	**1136**	**65.1**	**0**	**1.0×**

Pure greedy heuristics are the fastest approach, completing in 0.1 seconds, however this comes at a cost of solution quality. The greedy method employed 1148 bins, 12 more than my hybrid algorithm, resulting in a 1.05% increase in resource utilization. While this strategy shines in time-critical applications because to its O(n log n) complexity and deterministic nature, its inability to optimize beyond initial packing decisions renders it inappropriate for instances where bin utilization has a direct impact on operational costs [6]. The method's simplicity provides for easy implementation, but it severely limits its optimization potential because it cannot escape locally optimum solutions.

Using 1139 bins, Genetic Algorithms (GA) produce higher-quality solutions than greedy approaches or simulated annealing, while taking significantly longer to compute (64.45 seconds). The population-based approach of genetic algorithms provides powerful global search capabilities that assist avoid local optima, which is very useful in complex packing scenarios [2]. However, this incurs a large processing cost and requires careful parameter optimization. While pure GA surpasses greedy approaches by 0.79% in terms of solution quality, it falls short of my hybrid approach by 0.26%, despite having similar runtime requirements. The method's inherent parallelization offers the possibility of distributed computing implementations where hardware resources allow.

Simulated Annealing (SA) provides an intermediate solution, completing in 43.4 seconds with 1143 bins. This metaheuristic outperforms greedy techniques, but it falls short of both pure GA and my hybrid approach in terms of solution quality. While the method is theoretically guaranteed to find global optima given appropriate cooling schedules, in practice it is sensitive to parameter selection and performs poorly on problems with flat solution landscapes [4]. The 0.62% difference between simulated annealing and my hybrid approach indicates that, while SA is useful for medium-sized problems, it cannot match the sophisticated optimization capabilities of our combined methodology.

My hybrid algorithm's performance advantage is derived from its phased optimization technique. The initial greedy phase quickly delivers a high-quality solution (about 90% of final quality in only 1% of total runtime), which is then improved upon by subsequent genetic algorithm refinements. This division of labour enables each component to concentrate on its strengths, namely the greedy heuristic for speedy solution generation and the GA for accurate optimization. As a result, the strategy outperforms any pure approach in terms of solution quality while remaining computationally efficient.

4 Conclusion and Recommendations

4.1 Summary of Key Findings

This project successfully solved the 1D bin packing problem using a hybrid approach that used greedy heuristics with genetic optimization. The implementation performed admirably, matching the best-known solutions for nine of ten benchmark instances while outperforming the benchmark on the largest instance (instance_large_10) by employing 373 bins rather than the best-known 375. The algorithm's performance was particularly impressive, as it completed all computations in approximately 65 seconds while preserving solution quality. Key metrics revealed the hybrid approach's superiority over pure techniques; it enhanced greedy heuristics by 1.05% in bin utilization and produced more consistent results than standalone genetic algorithms. These results demonstrate the efficacy of combining speedy greedy initialization with extensive genetic optimization, especially for larger issue situations where pure approaches frequently fail.

4.2 Future Work and Recommendations

While the existing hybrid algorithm performs well, numerous significant modifications could expand its potential and practical applications. First, adaptive parameter control would enable the algorithm to automatically modify critical parameters such as mutation rate and population size during execution in response to real-time performance measurements [3]. This self-tuning capacity would improve the algorithm's robustness across diverse problem types while decreasing the requirement for manual configuration.

The algorithm could benefit from more hybridization by including local search strategies between genetic generations. Methods such as simulated annealing or tabu search may aid in escaping local optima and refining solutions, especially for tightly limited situations [7]. Furthermore, incorporating machine learning may allow predictive parameter optimization based on problem characteristics, thereby enhancing performance in specialized circumstances.

The successful implementation of these ideas would help to establish hybrid algorithms as powerful tools for solving complicated optimization problems in inventory management and beyond. Future work could yield even greater improvements in packing efficiency and operational cost savings by fine-tuning the balance between heuristic initialization and metaheuristic optimization, while also broadening the algorithm's applicability to a wider range of real-world logistics scenarios. This project's insights serve as a solid foundation for designing next-generation optimization algorithms capable of dealing with more complex supply chain difficulties.

References

[1] G. Carmona-Arroyo, Jenny Betsabé Vázquez-Aguirre, and M. Quiroz-Castellanos. "One-Dimensional Bin Packing Problem: An Experimental Study of Instances Difficulty and Algorithms Performance". In: *Studies in Computational Intelligence*. Springer, 2021, pp. 171–201. DOI: 10.1007/978-3-030-68776-2_10. URL: https://doi.org/10.1007/978-3-030-68776-2_10.

[2] S. A. Hussain and V. U. K. Sastry. "Application of genetic algorithm for bin packing". In: *International Journal of Computer Mathematics* 63.3-4 (1997), pp. 203–214. DOI: 10.1080/00207169708804561. URL: https://doi.org/10.1080/00207169708804561.

[3] M. A. Kaaouache and S. Bouamama. "Solving bin packing problem with a hybrid genetic algorithm for VM placement in cloud". In: *Procedia Computer Science* 60 (2015), pp. 1061–1069. DOI: 10.1016/j.procs.2015.08.151. URL: https://doi.org/10.1016/j.procs.2015.08.151.

[4] T. Kämpke. "Simulated annealing: Use of a new tool in bin packing". In: *Annals of Operations Research* 16.1 (1988), pp. 327–332. DOI: 10.1007/bf02283751. URL: https://doi.org/10.1007/bf02283751.

[5] C. Munien and A. E. Ezugwu. "Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications". In: *Journal of Intelligent Systems* 30.1 (2021), pp. 636–663. DOI: 10.1515/jisys-2020-0117. URL: https://doi.org/10.1515/jisys-2020-0117.

[6] S. V. *Mastering optimization*. Accessed: 2025-05-06. IBM Developer. 2025. URL: https://developer.ibm.com/articles/mastering-optimization/.

[7] Cagri Yesil, Hasan Turkyilmaz, and E. E. Korkmaz. "A new hybrid local search algorithm on Bin Packing problem". In: *2012 12th International Conference on Hybrid Intelligent Systems (HIS)*. IEEE, 2012, pp. 161–166. DOI: 10.1109/HIS.2012.6421327. URL: https://doi.org/10.1109/his.2012.6421327.