Optimisation Algorithms for One-Dimensional Bin Packing Problem: A Comparative Study

Anshana Manoharan
School of Computer Science
University of Nottingham Malaysia
hcyam4@nottingham.edu.my

Anshali Manoharan
School of Computer Science
University of Nottingham Malaysia
hcyam5@nottingham.edu.my

Aravindh Palaniguru
School of Computer Science
University of Nottingham Malaysia
hcyap1@nottingham.edu.my

Varsagasorraj A/L Vasagarajan
School of Computer Science
University of Nottingham Malaysia
hfyvv2@nottingham.edu.my

Abstract*—The Bin Packing Problem (BPP) is a timeless combinatorial optimization problem, which belongs to a special class of problems called NP-hard. It seeks to efficiently pack a set of items, each with its specified size, into identical bins with limited capacity. The goal is to minimize the number of bins needed to accommodate all items while ensuring that no bin exceeds its capacity constraint. This study provides an overview of the algorithms used over the years to solve the BPP. Additionally, we implement and analyse four algorithms: Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT), Simulated Annealing (SA), Minimum Bin Slack (MBS), and Hybrid Ant Colony Optimisation (ACO) with Iterated Local Search. A comparative study is carried out for each algorithm based on solution quality and execution time to identify the most ideal algorithm given their configurations.

Keywords*—One Dimensional Bin Packing Problem, 1DBPP, exact methods, approximate methods, GGA-CGT, ACO, MBS, SA, Grouping Genetic Algorithm with Controlled Gene Transmission, Hybrid Ant Colony Optimisation with Iterated Local Search, Minimum Bin Slack, Simulated Annealing, solution quality, execution time

I. Introduction

The Bin Packing Problem (BPP) is a combinatorial optimisation problem, which belongs to the NP-hard class [1]. The problem's structure and applications have been researched since the 1930s, while the classical version of BPP was explored in the early 1970s [2]. The Bin Packing Problem has numerous applications such as supply chain management, which involves vehicle, container, or cargo loading, as well as scheduling, resource allocation, financial budgeting and much more [3].

In this paper, we focus on the one-dimensional Bin Packing Problem (1DBPP), which can be defined formally as follows.

Given that we have an unlimited number of bins with a fixed capacity $c > 0$ and a set of n items, each with a specified weight $0 < w_i \leq c$, 1DBPP involves packing all the items into the minimum number of bins without surpassing the bin capacity.

Over the past years, researchers have developed algorithms to solve the BPP, ranging from exact and approximate algorithms to heuristics and metaheuristics [4].

This study provides a comprehensive overview of the algorithms utilised to solve the BPP over the recent years. Following an introduction to exact and approximate methods in Section II, Section III delineates our methodology for selecting four algorithms from the literature review and details the implementation of each. Section IV then presents the results of our practical implementation, and we critically evaluate the chosen algorithms, highlighting their strengths and weaknesses to conclude on the most optimum algorithm. Finally, the study is concluded with final remarks in the closing section.

II. Literature Review

Over the recent years, significant advancements have been made in developing algorithms to address the BPP. Classified as an NP-hard problem [1], it requires excessively high amounts of computation to construct optimal solutions, even for small-scale problem instances [5]. From the early to late 1990s, researchers proposed a plethora of exact methods, including column generation, reduction procedures, and branch-and-bound techniques [6], alongside approximate methods such as genetic algorithms and evolutionary programming [7].

A. Exact Methods

Introduced in 1971, the first exact method involving branch-and-bound for the BPP was proposed by Eilon et al. [8]. However, the algorithm could only solve problem instances of a very moderate size [9].

With the gradual development of heuristics, enhanced lower bounds and reduction procedures over the years [9], a more potent algorithm, Martello-Toth-Procedure (MTP) employing the branch-and-bound strategy [v] emerged in the 1990s. It represents the state-of-art for exact solution methods for BPP [10]. Nevertheless, the computational time required for large instances remains unsatisfactory [10].

Another existing exact method involves directly solving mixed zero-one integer linear programming (MZOILP) formulations of the BPP with general-purpose mathematical programming software like CPLEX. Although the method generates feasible solutions, it is unable to obtain optimal solutions within an admissible amount of computation time. [11]

Format adapted and modified from ©2023 IEEE.

Conclusively, the current body of literature on exact methods for solving the BPP remains limited, primarily due to the applicability being constrained to very small problem instances.

B. Approximate Methods

Approximate algorithms are frequently favoured for solving BPP instances due to their polynomial time complexity, even though they do not ensure optimal solutions. Approximate algorithms can be classified as either online or offline algorithms [2], this is shown in Fig. 1.

Online algorithms operate without prior knowledge of the entire input stream, adapting dynamically as items are presented sequentially. In contrast, offline algorithms have access to the complete set of input data, allowing them to analyse the entire problem space before making packing decisions [12].

Among the approximate algorithms are heuristics, notably First Fit (FF), Worst Fit (WF), Best Fit (BF), and Next Fit (NF), as discussed comprehensively in the work of Coffman et al. [13]. These online algorithms also have their corresponding offline algorithms, namely FFD (First Fit Decreasing), WFD (Worst Fit Decreasing), BFD (Best Fit Decreasing), and NFD (Next Fit Decreasing). The most effective among these methods is the FFD algorithm [7]. However, these offline algorithms do not consistently provide optimal solutions [2], but are frequently integrated into advanced strategies, such as generating initial solutions in evolutionary algorithms [14].

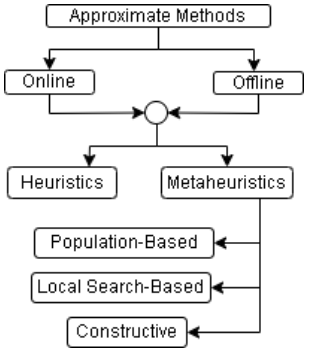

Fig 1. Classification of approximate methods for BPP.

Recently, a more improved heuristic had been introduced, which surpass the FFD and BFD algorithms by finding optimal and near-optimal solutions. Proposed by Gupta et al. [15], the heuristic known as the Minimum Bin Slack (MBS) has shown promise, occasionally delivering solutions comparable to those produced by a state-of-the-art algorithm [4]. Although various heuristic approaches have been proposed in existing literature, there have also been metaheuristic approaches inspired by natural phenomena that have captured considerable attention from researchers.

Metaheuristic algorithms achieve their final solution through different methods: constructive algorithms build solutions incrementally, population-based algorithms use evolutionary approaches to select the best solution from a population of solutions, and local search-based algorithms find the best solution by evaluating neighbouring solutions. This classification is depicted in Fig. 1.

An example is the ACO (Ant Colony Optimisation), which is a constructive metaheuristic algorithm which mimics the ability of ants to find the most efficient path between a food source and their nest [7]. Originally created to solve the Travelling Salesman Problem (TSP), it has since been adapted by researchers to solve the BPP as well [7].

Another approach is a population-based metaheuristic genetic algorithms (GAs), which saw significant enhancement through the adaptation into a grouping genetic algorithm (GGAs) by Falkenauer to suit grouping problems like the BPP [16]. Continued enhancements in grouping genetic algorithms culminated in a state-of-the-art algorithm developed by Quiroz et al. [14].

Local-search metaheuristic methods, such as Simulated Annealing (SA) [17] and Tabu Search (TS) [18], offer efficient optimisation strategies. SA, inspired by metallurgical annealing, surpasses the FFD heuristic [17]. Meanwhile, TS incorporates a "tabu list" to avoid revisiting recently explored solutions, enhancing its search efficiency.

III. METHODOLOGY

For the methodology of implementing algorithms to address the 1DBPP, we follow a structured approach. We begin by selecting suitable algorithms in Section A. Then, we show justification for chosen parameters and operators for the selected algorithms in Section B. Finally, the effectiveness of the selected algorithms in resolving BPP instances are assessed in Section C.

A. Algorithm Selection

The selection criteria for algorithms in existing literature for implementation encompass several key factors: ease of interpretation and implementation, optimality of solutions, ability to adapt to diverse problem instances, runtime efficiency, and the ability to balance exploration and exploitation within the solution space. The selected algorithms comprise Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT), Hybrid ACO with Iterated Local Search, SA and MBS. Their respective characteristics are summarised in Table 1 presented below.

TABLE I. CLASSIFICATION OF ALGORITHMS IMPLEMENTED

	Solution Strategy	**Search Strategy**	**Method**	**Type**
GGA-CGT	Improvement	Global	Stochastic	Meta-heuristic
SA	Improvement	Local	Stochastic	Meta-heuristic
Hybrid ACO with Iterated Local Search	Constructive	Local	Deterministic	Meta-heuristic
MBS	Constructive	Local	Deterministic	Heuristic

1) Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT)

GGA-CGT, a state-of-art grouping genetic algorithm proposed by Quiroz-Castellanos et al. [14] emerged as our first choice due to its impressive performance, as it outperformed the published results for the challenging class of instances Hard28, known to be of the greatest degree of difficulty for BPP algorithms [14].

The emphasis lies in passing on the best genes on chromosomes, maintaining equilibrium between selective pressure and population diversity. This approach fosters the emergence and refinement of superior solutions throughout the evolutionary process

Additionally, it utilises heuristic strategies that offer a balance between exploitation and exploration of the search space. GGA-CGT also avoids premature convergence, a common problem for genetic algorithms [19] and obtains better solutions with just a few generations [14]. Experimental results also show that GGA-CGT gives good results on diverse problem sets in terms of size and structure [14]. The implementation of this algorithm was guided by the pseudocode as shown below[14].

Algorithm 1 – Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT)

procedure GGA-CGT
1 generate an initial population *P* with FF-ñ heuristic
2 **while** *generation<max-generation and solution is not optimal* **do**
3 Select n_c individuals to crossover using controlled selection
4 Apply *gene-level crossover and FFD heuristic* to the n_c selected individuals
5 Apply *controlled replacement* to introduce offspring
6 Select n_m individuals and clone elite solutions using *controlled selection*
7 Apply *adaptive mutation and rearrangement of pairs* to the best n_m individuals
8 Apply *controlled replacement* to introduce clones
9 Update the global best solution
10 **end**

end GGA-CGT

1) Minimum Bin Slack (MBS)

The Minimum Bin Slack (MBS) is a bin-focused heuristic algorithm proposed by Gupta and Ho [15]. At each execution of the MBS, an attempt is made to find the best set of items that fit the bin's remaining capacity (slack).

Empirical evidence shows that MBS is superior in comparison to the widely used FFD and the BFD algorithms [15]. Despite the complexity of the MBS algorithm being $O(2^n)$, evidence demonstrates its practical efficiency in solving problem instances with varying parameters such as the number of items to be packed and the total number of items. If the maximum number of items to fit into a single bin is *u*, the computational complexity of the MBS procedure is effectively reduced to $O(n^u)$. Hence, the time taken to pack all the items results as $O(n^{(u+1)})$. This adaptation significantly mitigates the non-polynomial nature of the algorithm, enabling it to yield solutions within reasonable time frames.

A recursive implementation of the MBS algorithm proposed by Hindi et al. [20] is implemented in this study.

2) Simulated Annealing (SA)

Simulated Annealing (SA) is a probabilistic optimisation technique that iteratively adjusts the current solution to find the global optimum based on temperature. SA expands the search at higher temperatures to avoid local minima and refines solutions as the temperature decreases.

SA works effectively for problems of various sizes providing high scalability, including those involving thousands of items [11].

To implement SA, we have used the algorithm proposed by Brusco et al. [11]. SA utilises a form of morph lists called sets, exploiting the structure of matched sets of items. This specialised approach to item exchange within matched sets ensures efficient exploration of the solution space.

In the following pseudocode, Δ denotes the difference in the *z* value of the solutions before and after swap of items *r'* and *r''*, while a represents current solution. *z* represents the capacity of the heaviest bin of the current solution, and *z** represents best found value of *z* so far.

Algorithm 3 – Simulated Annealing (SA)

procedure SA
1 Generate initial solution a by randomly assigning items to bins. Calculate its minimax objective function value z*.
2 Set the best solution a*=a, z* = z, counter t_l = 0, and flag = 0. Choose 0 < cool < 1, templength, and initial temp.
3 From a randomly selected set t', pick an item r' in the heaviest bin b', and a different item r'' from another bin b''. Calculate Δ.
4 **if** Δ > 0, go to Step 6.
5 **if** Δ ≤ 0, accept swap, update a and z to its current values and go to Step 8.
6 Generate a random number rnd between 0 and 1.
7 **If** rnd < exp(−Δ/temp), accept the swap. Go to step 8.
8 **if** t_l < templength, t_l = t_l +1, go to Step 3. **else**, go to Step 9.
9 **if** flag=1, set flag = 0, temp = cool x temp, t_l = 0, and go to Step 3. **else**, **end**.
end SA

SA outperformed commercial software such as CPLEX by yielding solutions close to the lower bound for test problems with medium and large item sets [11]. This efficiency is crucial for handling large datasets, where computational time is a significant factor.

The probabilistic nature of SA allows it to escape local optima, giving it an edge in thoroughly exploring the search space which is particularly advantageous in complex and high-dimensional optimisation problems.

3) Hybrid Ant Colony Optimisation (ACO) with Local Search

ACO is a constructive metaheuristic inspired by the navigation behaviour of ants, who find the shortest path between their nest and food source using pheromone trails. The pheromone trails are updated based on the quality of the solutions found. Over time, better solutions reinforce stronger pheromone trails, guiding future ants toward more promising areas of the solution space. Through repeated iterations and the exchange of information via pheromone trails, the algorithm converges towards an optimal or near-optimal solution.

Levine et al. proposed a hybrid version that integrates iterated local search, revealing that this enhancement significantly boosts ACO's performance [7]. The pseudocode for this hybrid approach is as follows.

Algorithm 4 – Hybrid ACO with Iterated Local Search

```
procedure ACO
1    Initialise MAX_ITERATION, minimum pheromone
     threshold, pheromone constant k and evaporation
     rate
2    Create 2D array for pheromone values
3    Initialise pheromones with pheromone constant
4    Create object to store best solution
5    for MAX_ITERATION times do
6      Initialise number of bins with 0
7      for each item in items do
8       Extract item weights and counts
9       Sort remaining items in descending order
10        Create empty list for bins
11        repeat
12          Initialise current bin weight with 0
13          Create empty list for bin contents
14          for each item in remaining items do
15            if next item fits in current bin
16              Add next item weight to bin weight
17              Add item to bin list
18              Remove item from remaining list
19          Add bin to list of bins
20          Increment number of bins
21        until all items are packed
22        Calculate current solution fitness
23     if current solution fitness < best solution fitness
24        Set best solution to current solution
25        Increase pheromones on solution paths
26        Evaporate pheromones by evaporation rate
27        Ensure pheromones >= minimum pheromone
          threshold
28   return best solution
29 output best solution after MAX_ITERATIONS
30 end ACO
```

B. Parameter Choice and Operators

GGA-CGT was implemented using the parameters as published the authors [14], except for *max_gen*, which sets the maximum generations. Initially set at 500 by the authors for challenging problem instances, we optimised this parameter for our instances. Incremental tests were conducted, increasing *max_gen* until no further improvement was observed in the best solution's bin count, as illustrated in Fig. 2.

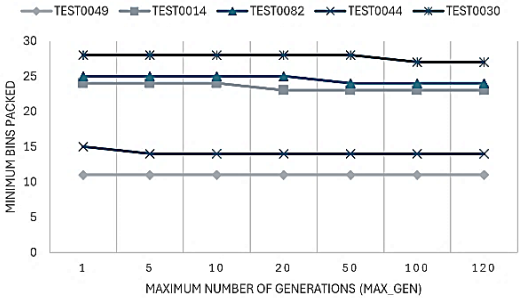

Fig 2. This line graph shows the parameter tuning session with the effects of varying *max_gen* on the minimum number of bins packed. The number of bins stopped improving after *max_gen* = 100.

The ACO algorithm has two parameters: the evaporation rate and the pheromone constant. The evaporation rate dictates the pace at which pheromones dissipate on trails, with higher rates facilitating rapid forgetting of suboptimal solutions. Conversely, lower rates promote broader solution exploration. Given the 1D nature of the BPP, evaporation rate variations have minimal impact. Thus, a rate of 0.5 was chosen to balance exploration and exploitation.

The pheromone constant adjusts the fitness evaluation process based on the fill ratio of bins. Values greater than 0.5 prioritise minimising the number of bins, while values smaller than 0.5 emphasise maximising the bin fill ratio. Like the evaporation rate, 0.5 was chosen to strike a middle ground, ensuring a balance between these objectives.

The MBS heuristic, lacking any adjustable parameters, requires no discussion in this context.

SA's parameters, like the cooling schedule and temperature length, can be adjusted to boost performance and achieve higher-quality solutions.

Using given problem-specific data to set the initial temperature has been proven to produce effective results [11]. After extensive testing, we determined that an initial temperature of 1000 and a final temperature of 10 provided the best outcomes, as evidenced by the data in Table II compared to lesser initial temperature of 100 and higher final temperature of 20.

This approach ensured an effective exploration of the solution space resulting in good z^* value. For cooling rate, we adopted the value of 0.9, as suggested by Brusco et al. [11]. This rate has proven to be well-suited for our simulations since lower cool rate yielded poorer z^* values as in Table 2.

Setting the templength to 200 as observed from Table 2, produced the most optimal z^* value compared to shorter templength ensuring thorough exploitation of the search space during search process. With all the chosen parameters, the z^* value resulted to be 9633,

which was a better *z** value than any other combination tested.

TABLE II. OBSERVED Z* VALUES FOR TEST0049

Test	Initial Temp	Final Temp	Temp length	Cool Rate	Sets	Z*
1	100	20	50	0.7	17	9897
2	100	20	50	0.7	13	9810
3	100	20	50	0.9	13	9793
4	100	20	200	0.7	13	9722
5	100	10	50	0.7	13	9807
6	**1000**	**10**	**200**	**0.9**	**13**	**9633**
7	1000	20	50	0.7	17	9996
8	1000	10	50	0.7	13	9660
9	1000	20	200	0.7	13	9785
10	1000	20	50	0.9	13	9662

C. Performance Evaluation

The evaluation of the four algorithms encompasses two key criteria: solution quality and execution time. Solution quality is assessed using the approach outlined by Quiroz et al. [4], employing the cost function (1) introduced by Falkenauer et al [21]. This function measures the effectiveness of the algorithm by calculating the average fullness of the bins in the solution.

$$F_{BPP} = \frac{\sum_{j=1}^{m}(l(j))^2}{m} \quad (1)$$

F_{BPP} is computed based on the number of bins *m* in the solution and the accumulated capacity *l(j)* of each bin *j*. Additionally, we measure the execution time of each algorithm. This measurement is taken in hundredths of a nanosecond *(ns/100)*.

IV. Result and Discussion

The experiments in this study were performed on an 11th Gen Intel(R) Core(TM) i3-1115G4 processor with a clock speed of 3.0 GHz and 4GB RAM. The algorithms are applied to a dataset with five problem instances; TEST0049, TEST0014, TEST0082, TEST0044, TEST0030. All the algorithms were implemented in Java and compiled using the IntelliJ IDE.

A. Execution time evaluation

Each algorithm was run 30 times per problem instance and their average computational time was calculated. As depicted in Fig. 3, GGA-CGT emerges as the most computationally demanding algorithm in our analysis.

GGA-CGT's extended runtime is due to its ambitious convergence criteria, halting only when the best solution size matches a calculated lower bound or after 100 generations. Conversely, the hybrid ACO and MBS achieve lower execution times owing to their constructive approach. SA's simplicity and minimal memory usage contribute to its relatively shorter computational time.

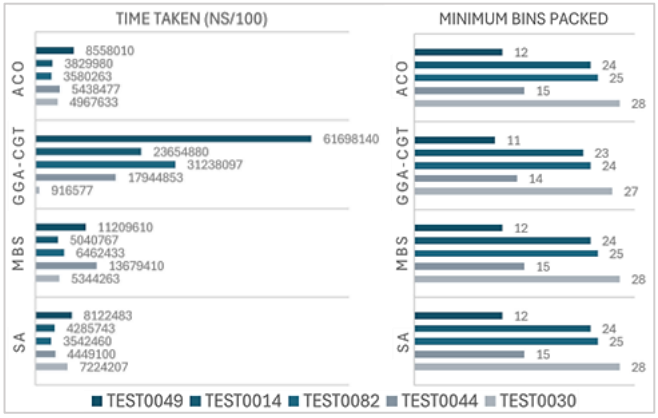

Fig 3. This figure shows bar charts of the average execution time and the minimum bins packed per algorithm for each problem instance.

B. Solution quality and variability

The assessment of solution quality involves computing the average F_{BPP} value across all problem instances for each algorithm. Upon reaching a F_{BPP} value of 1, it indicates the attainment of optimal solutions across all problem instances [4].

Table 3 highlights GGA-CGT as the algorithm closest to achieving optimal solutions across all problem instances. This indicates exceptionally high average bin fullness, which can be attributed to its use of the 'arrangement of pairs' technique. This method, outlined in the pseudocode, enhances the genetic material's quality during adaptive mutation, contributing to superior solution quality [4].

TABLE III. AVERAGE F_{BPP} VALUE FOR EACH ALGORITHM

Algorithm Type	Average F_{BPP} Value
SA	0.909
Hybrid ACO with Iterated Local Search	0.937
MBS	0.940
GGA-CGT	**0.986**

The bar chart on the right in Fig. 3. shows the minimum number of bins achieved by each algorithm after 30 runs.

The hybrid ACO approach, MBS and SA all achieve the same minimum number of bins. Conversely, GGA-CGT exhibits a slight improvement, achieving one fewer bin. This could be due to its ability to achieve a higher average bin fullness, (as indicated by its near-optimal F_{BPP} value) it achieves superior space utilisation.

It is likely that MBS and the hybrid ACO approach end up in suboptimal solutions since they are constructive algorithms and have limited exploration abilities. SA on the other hand, if its initial temperature value was too low, it could have restricted its ability to explore alternative solutions effectively, potentially leading to suboptimal outcomes.

In addition to the comparison the solution quality and execution time of the algorithms, the solution variety produced during the 30 runs were also investigated. The hybrid ACO approach and the MBS algorithm were excluded due to their deterministic nature. The results obtained are presented in Fig.4.

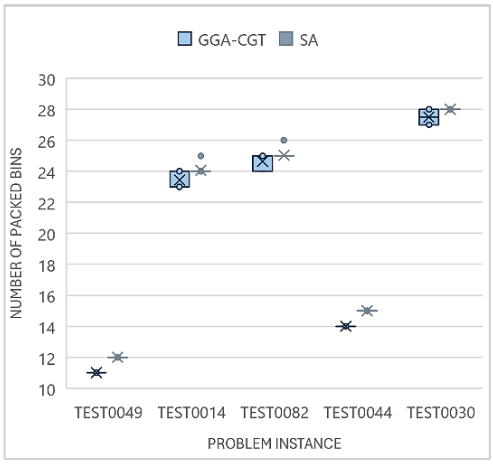

Fig 4. This box plot shows the range of solutions generated by GGA-CGT and SA during the 30 runs.

In comparison to GGA-CGT, SA demonstrates higher consistency in its solutions. While GGA-CGT may achieve superior quality solutions across the 30 runs, it is notable that for 60% of the instances, there remains a trivial variation in the solutions, differing by a single bin. SA varies similarly in 40% of the instances, however, when it does, it provides comparatively worser solutions. Conclusively, SA provides a more consistent performance in comparison to GGA-CGT, but provides suboptimal solutions. Meanwhile GGA-CGT has a higher variability in its solutions but reaches near-optimal solutions.

CONCLUSION

When choosing the optimal algorithm based on the minimum number of bins packed and execution time, determining a clear preference can be challenging. GGA-CGT is far superior in terms of solution quality and can be concluded as the most optimal of the four algorithms. Despite GGA-CGT showing relatively poorer performance in terms of execution times, it's noteworthy that it still operated within the same time range, measured in hundredths of nanoseconds. This indicates that while there may be variations in performance, the differences are negligible at such a fine granularity of time measurement. Thus, when considering the efficacy of algorithms in bin packing scenarios, factors beyond such execution times should be considered to ascertain their true effectiveness. In this case, looking at the F_{BPP} value.

Although GGA-CGT is superior to the rest of the algorithms implemented, these algorithms are still able to reach similar solutions with an impressive single offset of the bin number in comparatively shorter amounts of time. The performance of these algorithms can be further evaluated with different bin capacities and more diverse problem instances.

REFERENCES

[1] M. R. Garey, D. S. Johnson, and Others, ‘A Guide to the Theory of NP-Completeness’, Computers and intractability, pp. 37–79, 1990.

[2] C. Munien and A. E. Ezugwu, ‘Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications’, Journal of Intelligent Systems, vol. 30, no. 1, pp. 636–663, 2021.

[3] U. Elııyı and D. T. Elııyı, ‘Applications of bin packing models through the supply chain’, International Journal of Business and Management Studies, vol. 1, no. 1, pp. 11–19, 2009.

[4] G. Carmona-Arroyo, J. B. Vázquez-Aguirre, and M. Quiroz-Castellanos, ‘One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance’, Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications, pp. 171–201, 2021.

[5] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, ‘Worst-case performance bounds for simple one-dimensional packing algorithms’, SIAM Journal on computing, vol. 3, no. 4, pp. 299–325, 1974.

[6] A. C. F. Alvim, C. C. Ribeiro, F. Glover, and D. J. Aloise, ‘A hybrid improvement heuristic for the one-dimensional bin packing problem’, Journal of heuristics, vol. 10, pp. 205–229, 2004.

[7] J. Levine and F. Ducatelle, ‘Ant colony optimization and local search for bin packing and cutting stock problems’, Journal of the Operational Research society, vol. 55, pp. 705–716, 2004.

[8] S. Eilon and N. Christofides, ‘The loading problem’, Management Science, vol. 17, no. 5, pp. 259–268, 1971.

[9] M. Delorme, M. Iori, and S. Martello, ‘Bin packing and cutting stock problems: Mathematical models and exact algorithms’, European Journal of Operational Research, vol. 255, no. 1, pp. 1–20, 2016.

[10] P. Schwerin and G. Wäscher, ‘The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP’, International transactions in operational research, vol. 4, no. 5–6, pp. 377–389, 1997.

[11] M. J. Brusco, H. F. Köhn, and D. Steinley, ‘Exact and approximate methods for a one-dimensional minimax bin-packing problem’, Annals of Operations Research, vol. 206, pp. 611–626, 2013.

[12] B. Kroth, ‘CS787 Project: Bin Packing: A Survey and its Applications to Job Assignment and Machine Allocation’, 2013.

[13] E. G. Coffman Jr, J. Y.-T. Leung, and J. Csirik, ‘Variants of Classical One-Dimensional Bin Packing’. 2007.

[14] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimenez, C. Gómez, H. J. F. Huacuja, and A. C. F. Alvim, ‘A grouping genetic algorithm with controlled gene transmission for the bin packing problem’, Computers & Operations Research, vol. 55, pp. 52–64, 2015.

[15] J. N. D. Gupta and J. C. Ho, ‘A new heuristic algorithm for the one-dimensional bin-packing problem’, Production planning & control, vol. 10, no. 6, pp. 598–603, 1999.

[16] E. Falkenauer, ‘A hybrid grouping genetic algorithm for bin packing’, Journal of heuristics, vol. 2, pp. 5–30, 1996.

[17] Y. Wu, M. Tang, and W. Fraser, ‘A simulated annealing algorithm for energy efficient virtual machine placement’, in 2012 IEEE international conference on systems, man, and cybernetics (SMC), 2012, pp. 1245–1250.

[18] J. El Hayek, A. Moukrim, and S. Negre, ‘A TABU SEARCH METHOD FOR THE NON-ORIENTED TWO-DIMENSIONAL BIN-PACKING PROBLEM’, IFAC Proceedings Volumes, vol. 39, no. 3, pp. 487–492, 2006.

[19] S. Malik and S. Wadhwa, ‘Preventing premature convergence in genetic algorithm using DGCA and elitist technique’, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 6, 2014.

[20] K. Fleszar and K. S. Hindi, ‘New heuristics for one-dimensional bin-packing’, Computers & operations research, vol. 29, no. 7, pp. 821–839, 2002.

[21] E. Falkenauer, A. Delchambre, and Others, ‘A genetic algorithm for bin packing and line balancing’, in ICRA, 1992, pp. 1186–1192.