Algorithms for 1D-BPP: A Study

Joseph Emmanuel Chay Huan Qin
School of Computer Science
University of Nottingham Malaysia
hcyjc11@nottingham.edu.my

Lua Chong En
School of Computer Science
University of Nottingham Malaysia
hcycl5@nottingham.edu.my

Muhammad Fadil Amin Bin Arsani
School of Computer Science
hcyma9@nottingham.edu.my

Mubashir Mohd Shoukat
School of Computer Science
University of Nottingham Malaysia
edymm2@nottingham.edu.my

Abstract

The Bin Packing Problem (BPP), a classic NP-hard challenge, seeks to efficiently pack items to minimize wasted space. This study addresses the offline version of the One-Dimensional BPP (1D-BPP), where item weights are known in advance, allowing rearrangement before processing. Solutions are approximated using heuristic or metaheuristic approaches. In this paper, we provide an overview of 1D-BPP-solving algorithms, implement, and evaluate four of them: First Fit Decreasing (FFD), Simulated Annealing (SA), Genetic Algorithm (GA), and Tabu Search (TS). An analysis of their performance in terms of solution optimality and computation time is also conducted.

Keywords— Bin Packing Problem, 1-Dimensional BPP, First Fit Decreasing, Simulated Annealing, Genetic Algorithm, Tabu Search, Computation Time, Run Time, Convergence.

I. Introduction

The 1D-BPP is a classic optimization problem that involves packing a set of items of varying sizes into a minimum number of bins, where each bin has a fixed capacity. Initially proposed in the 1960s [1][2], the problem has since garnered significant attention in both academic and industrial contexts due to its relevance in logistics, manufacturing [3], and even in healthcare industries [4].

At its core, the 1D-BPP aims to find an optimal or near-optimal packing arrangement to minimize the number of bins used while ensuring that the total size of items in each bin does not exceed its capacity. Various solutions have been developed over the years, including approximate heuristic algorithms such as First Fit Decreasing (FFD) and Best Fit Decreasing (BFD), as well as metaheuristic methods like Tabu search, genetic algorithms [5] and simulated annealing [6]. Recent research has also explored hybrid approaches combining exact and heuristic techniques to achieve improved performance [7].

This paper aims to provide an overview of solution methods for the 1D-BPP, implement and evaluate the effectiveness of selected algorithms, and propose novel hybrid approaches to tackle the problem efficiently.

II. Literature Review

The classical one-dimensional bin packing problem aims to minimize the number of bins, m required to pack an n list of items, $L = (p_1, p_2, \dots, p_n)$ where each item p_i has a size of $s(p_i)$ that must satisfy $0 \leq s(p_i) < C$, and C represents the fixed capacity of each bin. The objective is to partition the items into subsets $B_1, B_2, \dots, B_m$, such that the sum of the sizes of items in each subset B_j does not exceed the capacity C. Each subset B_j is viewed as the contents of a bin, and the packing arrangement must ensure efficient space utilization while minimizing the total number of bins used [8].

In 1990, Martello [9] has represented the problem as a mathematical model, where, let *u* be an upper bound on the minimum number of bins required to pack all the items. Assume that these bins are numbered as 1, 2,..., *u*. Let *n* be the number of items to pack. Two binary decision variables are introduced:

$$y_i = \begin{cases} 1 & if\ bin\ i\ is\ used\ in\ solution; \\ 0 & otherwise;\ \ 1 \leq i \leq u \end{cases} \quad (1)$$

$$x_{ij} = \begin{cases} 1 & if\ item\ j\ is\ packed\ into\ bin\ i; \\ 0 & otherwise\ ;\ 1 \leq i \leq u; 1 \leq j \leq n \end{cases} \quad (2)$$

Then,

Minimize $\sum_{i=1}^{u} y_i$ (3)

s.t $\sum_{i=1}^{u} w_j x_{ij} \leq c y_i;\ \ 1 \leq i \leq u$ (4)

$$\sum_{i=1}^{u} x_{ij} = 1\ ; 1 \leq j \leq n$$
$$y_i \in 0,1\ ;\ 1 \leq j \leq n$$
$$x_{ij} \in 0,1; 1 \leq i \leq u; 1 \leq j \leq n \quad (5)$$

The constraints enforce that the maximum capacity of each bin must not be exceeded and that each item in the list must only be packed into one bin. The goal of this optimization problem is to minimize the number of bins, N, that can be mathematically expressed as follows:

$$N \geq \left\lceil \frac{(\sum_{i=1}^{n} s_i)}{C} \right\rceil \quad (6)$$

where *Si* represents the weight, of each item in the list of *n* items, and *C* represents the fixed capacity of the bins.

A. Heuristic Approaches

Expanding on Martello's formulation, several heuristic approaches such as First Fit Decreasing (FFD) has been used to solve the problem. This approach aims to find a satisfactory solution within a reasonable time frame without necessarily guaranteeing optimality.

FFD operates by first rearranging the item list by weight in decreasing order. The items is then placed into the first available bin that can accommodate them [10].

B. Metaheuristics Approaches

Heuristic approaches often exploit only a portion of the solution space, focusing on local solutions rather than exhaustively searching all possibilities. However, this approach risks getting trapped in suboptimal solutions known as local optima. To overcome this limitation, metaheuristic techniques are employed by introducing an escaping mechanism to further explore the other solution space, enabling the search to escape local optimum and converge towards more globally optimal solutions.

The metaheuristic approach for our 1D-BPP falls into two categories, Local Search (LS) or Population Based Search (PBS).

LS metaheuristics focus on exploring the solution space by iteratively improving a single candidate solution through small incremental changes [11]. This approach aims to find a locally optimal solution within a reasonable computational time frame but may not guarantee the global optimum. Examples of local search metaheuristics include Simulated Annealing (SA) and Tabu Search (TS) with the former, inspired by metallurgy, gradually reduces the acceptance of worse solutions over iterations through a process known as temperature cooling, allowing exploration of broader solution spaces [6], while the latter maintains a list of already explored solutions to encourage exploration of diverse region and avoid cycling between the same candidate of solutions [12] .

Meanwhile, PBS metaheuristics maintain a population of candidate solutions and iteratively evolve them to find the optimal or near-optimal solutions. Operations for this kind of search involves selection, crossover, and mutation to generate new candidate solutions from the existing populations. One of the foundational algorithms for this kind of search is Genetic Algorithm (GA). In 1975, John Holland [13] introduced GA which mimics the process of natural selection and evolution. Through successive generations, individuals in the population undergo selection, crossover (recombination), and mutation operations to produce offspring with potentially improved fitness values. The process iterates until a termination criterion is met, such as reaching a maximum number of generations or achieving a satisfactory solution.

By incorporating escaping mechanisms such as temperature cooling in SA, Tabu List in TABU and maintaining a diverse population in GA, metaheuristic techniques effectively explore a broader solution space, avoids local optimum and has a better chance to converge towards optimal or near-optimal solutions.

III. METHODOLOGY

The methodology for implementation of algorithms to solve the 1D-BPP consists of a) selecting suitable algorithms b) justifying the chosen parameters and operators c) evaluating effectiveness of each algorithm.

A. Algorithm Choice

For any given 1D-BPP instance, there are 2 variables that are fed into the algorithms. It is the *items* and the *binCapacity*. Therefore, each algorithm must be able to accept these 2 parameters and operate on multiple test cases. When considering the algorithms to implement, the factors that were considered were: efficiency in packing items into bins, ability to escape local optima and explore global optima, optimality of solution and time taken to get to solution. The chosen algorithms are First Fit Decreasing (FFD), Simulated Annealing (SA), Genetic Algorithm (GA), and Tabu Search (TS).

TABLE I. CLASSIFICATION OF ALGORITHMS IMPLEMENTED

	Algorithm Type	**Method**	**Search Strategy**
FFD	Heuristic	Greedy	Non-Optimised Search
SA	Meta-heuristic	Stochastic	Local Search
GA	Meta-heuristic	Evolutionary	Population-based Search
TABU	Meta-heuristic	Memory-based	Iterative Improvement

1. First Fit Decreasing

FFD for 1D BPP is a greedy heuristic algorithm that is non-optimised. It operates by sorting items in a nonincreasing order based on the item weights [10]. Then sequentially packs each item into the first bin that it fits, always starting from the first bin. If the item does not fit in any existing bin, a new bin is opened and the item is packed into it. The pseudocode for the implementation used in this study is given below.

Algorithm 1 – First Fit Decreasing Algorithm

```
sort items in descending order by weight.
initialize an empty list of bins.
        for each item
                place the item in first available bin
                if cannot fit, create a new bin
                        add item to new bin
                        append new bin to list
```

```
    end for
return number of bins
```

FFD is chosen for multiple reasons: ease of implementation, low computational complexity, and the ability to reach near-optimal results [10]. The worst case performance bounds of the FFD algorithm is 11/9 with the additive constant of 6/9 [14]. Therefore, FFD algorithm uses at most 11/9 times as many bins compared to the optimal algorithm and that even in the worst-case, FFD will use at most 6/9 times more bins regardless of the total number of bins [15]. Furthermore, FFD has been regarded as one of the better performing algorithms by minimising the number of bins required [20].

2. Simulated Annealing

SA for 1D BPP is stochastic randomised search meta-heuristic algorithm. Simulated Annealing is also known as statistical cooling, where the approach involves gradually guiding a combinatorial system towards configurations of globally minimal energy [16]. At each iteration, the system is perturbed through a set of conditions. The temperature is lowered slowly to maintain *thermal equilibrium*, and the probability can be shown as follows [6][17]:

$$Prob(E + dE) = Prob(E)e - \frac{dE}{kT} \qquad (7)$$

The pseudocode for the implemented SA is described in Algorithm 2.

Algorithm 2 – Simulated Annealing

```
generate a random candidate solution
set the temperature
while T > T0
    repeat
    if T > threshold
        generate C1
    else
        generate C2
        ΔC = C(C') – C(C)
        if ΔC < 0
            C ← C1 || C2
        else if η < e-ΔC/T
            C ← C1 || C2
        else
            continue
    until thermal equilibrium reached
    apply cooling factor
end while
return C
```

Unlike FFD, SA is able to explore and exploit the solution space, converging to a near-optimal solution [17]. Having the ability to accept solutions that degrade the objective function at high temperatures, thereby exploring the solution space. As the temperature decreases, using candidate solutions that improve the objective function. Additionally, SA is effective at classic NP-hard combinatorial optimisation problems [19].

3. Genetic Algorithm

GA uses mechanisms including: selection, crossover, and mutation. It's initial population is generated using FF which then evolves through successive generations via mutation. Each generation of a new population is created to seek for a better population hence improved results. The solution is determined from selecting individuals based on fitness, then applying crossover and mutation to produce an offspring that is added into the population, therefore the population can contain better quality solutions [21]. The iterative process continues to refine the population over time, leading to improved results [20]. The pseudocode for the implemented GA is described in Algorithm 3.

Algorithm 3 – Genetic Algorithm

```
Initialise population
repeat until termination
    create new population
    evaluate fitness of each individual in population

    repeat until termination
        perform selection using tournament selection
        perform crossover to produce offspring
        apply mutation to the offspring
        add offspring to newly created population
    new population ← old population
return best individual in population
```

In contrast with SA and FFD for solving the one-dimensional bin packing problem. GAs utilises a population-based approach, allowing simultaneous exploration of multiple solutions, which significantly enhances the algorithm's ability to avoid local optimum and find more globally optimal solutions [20]. This contrasts with SA, which incrementally improves a single solution and may get trapped in local optimum. Unlike FFD, a greedy heuristic that quickly reaches a solution by placing items into the first suitable bin, GAs apply crossover and mutation across a diverse set of solutions, thus providing a mechanism to escape suboptimal configurations [20][21].

4. Tabu Search

TS is a memory-based iterative meta-heuristic, with the ability to efficiently explore and exploit the solution space by maintaining a tabu list [22]. The tabu list record recent moves to avoid revisiting them, therefore allowing the algorithm to escape local optimum. TS operates by iteratively moving from one solution to a neighbouring solution, thereby balancing exploration and exploitation. The pseudocode for the implemented TS is described in Algorithm 4.

Algorithm 4 – Tabu Search Algorithm

```
Begin
    generate initial solution
    for all client numbers do
        find nearest neighbor
        for max iterations
            update tabu list
            generate new solution s'
```

```
            calculate cost of S_current
            value = evaluate(S_current)
            if (value < s_best)
                  s_best = value
      end for
      update tabu list
end
```

TS is able to achieve near-optimal results for NP-Hard problems [22] with short running time [23]. Unlike SA, TS converges more aggressively to local optimum which contributes to making the best available move at each iteration and devoting more effort to exploring regions of good solutions [23]. Additionally, TS uses deterministic moves, reduces the distribution of results due to the initial solution or presence of parameters.

B. Operator and Parameter Choice

For SA, the standard annealing algorithm and annealing schedule is implemented [16]. The initial solution is chosen at random. Additionally, the code implements two different candidate configurations. The first candidate configuration is randomly selecting a bin, then selecting an item from the bin and placing the selected item into another randomly selected bin, this is operated at higher temperatures. The second candidate configuration is randomly selecting two different bins and exchanging positions of a randomly selected item from each bin, which is operated at lower temperatures. The main purpose is coarsely optimising the allocation of items in bins at higher temperatures, then making finer adjustments at lower temperatures to achieve a better objective function [19]. By using two different types of candidate solutions, SA explores different parts of the solution space [19], leading to exploration and exploitation and the ability to escape local optimum [18].

Temperature is set to 100 and T_0 (target temperature) is set to 0.1, where F (cooling factor) is 0.95. Initialising the temperature at a high value allows the SA algorithm to escape the local optimum and explore the solution space efficiently. Setting T_0 at a low value of 0.1, indicate that the algorithm converges to a near-optimal solution by taking smaller steps in every iteration. Finally setting F to 0.95, in between the suggested interval of 0.9-1.0 [17]. The temperature threshold of 65 for determining the candidate configuration was chosen after trial and error that yielded near-optimal results.

TABLE II. PARAMETERS OF GENETIC ALGORITHM

Parameter	**Value**
Population Size	500
Generations	50
K	2
Tournament Selection Rate	0.8
Crossover Probability	0.5
Mutation Probability	0.5

Genetic Algorithm (GA) parameters were determined through empirical trials. Population sizes ranging from 100 to 10000 were tested, with 500 identified as optimal. Convergence typically occurred within 50 generations. Tournament selection with a K value of 2 and a selection rate of 0.8 showed the best performance. Mutation and crossover rates from 0 to 1 were tested, with the most effective strategy involving half of each generation using mutation and the other half using crossover.

TABLE III. PARAMETERS FOR TABU SEARCH ALGORITHM

Parameter	**Value**
Max Combination Lengths	15
Max Iterations	2000
Max No Change	500

Testing TS parameters involved exploring ranges: Max Combination Lengths (10-20), Max Iterations (1000-3000), and Max No Change (200-800). Optimal settings were found: 15 for Max Combination Length, 2000 for Max Iterations, and 500 for Max No Change. Lower Max Combination Lengths limited exploration capabilities, while higher values extended computational time. Max Iterations at 2000 provided balance for exploration and exploitation, while Max No Change at 500 optimized stopping criteria, avoiding premature termination or unnecessary search prolongation.

IV. RESULT AND DISCUSSION

The simulations in this study were conducted on a 13th Generation Intel® Core™ i7-13700H Processor (24 MB Cache, 14 cores, 20 threads, with a base clock speed of 2.40 GHz and a turbo boost of up to 5.00 GHz) and 64 GB of DDR5 RAM, running at 4800 MT/s in dual channel configuration. All algorithms are developed in Java and compiled using the IntelliJ IDE.

A. Optimality of Algorithm Choice

The parameter settings were configured, and each algorithm was run 30 times to calculate the average minimum number of bins required.

Solution optimality was determined based on both the minimum number of bins and a minimal total amount of wasted space across all bins for each dataset. According to the results depicted in Fig 1 and Fig 4, Genetic Algorithm stands out as the only method capable of reaching the optimal solution for TEST0049 and TEST0044.

Genetic Algorithm demonstrated favourable performance in handling TEST0044 and TEST0049, achieving optimal solutions of 14 bins and 11 bins, respectively. The space within the bins was utilized efficiently, resulting in minimal wasted space across all

bins, totalling of an average of approximately 73 spaces left for TEST0044 and 60 spaces left for TEST0049. Conversely, FFD, Simulated Annealing, and Tabu Search fell short, only reaching solutions that were suboptimal for these datasets. The total wasted space relative for these algorithms for TEST0044 amounted to 9953 and 127 excluding the outlier, and for TEST0049, it was approximately 10,060 and 180 excluding the outlier.

Across other datasets, all algorithms performed identical with only being able to reach suboptimal solutions, requiring one additional bin on average compared to the ideal outcome.

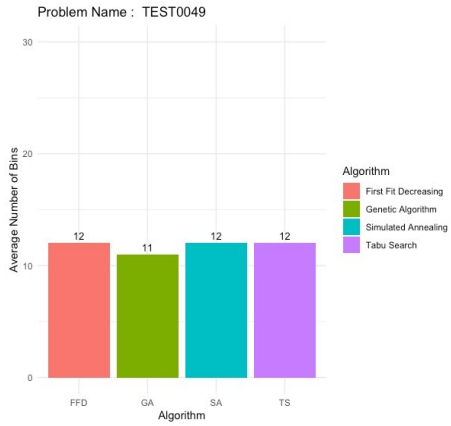

Fig 1. Average number of bins for the algorithms in TEST0049

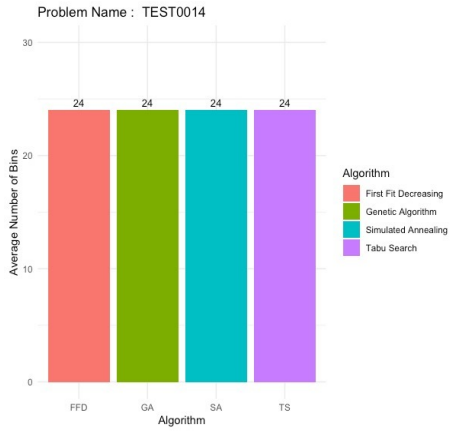

Fig 2. Average number of bins for the algorithms in TEST0014

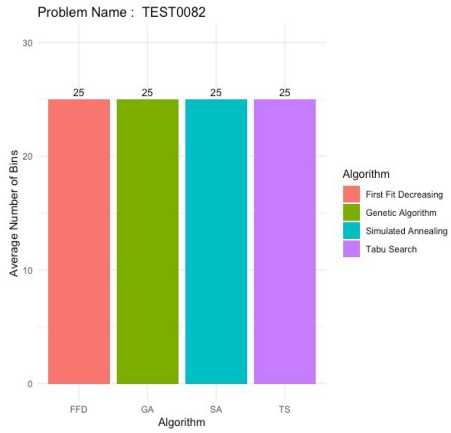

Fig 3. Average number of bins for the algorithms in TEST0082

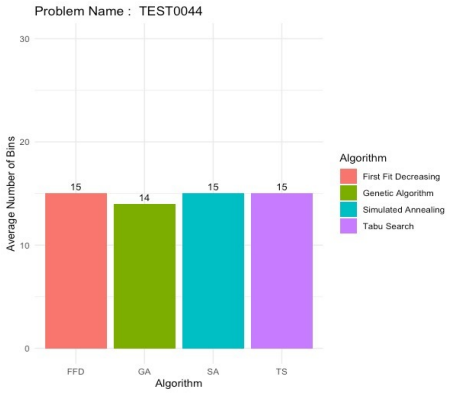

Fig 4. Average number of bins for the algorithms in TEST0044

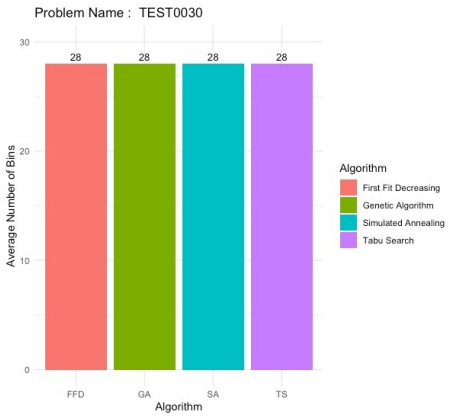

Fig 5. Average number of bins for the algorithms in TEST0030

B. Runtime Characteristics

In our study aimed at determining the minimum number of bins required, each algorithm underwent 30 runs to capture their respective runtime distributions, as visually presented in Figure Number. Among these algorithms, the First Fit Decreasing (FFD) heuristic emerged as the standout performer, showcasing an average runtime of approximately ~0.1 milliseconds.

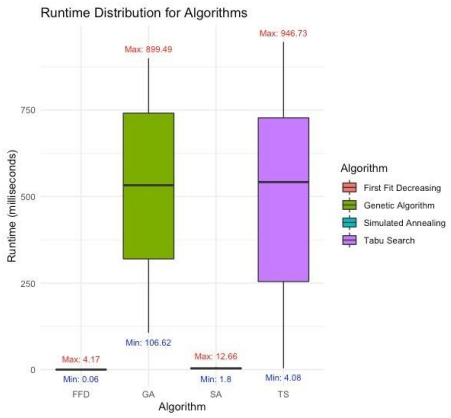

Fig 6. Average runtime (milliseconds) for all algorithms

This result surpassed both the GA, averaging around ~532 milliseconds and Tabu Search (TS), averaging around ~542 milliseconds. Following FFD closely behind was Simulated Annealing (SA), which

demonstrated an average runtime of approximately ~3.52 milliseconds.

These results are consistent with the methodology employed. The inherent randomness of TS in both its initial solution setting and move selection contributes to the wide variation in its runtime distribution with the maximum runtime at ~947 milliseconds and minimum runtime at ~4.1 milliseconds. Additionally, this randomness can lead to a slower convergence rate compared to GA when seeking optimal solutions.

On the other hand, our implementation of GA deploys an underlying heuristic namely FF which helps in setting up the initial solution. Integrating a heuristic will allow for a high-quality initial solution which was able to help accelerate the convergence rate. In contrast to TS, its randomness in the initial solution is usually of lower quality which might lead to it being trapped in local optima due to its reliance on local search and Tabu restrictions.

There is a notable difference between FFD and SA due to SA's reliance on an iterative exploration strategy that involves a gradual traversal of the solution space, allowing for comprehensive exploration but potentially leading to prolonged convergence. In contrast, FFD's deterministic approach favours swift decision-making based on predefined criteria, i.e. sorting items in decreasing order first and then packing them into the first available bin, thereby often resulting in faster convergence.

CONCLUSION

This study examines the effectiveness of four commonly used heuristic and metaheuristic algorithms in solving the 1D-BPP by assessing their strengths and limitations. Among these algorithms, GA stood out as the only one capable of achieving an optimal solution for two out of the five datasets. This success can be attributed to its balanced approach between exploration and exploitation, particularly when a parameter is fine-tuned. On the other hand, FFD closely followed GA by consistently approaching near-optimal solutions across all datasets, at a faster convergence rate. In summary, the choice of the most suitable algorithm boils down to a trade-off between GA with its integrated FF heuristic, which prioritizes optimal solutions, and a standalone FFD heuristic, which offers quicker computational times suitable for real-time applications where slight suboptimality is acceptable.

REFERENCES

[1] Coffman, E. G., Jr., Garey, M. R., & Johnson, D. S. (1984). Approximation algorithms for bin packing: A survey. In Approximation algorithms for NP-hard problems (pp. 46-93). PWS Publishing Co.

[2] Martello, S., & Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. John Wiley & Sons.

[3] U. Eliiyi and D. T. Deniz, "Applications of bin packing models through the supply chain", Int. J. Bus. Manag. Stud., vol. 1, no. 1, pp. 11-19, Jan. 2009.

[4] A. Laurent and N. Klement, "Bin packing problem with priorities and incompatibilities using PSO: Application in a health care community", Proc. 9th IFAC Conf. Manuf. Modelling Manage. Control, pp. 2744-2749, Aug. 2019.

[5] Falkenauer, E. (1998). Genetic algorithms and grouping problems. John Wiley & Sons.

[6] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

[7] Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman and Company.

[8] K. Sörensen and F. Glover, "Metaheuristics", Encyclopedia Oper. Res. Manage. Sci., vol. 62, pp. 960-970, 2013.

[9] S. Martello, Knapsack Problems: Algorithms and Computer Implementations, New York, NY, USA:Wiley, 1990.

[10] D. S. Johnson, Near-Optimal Bin Packing Algorithms, Cambridge, MA, USA:Massachusetts Institute of Technology, 1973.

[11] E. Aarts and Jan Karel Lenstra, Local Search in Combinatorial Optimization. Princeton University Press, 1997.

[12] Glover, Fred. "Tabu search—part I." ORSA Journal on computing 1.3 (1989): 190-206.

[13] Holland, J. H. (1975). Adaptation in natural and artificial systems. University of Michigan Press.

[14] György Dósa, Rongheng Li, Xin Han, Zsolt Tuza, Tight absolute bound for First Fit Decreasing bin-packing: FFD(L)⩽11/9 OPT(L)+6/9, Theoretical Computer Science, Volume 510, 2013, Pages 13-61

[15] E. Wessa and A. Atia, "Parallelization of One Dimensional First Fit Decreasing Algorithm," 2021 16th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, Egypt, 2021, pp. 1-5

[16] R.L. Rao, S.S. Iyengar, Bin-packing by simulated annealing, Computers & Mathematics with Applications, Volume 27, Issue 5, 1994, Pages 71-82

[17] Hansen, Per Brinch, "Simulated Annealing" (1992). Electrical Engineering and Computer Science - Technical Reports. 170.

[18] Cort´es-Guzm´an, M. and Hinojosa-Cavada, C. (2019) A random search-based selection hyper-heuristic for solving the online class constrained one-dimensional bin packing problem

[19] Daniel Delahaye, Supatcha Chaimatanan, Marcel Mongeau. Simulated annealing: From basics to applications. Gendreau, Michel; Potvin, Jean-Yves. Handbook of Metaheuristics, 272, Springer (2019)

[20] C. Munien, S. Mahabeer, E. Dzitiro, S. Singh, S. Zungu and A. E. -S. Ezugwu, "Metaheuristic Approaches for One-Dimensional Bin Packing Problem: A Comparative Performance Study," in IEEE Access, vol. 8, pp. 227438-227465, 2020, doi: 10.1109/ACCESS.2020.3046185 (2020)

[21] Munien, C. & Ezugwu, A. (2021). Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. Journal of Intelligent Systems, 30(1), 636-663.

[22] Glover, Fred & Laguna, Manuel. (1999). Tabu search I. 10.1287/ijoc.1.3.190.

[23] Harun Pirim, Engin Bayraktar and Burak Eksioglu (2008) Tabu Search: A Comparative Study .