IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 7, 2020, accepted December 16, 2020, date of publication December 21, 2020, date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3046185

Metaheuristic Approaches for One-Dimensional Bin Packing Problem: A Comparative Performance Study

CHANALEÄ MUNIEN[1], SHIV MAHABEER[1], ESTHER DZITIRO[1], SHARAD SINGH[1], SILULEKO ZUNGU[1], AND ABSALOM EL-SHAMIR EZUGWU[2], (Member, IEEE)
[1]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban 4001, South Africa
[2]School of Computer Science, University of KwaZulu-Natal–Pietermaritzburg, Pietermaritzburg 3201, South Africa

Corresponding author: Absalom El-Shamir Ezugwu (ezugwua@ukzn.ac.za)

ABSTRACT Nature-inspired metaheuristic algorithms have steadily gained popularity over the last two decades. They have been applied to a plethora of optimization problems both in continuous and combinatorial domains. In this paper, the one-dimensional bin packing problem is solved through the implementation of two underlying heuristics, namely, best fit and better fit, and four representative state-of-the-art global metaheuristic algorithms, namely, firefly algorithm, genetic algorithm, adaptive cuckoo search algorithm, and artificial bee colony algorithm. The underlying best fit and better fit heuristics are employed by the four aforementioned global metaheuristic algorithms as reordering heuristics and local search improvement mechanisms are used for generating good packing schema. Furthermore, these two local heuristics possess special characteristics which, when incorporated into the global metaheuristics, allow them to escape local optimums and avoid getting stuck, and more so, enables the algorithms to generate good quality solutions. The main focus of this paper is the presentation of a systematic performance evaluation study for the representative algorithms, with some initial computational results that show the effectiveness of the respective algorithms and their ability to achieve promising solutions. The experiments conducted here were carried out using three standard bin packing problem datasets categories with over 1,210 instances in total, each with differing capacities, number of items and distributions between item weights. The numerical results of the representative algorithms were compared with the solutions achieved with the underlying heuristic techniques, in this case, the best fit and better fit heuristics, respectively. Similarly, the analysis of the initial computational results obtained revealed the superior performance of the individual algorithm implementation. Moreover, performance was established by taking into account both the algorithms computational time and the solution quality. Overall, several observations regarding the solution and the underlying heuristics were made. It is worth noting that by utilizing best fit heuristic, the algorithms attained optimal solutions for instances of the easy dataset requiring smaller capacity bins. However, as the complexity of the instances increased, the ability to produce high-quality results decreased. Nevertheless, this heuristic can produce near-optimal solutions and a good packing schema for most instances. On the other hand, utilizing better fit as the underlying heuristic results in optimal solutions almost all the time, regardless of capacity and number of items.

INDEX TERMS Bin packing problem, best fit, better fit, firefly, genetic algorithm, adaptive cuckoo search, artificial bee colony.

I. INTRODUCTION

In the classical one-dimensional bin packing problem, we are given a positive, fixed bin capacity, C, and a list of n items $L = (p_1, p_2, \ldots, p_n)$, where p_i has a size, $s(p_i)$ that must satisfy the constraint, $0 \leq s(p_i) < C$. In other words, the size of an item must not exceed the fixed bin capacity [30]. The requirement is to determine the smallest integer, m, such that there is a partition of $L = (B_1 \cup B_2 \cup \ldots \cup B_m)$ where the sum of the sizes of the items, $p_i \in B_j$, does not exceed the

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

capacity C. Each set, B_j is usually viewed as the contents of a bin capacity, C. The sum of the sizes of the items $p_i \in B_j$ must not exceed C. The set B_j can be seen as the contents of a bin. Clearly speaking, items in a list must be packed into containers, or bins, of a fixed size. The packing must be done such that the number of containers required is minimized.

Mathematically, the bin packing problem (BPP) can be modelled as follows [31]. Let u be an upper bound on the minimum number of bins that are required to pack all the items in a given list. It can be assumed that these bins are numbered as $1, 2, \ldots, u$. Let n be the number of items that are to be packed. Two binary decision variables must be introduced as represented in Equations (1) and (2) below:

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

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

Then,

$$\textit{Minimize} \sum_{i=1}^{u} \boldsymbol{y_i} \tag{3}$$

$$\text{s.t.} \sum_{i=1}^{u} w_j \boldsymbol{x_{ij}} \leq c y_i \quad 1 \leq i \leq u \tag{4}$$

$$\begin{aligned} &\sum_{i=1}^{u} \boldsymbol{x_{ij}} = 1 \quad 1 \leq j \leq n \\ &\boldsymbol{y_i} \in 0, 1 \quad 1 \leq j \leq n \\ &\boldsymbol{x_{ij}} \in 0, 1 \quad 1 \leq i \leq u; \ 1 \leq j \leq n \end{aligned} \tag{5}$$

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 can be mathematically expressed as follows:

$$N \geq [\frac{(\sum_{i=1}^{n} S_i)}{C}] \tag{6}$$

where S_i represents the size of each item in the list of n items, and C represents the fixed capacity of the bins.

Although our study focuses on the one-dimensional BPP, there is a myriad of variants of the BPP, including the classical one-dimensional bin packing problem (1D-BPP), the two-dimensional bin packing problem (2D-BPP), and the three-dimensional bin packing problem (3D-BPP). In the 1D-BPP, there is a set of items to be packed into a bin, $\boldsymbol{X} = 1, 2, \ldots, n$ where n is the last item in the list [41]. An item carries a specified size, and the bins have a fixed capacity. The goal is to find the minimum number of bins without exceeding the capacity of the bins. While (2D-BPP) works with a group of rectangles correctly identified by their height and width that must be packed into a minimum number of containers [42], the (3D-BPP) consists of a limitless number of similar three-dimensional bins possessing depth, height, and width. Moreover, researchers have studied specific versions of the BPP to solve specific problems. An example of this is Coffman, *et al.*, [58] who introduced the concept of maximizing the total number of items that are packed into each bin. This was intended to model processor and storage allocation problems. In this case, given a specific number of bins, m, and a list of items, $\boldsymbol{X}$, the aim is to pack the maximum subset of X into the bins such that the capacity of the bin is not exceeded. Another example of researchers investigating variants is Krause *et al.*, [59], who introduced the BPP that restricts the number of items that are allowed to be packed into a single bin. Therefore, for some positive number, k, each bin in the solution must contain at most k items. This variant attempts to solve a task scheduling problem.

The BPP has several real-world applications such as industrial applications [24], [25], packaging design in supply chain management [26] and even in health care communities [27]. Specifically, there are applications of the BPP in the supply chain industry, such as filling up containers and the loading of trucks with a given weight limit; this application requires detailed specifications to be considered when performing the BPP such as the various sizes of the items or special transportation needs. Real-world application examples from this instance are the multi-container loading problem [43], and the multi-pallet loading problem [44]. Other classic examples include the allocation of memory in computers and the assigning of commercials to station breaks. In a general sense, the BPP is a popular and widely researched combinatorial optimization NP-hard problem. This means that it is very unlikely that any deterministic polynomial-time algorithms that solve this problem do exist in practice [23]. Although there are a handful of approximate algorithms that are able to solve very small problem instances for the BPP, the feasibility of this approach decreases as the magnitude of the problem instances increases. Therefore, for real-world applications, a heuristic approach must be used to find satisfactory results [28].

In this study six population-based, nature-inspired metaheuristic algorithms were employed to solve the classical one-dimensional bin packing problem, namely, the firefly algorithm (FA) [5], hybrid firefly algorithm (FAH) [11], [65], adaptive cuckoo search algorithm (ACSA) [13], hybrid cuckoo search genetic algorithm (CSGA), artificial bee colony (ABC) [39], and the genetic algorithm (GA) [50], [64]. It is noteworthy to mention here that the main focus and contribution of this paper is to evaluate the capabilities of the selected meta-heuristics algorithms to solve the one-dimensional bin packing problem. More specifically, it is to investigate the effectiveness and efficiency of those well-known optimization algorithms namely, the GA, FA, CS, ABC, and some of their hybrids that have notable track records in finding good quality solutions for a variety of difficult and complex practical optimization problems. Furthermore, it is to also carry out comparisons on the various levels of difficulties for the standard BPP datasets and to determine how well each of the aforementioned algorithms performs on different dimensions of the 1D BPP.

It is equally important to note that the main benefits of utilizing metaheuristics to solve complex optimization problems include the algorithms' ability to easily handle complex constraints present in real-life applications and produce high-quality solutions while requiring shorter computational time [29], [72]–[75]. Each of the algorithms was adapted or modified and applied to the problem at hand. Furthermore, the best fit and better fit heuristics were tested with each algorithm to determine which underlying packing mechanism resulted in better results, measured by the computational expense (or time taken to find a solution), as well as the feasibility of the final solution. The technical contribution of this paper is summarized as follows:

- A review of the state-of-the-art metaheuristic optimization algorithms for the one-dimensional BPP
- A systematic performance study of representative metaheuristic algorithms for the one-dimensional BPP
- The implementation of two hybrid metaheuristics, namely, hybrid cuckoo search genetic algorithm and mutated firefly algorithm to solve the BPP
- The presentation and evaluation of some initial results for the one-dimensional BPP using metaheuristic algorithms approaches.

The rest of the paper is structured as follows: Section II presents a detailed literature review of the different heuristics methods employed to solve the BPP. Section III gives a detailed formulation and model implementation discussions on the respective representative algorithms employed in this paper to solve the BPP. Experimentations and comparative performance study of the representative algorithms are presented in Section IV. Finally, the concluding remarks and future research direction is given in Section V.

II. LITERATURE REVIEW

The drive from deterministic to stochastic algorithms has been stimulated by the quest to escape local optima and obtain global optima [1]. Some methods, such as Tabu Search, employ a semi-deterministic approach; however, other methods such as Simulated Annealing (SA) use a completely stochastic approach [1]. The one-dimensional BPP has seen a great number of algorithms applied to it, from both classical based heuristic techniques and global metaheuristic algorithms angles, and also from deterministic and stochastic angles. In this section, we present a review from the perspective of classical and global metaheuristic algorithms available in the literature that have been applied to solve the one-dimensional bin packing problems.

A. CLASSICAL HEURISTIC APPROACHES FOR 1D-BPP

The traditional set of BPP optimization algorithms are all on-line and sequential heuristics. Sequential, off-line extensions of some of these traditional heuristics have also been formulated. On-line algorithms for the BPP feed data to the heuristic without changing their initial ordering, whereas off-line algorithms for the BPP involve a reordering of items before being fed to the heuristic. This reordering most commonly makes use of sorting of the items [2]. These reordering heuristics are, specifically, the First Fit, Best Fit and Next Fit heuristics. The First Fit works by adding an item into the first bin that it fits into. If an item does not fit into the current bin, a new bin is made available, and the item is inserted into the new bin. Bins are cycled through sequentially in a fixed order. Next Fit functions slightly differently; items are placed into the same bin until the bin cannot take the next item. Once this occurs, the bin is closed and will no longer be used. Best Fit works on the premise of looking at all bins and identifying the bin that the current item can be added to, to result in the least wastage [3], [4]. The First Fit Decreasing and Best Fit Decreasing heuristics are created by reordering the list of items, by weight, into decreasing order before being fed to the respective sequential heuristic. These have been shown to perform better than their respective predecessors. Listing the traditional and extension heuristics, in descending order of performance, one obtains Best Fit Decreasing, First Fit Decreasing, Best Fit, First Fit and Next Fit [3]. The better fit algorithm was proposed by [38] for the bin packing problem. The better fit algorithm replaces the object that is already packed in the bin with the next object on the list if the next object fills the bin better. The time complexity of this algorithm is $O(n^2m)$ where n is the number of objects and m is the number of distinct sizes in the list. This algorithm was proved to produce better results than the best fit algorithm. This is in stark contrast to the best fit; the time complexity of this algorithm is O $(n \log n)$. The best fit produces the worst packing of 1.7* optimum [38].

B. METAHEURISTIC APPROACHES FOR 1D-BPP

In recent years, metaheuristic approaches to solve the one-dimensional BPP have become popular. This is largely due to these algorithms' ability to easily handle complex constraints that are present in real-life applications of optimization problems, and the fact that metaheuristics require less computational time to produce high-quality solutions [29]. The Whale Optimization Algorithm was adopted in the work of [34] to solve the one-dimensional bin packing problem. Whale Optimization Algorithm is a swarm intelligent metaheuristic that imitates the peculiar hunting strategy of humpback whales, known as the 'bubble net strategy' [35]. This adaptation incorporated Lévy flights, an additional mutation phase, and a logistic chaotic map to enhance the exploration capabilities of the original algorithm. The results of this algorithm were very similar to that of the Adaptive Cuckoo Search [14]. However, more reasonable solutions were found with fewer iterations and search agents.

Kucukyilmaz and Kiziloz [61] proposed a novel scalable island-parallel grouping genetic algorithm for the well-known combinatorial optimization problem, which is the one-dimensional bin-packing. The authors provided a thorough experimental evaluation of the parallel model and reported significant improvements on the Hard28 problem instances by outperforming the state-of-the-art existing genetic algorithms. Furthermore, they also analyzed and evaluated the parallelization parameters of the proposed algorithm with an emphasis on problem search-space diversity and reported several interesting results. In another related work, Dokeroglu and Cosar [62] proposed a set of robust and scalable hybrid parallel algorithms that are capable of exploiting and leveraging the advantages of parallel computation techniques, evolutionary grouping genetic metaheuristics, and bin-oriented heuristics to obtain solutions for large scale one-dimensional BPP instances. In their study, a total number of 1,318 benchmark problems were examined with the proposed set of algorithms, and it was shown that optimal solutions for 88.5% of these instances could be obtained with practical optimization times while solving the rest of the problems with no more than one extra bin.

Abd Elminaam *et al.*, [32] proposed an adaptive procedure using a recently optimized swarm algorithm and fitness-dependent optimizer (FDO), named the AFDO, to solve the one-dimensional BPP. Their algorithm was based on the generation of a feasible initial population through a modified well-known first fit heuristic approach, in which the authors adapted the most critical parameters of the algorithm for the problem to obtain a final optimized solution. Their study was the first to apply the fitness-dependent optimizer algorithm in a discrete optimization problem, especially for solving the BPP. In their implementation test scenario, the adaptive algorithm was tested on 30 BPP benchmark instances. The obtained results of the algorithm were compared with those of other popular algorithms, such as the PSO algorithm, crow search algorithm, and Jaya algorithm. The analysis of results by the authors showed that the AFDO algorithm obtained the smallest fitness values and outperformed the PSO, CS, and Jaya algorithms by 16%, 17%, and 11%, respectively. Similarly, in terms of execution time, the AFDO algorithm showed superiority with improvements over the execution times of the PSO, CS, and Jaya algorithms by up to 46%, 54%, and 43%, respectively.

Gherboudj [63] presented an adaptive African Buffalo Optimization (ABO) for solving the NP-hard one-dimensional BPP. The ABO algorithm was used in combination with the ranked order value method to obtain discrete values and BPP heuristics to incorporate the problem knowledge. The performance of the ABO algorithm was tested on 1,210 of one-dimensional BPP problem instances. The obtained results were compared with those found by recently developed algorithms in the literature, and the computational results revealed the effectiveness of the ABO algorithm and its ability to achieve best and promising solutions compared to other existing metaheuristic algorithms.

Ashamawi *et al.*, [36] modified the Squirrel Search Algorithm and used it to solve the classical one-dimensional bin-packing problem. This algorithm mimics the behaviour of flying squirrels and their use of gliding [37]. The adaptation includes the generation of random but feasible initial solutions and various operating strategies to update the solutions. The overall results of the proposed algorithm performed very well in comparison to other algorithms, but specifically Particle Swarm Optimization. Even though optimal solutions were not always reached, the final solutions were much closer to the optimal number.

To solve discrete optimization problems, Abdul-Minaam *et al.*, [32] adapted the Fitness Dependent Optimizer algorithm and tested this adaptation on the one-dimensional bin packing problem. This is also a swarm intelligent algorithm that draws inspiration from the characteristics of the reproductive process of bee swarms and their collective decision-making behaviour [33]. A random initial population and an update on the critical parameters of the algorithm were added to the adaptive algorithm. The results of the proposed algorithm significantly outperformed state-of-the-art metaheuristics, including Particle Swarm Optimization and Crow Search Algorithm, with respect to average fitness values. Additionally, satisfactory results were obtained within a reasonable time. Where the algorithm failed to reach optimal or near-optimal solutions, it achieved better item packing schemas with a lower number of bins compared to the results of the other algorithms it was tested against.

Simulated annealing (SA) is another metaheuristic algorithm mainly utilized for global optimization in an ample search space. Pinto *et al.*, [45] proposed the similarity in matter physics and annealing in which they explained annealing as a physical procedure in which a solid is heated to the maximum temperature in a hot bath; at this high temperature all material is in the liquid state, and the particles randomly arrange themselves. Furthermore, as the temperature of the hot bath is cooled gradually, all the particles of this structure will be arranged in the state of lower energy. The authors also elaborated on the four existing aspects of the algorithm they looked at: 1) the initial solution which is generated using a heuristic and chosen at random; 2) the neighbourhood which is also generated at random and mutates the currently existing solution; 3) the acceptance criteria, where the solution is accepted when a neighbour has a lower cost value or a higher cost value given a probability p is accepted; and 4) stopping criteria include when a maximum CPU time is reached, maximum iterations have been reached and getting a solution with a lower value than the threshold. The authors further elaborated on SA being a heat treatment that aims at expanding the flexibility of a specific metal while lessening the hardness to make it easier to work with.

Bertsimas *et al.*, [46] focused on how the SA algorithm works by copying the process in which a solid is steadily cooled down, so its structures eventually freeze; furthermore, the authors looked at how fast the SA converges to the optimal solution. No definite solution was found as many factors were in place in their experiment. The simulated annealing algorithm has a crucial characteristic that accepts worse solutions. This characteristic allows it to escape local optimums and avoid getting stuck. Reference [45] also elaborated on the limitations of the algorithm. Amongst discussed limitations was the long computational time and when and how to decrease the temperature of SA. The SA has an acceptance function that determines when to take a solution to prevent local optimums. The algorithm checks if the neighbour solution is better than our current solution. If the neighbour solution is not better, we consider the following factors: 1) How much worse is the neighbour solution? 2) How high is the current temperature of the system? Furthermore, there exist some convergence outcomes, that outlines "under certain mild conditions an optimal solution is found with a probability of 1" [47].

There exist very few implementations of SA for the bin packing problem. Our work focuses on the development of the SA for the BPP, and to provide further studies on the work already done in this area. Rao *et al.*, [48] looked closely at such an implementation of the BPP. The authors compared the performance of five heuristic algorithms, including the SA. The studied algorithms were 1) The Largest piece first (LPF) that aims to pack the largest piece first in a list of items arranged in decreasing order; 2) The Shortest piece first (SPF) which is similar to LPF except that items are arranged in increasing order; 3) First fit decreasing (FFD) in which list items are sorted in non-increasing order; and 4) the First fit increasing (FFI) which is similar to the FFD except that items are sorted in increasing order. In the experiment, the FFD and SA were the better performing algorithms. However, it was noted that the SA was the only algorithm that performed consistently with an increase in the number of bins. This tells us that the quality of the solution is constantly concise and of quality. At the same time, Sonuc *et al.*, [49] compared the FFD alone and the FFD plus SA(FFD+SA) performances. The FFD+SA again performed better than the FFD according to the experiment results on which only FFD+SA obtained optimal results. This gives us the slightest clue that the algorithm can produce a quality result for the BPP.

III. REPRESENTATIVE METAHEURISTIC ALGORITHMS

A. FIREFLY ALGORITHM

This algorithm has become something of a stalwart in the field of optimization with a plethora of applications, and in order to be made fit for these arietinous uses, it must be altered in some way [5]. In this paper, the Firefly Algorithm is modified to optimize combinatorial optimization problems, specifically, the one-dimensional BPP.

The Firefly Algorithm (FA) is relatively new to Swarm Intelligence Algorithms, having only been created in 2008 [6]. It is a nature-inspired, population-based, metaheuristic algorithm underpinned by stochastic searching mechanisms and it draws inspiration from the effect of the bioluminescent light of fireflies on the swarm they are in. Whilst it does not guarantee finding the optimal solution, its stochasticity aids the algorithm from being "stuck" in local optima [5]. As with any metaheuristic algorithm, two key factors must be accounted for: the payoff between exploration and exploitation [7].

Whilst the true reason for the flashing light of fireflies remains unknown, it is speculated that it serves a two-fold purpose: to attract potential mates and to lure prey toward the flies med. The basis of the FA considers only one of these functions: the flashing lights of fireflies is used to attract potential mates [8] and makes three simplifying assumptions about this process. Firstly, the fireflies are unisex, thus are capable of attracting any firefly [8], [9]. Secondly, how attractive a firefly is proportional to how bright it is and less bright fireflies move toward brighter fireflies [8], [9]. It should be noted that if a firefly is not surrounded by a brighter firefly, then its movement is random [9]. Furthermore, the brightness, and consequently, the attractiveness, decreases as the distance between two fireflies increases [10]. Thirdly, the brightness of a firefly is governed by the fitness function, which in turn, is derived from the objective function [5], [8]. These rules can be summarised as follows:

- Fireflies are unisex
- Brightness determines how attractive a firefly is
- Brightness is governed by the fitness function used in the specific use-case.

The implementation of the FA requires two fundamental considerations: how the light intensity of a firefly varies and designing the meaning of attractiveness [8], [9]. The manner in which these are dealt with is by assuming that attractiveness is contingent on the light intensity, which is contingent on brightness, which is dependent on the fitness function, derived from the objective function.

Light intensity is affected by the original light intensity (I_0), coefficient of absorption (γ) and the distance between the observer and the light source (r). Light intensity is subject to the inverse square law and consequently varies as follows $I(r) = I/r^2$ [8]. If $r = 0$ this will give rise to a mathematical impossibility, and this is accounted for by formulating the light intensity in Gaussian form [5]. The aforementioned considerations give rise to expression in Equation (7):

$$I\,(r) = I_0 e^{-\gamma r^2}. \tag{7}$$

The brightness of any particular firefly must be considered from the perspective of its fellow fireflies [8]. This implies that brightness is affected by the original brightness ($\beta_{\mathbf{0}}$), the coefficient of light absorption (γ) and the distance between the two fireflies (r). Therefore, brightness is formulated as represented in Equation (8):

$$\beta = \beta_0 e^{-\gamma r^2}. \tag{8}$$

The brightness and light intensity is not the same. Light intensity is a measure of the objective brightness of a firefly; however, brightness is a measure of how one firefly perceives the brightness of another firefly, that is to say, it is a relative measure [5], [8].

The distance, r, requires formalization. The distance can be measured in different ways. It is standard practice to define as Euclidean Distance [8], [9]. Consequently, r_{ij}, the Euclidean Distance between firefly s_i and firefly s_j, is defined as shown in Equation (9):

$$\boldsymbol{r_{ij}} = \|s_i - s\_j\| = \sqrt{\sum_{k=1}^{n} (s_{ik} - s_{jk})}, \tag{9}$$

Algorithm 1 Hybridized Muted Firefly Algorithm

```
begin
Objective Function f(x), $\mathbf{x} = (x_1, \ldots, x_n)^T$
Initialize population of fireflies $\mathbf{x_i} = (i = 1, 2, 3, \ldots, n)$
Initialize the parameters: $\gamma$, n, $\beta$ $\alpha$ and $\pi$
Determine light intensity $I_j$ at $\mathbf{x}_i$ by $f(\mathbf{x}_i)$
while (t < MaxGeneration)
for i = l : n all n fireflies
    for J = l : i all n fireflies
        if ($I_j > I_i$)
            Move the i^th firefly to the j^th firefly with distance r computed by the Euclidean Distance
            Mutate the i^th firefly with probability n
        end if
        Attractiveness varies with distance r via exp $[-\gamma r^2]$
        Evaluate the new solution and compute the new light intensity
        Decrease the mutation rate, $\pi$, in accordance with the current iteration number, t
    end for j
end for i
Rank the fireflies and identify the current best
end while
end
```

TABLE 1. Parameter setting.

β_0	0.3
γ	0.3
Population size	20

where n is the dimension of the solution vector. s_{ik} is k^{th} element of s_i and s_{jk} is the j^{th} element of s_j.

Finally, the way in which a firefly moves is determined by three factors: the current position of a firefly (s_i), the degree to which the i^{th} firefly is attracted to the j^{th} firefly, and lastly, a random walk weighted by α and randomized by ϵ, a random number between 0 and 1 inclusive, drawn from the Gaussian distribution [8]. This gives the following formalization of movement as represented in Equation (10):

$$s_i^{t+1} = s_i^t + \beta_0 e^{-\gamma r_{ij}^2} \left(s_i^t - s_j^t \right) + \alpha \epsilon_i^t. \tag{10}$$

Noting the above Equations 7 - 10, the FA has two special cases in behaviour, both asymptotic in nature [5]. The first is when $\gamma \to 0$. This results in a $\beta = \beta_0$ which is a special case of Particle Swarm Optimization. The second is when $\gamma \to \infty$. This results in the second term of the last Equation above becoming 0, thus making this case of FA a version of Simulated Annealing. The parameter settings for this study are as shown in Table 1 below.

B. HYBRID FIREFLY ALGORITHM

A hybridized version of the FA was also implemented. This involved randomly mutating a firefly with a decreasing rate of mutation. An additional stochastic mechanism of this sort serves to ensure avoidance of the algorithm being trapped within local optima. Furthermore, the mutation rate decreases as the number of iterations increases, ensuring that in the initial stages of running the algorithm favours exploration and in the later stages, it favours exploitation. Here, π represents the rate of mutation. The mutation itself involves reordering the arrangement of items passed into the underlying better fit heuristic, subsequently leading to an entirely different solution and firefly. The pseudocode for this hybridized mutated FA is shown in Algorithm listing 1.

The parameter settings are as shown in Table 2.

TABLE 2. Parameter settings.

β_0	0.3
γ	0.3
α	30
Population size	20

1) IMPLEMENTATION

Implementation of the two versions of the FAs was done in Java. The FA was initially designed for continuous optimization problems [11]. This research involves using the FA for the one-dimensional BPP, which is an NP-hard, combinatorial optimization problem. A cursory summary of the representation is that a firefly is nothing but an ordering of the items for the BPP. The order in which items are passed to the underlying heuristic effects significant change is seen in the more performant First Fit Decreasing over its predecessor First Fit [3]. The same is true of the best fit decreasing and best fit.

2) EXTENSIONS AND ADVANCEMENTS
The FA has seen an abundance of variations in a relatively short space of time. The two broad categories of these are Modified FAs and Hybrid FAs. Modification of the FA has focussed largely on enabling it to handle a greater ambit of optimization problems Hybridization of the FA places emphasis on managing the functioning and efficiency of the algorithm. Some notable modification ventures include advancing elitist and binary algorithms and the chaos FA, and notable ventures into the hybridization of the FA have been incorporating neural networks and genetic algorithms [5].

3) EFFICIENCY
The FA has proven itself as a highly performant algorithm, outperforming many other metaheuristic algorithms [5], [10]. This efficiency has been attributed to three primary characteristics of the FA.

Firstly, FA includes a notion of distance between the fireflies. As a result, a firefly is attracted to the brightest firefly in its local perimeter. Thus any particular firefly will not necessarily be attracted to the globally brightest firefly. This structure has the effect of separating the swarm into smaller sub-swarms. Consequently, FA copes well with optimization problems with a high non-linearity and a multi-modal nature [8], [10].

Secondly, it ameliorates two key issues of PSO. There is no issue of dealing with either a historical or global best, and there are no issues from the use of velocities. Thus, FA gains by not being slowed by the disadvantages associated with these issues [8], [10].

Lastly, the FA is extremely flexible, facilitating a high degree of control over its modality and being capable of suiting a variety of optimization problems through its parameters [8], [10]. If one were to alter the parameters, the FA could be augmented to behave like PSO, DE or SA [8]. This, in some sense, makes the FA a generalization of these three algorithms.

4) ADVANTAGES
- Simple to implement [5].
- It is efficient [5].
- As with any population-based algorithm, it avoids the issue of searching from a poor starting solution [5].

5) DISADVANTAGES
- It is known to have a slow rate convergence to the global optimum [12].
- Whilst it does have mechanisms to avoid being trapped in local optima [5], a chance still exists that it will happen [12].

C. ADAPTIVE CUCKOO SEARCH ALGORITHM
Yang and Deb in 2009, developed the Cuckoo Search (CS) via Lèvy flights metaheuristic algorithm [13]. This algorithm was inspired by the breeding strategy of certain cuckoo species, as they tend to lay their offspring in nests belonging to different types of birds, or host birds. Often, these cuckoos choose a nest that contains recently laid eggs. A host bird could react to the realization of a foreign egg in their nest in two ways -- they may get rid of the alien egg by discarding it, or they may abandon the entire nest and rebuild elsewhere. A number of cuckoo birds have evolved such that the female parasite cuckoos are able to mimic the appearance of the eggs of specific host birds, thereby reducing the odds of their own eggs being discovered and abandoned, which leads to an increase in reproductivity [15]. These cuckoo eggs tend to hatch before the other eggs present in the nest. The first instinct of the chick is to evict the host bird's eggs by propelling these eggs out of the nest, blindly. This leads to an increased portion of food provided to the cuckoo chick. The Lèvy flights mechanism replaces the simple random walk in order to enhance the performance of CS and more effectively explore the search space [13].

1) LÈVY FLIGHTS MECHANISM
Animals tend to search for food in a random or quasirandom manner. During their foraging path, the next move is based on the current state or location, and the transition probability to the next state, or location, and so can be considered a random walk. Research has highlighted that the flight-behaviour of various animals and insects (such as the fruit fly) demonstrate the characteristics of Lévy flights [13]. Named after the French mathematician Paul Lévy, the Lévy flight is described as a ''random walk in which the step-lengths are calculated with a heavy-tailed probability distribution'' [15]. After a significant number of steps, the distance from the origin of the random walk leans toward a stable distribution [17].

2) RULES OF CS
The three main principles of this algorithm are:
- Every cuckoo bird lays only one egg at a time and randomly chooses a nest to dump this egg in.
- The nests that contain eggs of the highest quality will proceed to the next generation.
- There is a fixed number of available hosts, and the cuckoo egg is discovered by the host bird with a probability, $p_a, \in [0, 1]$.

The possibility that a host bird discovers a foreign egg and reacts in the abovementioned ways can be determined by a fraction, p_a,of the fixed number of nests, n, which are replaced by new nests by using the local walk operator. A few new solutions may be generated by Lèvy walks around the best solution attained so far in order to accelerate the local search. However, to avoid getting trapped in a local optimum, the authors [13] suggested that a large portion of the new solutions be generated by far field randomization so that these new solutions are relatively far from the best-known solution.

3) ADVANTAGES
- Much like Genetic Algorithms and Particle Swarm Optimization, Cuckoo Search is a population-based

algorithm. Also, similar to Harmony Search, it uses a sort of elitism/selection [22].

- Randomization is more efficient due to the large steps [22].
- There are fewer parameters to be tuned, compared to that of Genetic Algorithms and Particle Swarm Optimization algorithms [22].
- Due to the ability of a nest holding multiple solutions (a set), CS had the ability to extend to a sort of meta-population algorithm [22].

4) IMPROVEMENTS ON THE CUCKOO SEARCH ALGORITHM

Over the years, various adjustments have been made to improve the CS algorithm. In 2018, Mareli *et al.*, [16] reviewed this algorithm and showed that in comparison to Simulated Annealing, Differential Evolution, and Particle Swarm Optimization, the CS algorithm was more general and efficient. In addition, these authors found that the improvements in efficiency were due to employing various probability distribution functions such as Gamma, Gauss and Cauchy, to control the step sizes of the random walk. Previously, Valian *et al.*, [17] set out to enhance the convergence rate and accuracy of the CS by proposing a strategy for tuning the search parameters, in place of keeping these parameters constant. The authors evaluated their presented strategy by applying the improved algorithm for training feedforward neural networks for two benchmark classification problems, an iris dataset and a breast cancer dataset, and demonstrated the significant enhancement of the efficiency of the algorithm.

Due to the CS being designed with continuous optimization problems in mind, Zakaria *et al.*, [14] adapted the algorithm to solve the bin packing problem by utilizing permutations of positive integers based on Lévy flights and a mechanism to decode continuous solutions. The ranked order value (ROV) rule was proposed to discretize solutions. The foundations from the original algorithm remained the same in this adaptive version. For this study, the algorithm was used as follows: As proposed by Yang *et al.*, [13], the parameter p_a is used to govern the balance between the global and local explorative random walks.

The local random walk can be defined as:

$$x_i^{t+1} = x_i^t + \alpha s \otimes H\left(p_a - \varepsilon\right) \otimes \left(x_j^t - x_k^t\right) \tag{11}$$

where x_i^{t+1} represents a new solution, x_i^t is an existing solution; s represents the step size;

$H\left(p_a - \varepsilon\right)$ is a Heaviside function where ε represents a random number, while x_j^t and x_k^t are two solutions selected randomly from the current population (of nests). The global walk (by Lévy flight) can be defined as expressed in Equation (12):

$$x_i^{t+1} = x_i^t + \alpha L(\varphi) \tag{12}$$

where $L(\varphi)$ represents the Lévy walk.

5) MAIN DRAWBACKS

- The convergence rate is likely to be affected by Lévy flights, and this results in a slow algorithm [18].
- May fall into local optimum solution if the parameters are not set correctly [19].

6) IMPLEMENTATION

The implementation of the adaptive cuckoo search algorithm was done by Zakaria *et al.*, [14]. According to the authors, the ranked order value produced results that were closest to the continuous solutions. This method is simple and "guarantees feasibility of new solutions without creating additional overhead" [14]. In the ROV method, the smallest value of a solution is found and assigned the lowest rank value, thereafter, the next smallest value is found and assigned the second smallest value. Continuing in this manner, the item permutation is produced.

We assume that a cuckoo bird deposits one egg in a nest and that every nest holds a single egg. The egg contains a continuous solution, and a corresponding integer permutation produced using the ROV technique. The best fit and better fit heuristics were used to place the items into the bins, corresponding to each solution, and to calculate the total wastage across all the bins. This value was used as the fitness value to determine the best nest.

7) THE OBJECTIVE FUNCTION

The aim of this algorithm was to minimize the wastage produced by a packing schema. The nests were constantly sorted according to ascending order of fitness value so that it was easy to keep track of the best nest found in the generations seen thus far.

The *wastage* of a bin is described as shown in Equation (13) below:

$$wastage = capacity - pointer \tag{13}$$

where the pointer represents the index that the bin's contents stop at. For instance, if the bin had a capacity of 10 and the items of value 1, 2, and 3 were added to the bin, the pointer would be at 6 $(1 + 2 + 3)$. Therefore, the wastage of this bin would be 4. This value shows how much of the bin's space is unused, and a higher amount of unused space indicates a less fit bin. It was observed that the lower the wastage of a bin was, the more filled it would be -- that led to the use of fewer bins.

Hence, the objective function is given as follows:

$$Minimize\, f : f = n \times capacity - \sum_{k=0}^{n} pointer_k \tag{14}$$

where, n is the number of bins required for a solution; *capacity* represents the maximum capacity for each bin (this is fixed), $pointer_k$ is the pointer of each bin, k. The parameter settings used are shown in Table 3.

TABLE 3. Parameter settings.

Number of nests	200
P_a	0.25
Number of iterations	200

D. HYBRID CUCKOO SEARCH GENETIC ALGORITHM

The hybrid algorithm implemented in this study was inspired by Lim, et al's. (2014) study on a hybrid cuckoo search-genetic algorithm (CSGA) to solve the hole-making sequence optimization problem [20]. The authors of this study were inspired by the fact that in some cases, if there is a paucity of suitable nests for a cuckoo bird to lay their egg in, parasitized nests can be utilized again by a second cuckoo bird. Since one egg is laid per nest, the cuckoo bird seeking for a nest will simply remove the existing egg and replace it with theirs – this is because the first egg will most likely hatch first [21]. This, along with the explanation of the CS algorithm in the previous section, helped the authors develop this hybrid.

In the CSGA algorithm, the Genetic Algorithm (GA) operators, mutation and crossover, replace the CS operators, random walks and Lévy flights, respectively. The following rules were taken into consideration:

- A chromosomal crossover occurs between the cuckoos that mate with one another.
- Cuckoo birds compete and lay eggs in host birds' nests. The best eggs survive.
- Cuckoo eggs evolve by mutation so that the probability of a host bird discovering a foreign egg is reduced.
- Eggs of lower quality are rejected by the host birds, and normal reactions apply (either evicting the egg from their current nest or abandoning their nest entirely and building a new one).

1) IMPLEMENTATION

The implementation of this algorithm followed that of Zakaria *et al.*, [14], however, the local and global walk operators were adapted accordingly. This ensured that all the assumptions made in the CS still held (all 3 rules mentioned). This section extends the CS and GA explanations.

The crossover operator is implemented as follows:

- Two solutions are randomly chosen from the current generation by means of a k-tournament (GA selection operation).
- Two random numbers between the upper and lower bounds are chosen (two-point crossover – eliminates the possibility of parents being completely switched around).
- The offspring are generated by interchanging the values of the parents within the selected points.
- A form of elitism is then employed. This means that a child will only progress if the evaluated fitness is greater than at least one parent.

The mutation operator, on the other hand, provides diversity to the population. It is implemented as follows:

- A mutation rate is set at the beginning of the algorithm.
- A number drawn from Gaussian distribution is checked against the mutation rate. If the number is less than the mutation rate, the mutation occurs.
- Two random numbers indicating the lower and upper bounds of the mutation are drawn.
- Then, the solution's items between the lower and upper bounds are reversed.

Algorithm listing 2 presents the pseudocode for the implemented hybrid cuckoo search genetic algorithm.

The parameter settings used are shown in Table 4.

TABLE 4. Parameter settings.

Number of nests	200
Mutation rate	0.25
Number of iterations	200
K (K-tournament)	10

E. ARTIFICIAL BEE COLONY

Artificial Bee Colony (ABC) algorithm was employed in this paper as a hybrid in order to solve the bin packing problem using the better fit algorithm and best fit algorithm as the underlying algorithms. This algorithm was proposed by Karaboga [40]. ABC is an optimization algorithm that mimics the food search behavior of honeybee swarms. The bee colony consists of 3 groups of bees: onlooker bees, employed bees, and scout bees. Each of the food sources consists of only one employee bee. The number of food sources in the hive is the same as the number as that of employed bees.

Employed bees go to their respective food sources and then return to the hive. When the employed bees return to the hive, they dance to invite onlooker bees to join; onlooker bees stare at the dancing of the employed bees and decide which to join based on the dances. The employed bee whose food source has been abandoned now becomes a scout bee and starts to search for a new food source.

1) ARTIFICIAL BEE COLONY ALGORITHM

In the ABC algorithm, the initial population is randomly generated. Each solution in the population consists of a corresponding objective function, fitness function, and there is a trial vector which is initially set to zero for all the solutions in the population.

A solution (food source) in the swarm is denoted by

$$x_i = x_{i,1}, x_{i,2} x_{i,3}, \ldots, x_{i,n}. \tag{15}$$

2) EMPLOYED BEE PHASE

In the employed bee phase a new solution v_i in the neighborhood of x_i is generated, and the Equation (16) is used to compute the new solution, which is expressed as follows:

$$v_{i,k} = x_{i,k} + \Phi_{i,k} \times (x_{i,k} - x_{j,k}) \tag{16}$$

Algorithm 2 Cuckoo Search Genetic Algorithm

```
Objective function $f(x), x = (x_1, \ldots, x_d)^T$
Generate the initial population of $m$ host nests $x_i (i = 1, \ldots, m)$
while (t < $MaxGeneration$) do
        Choose two solutions by k-toumament
        Solutions $(X_i, Z_i)$ crossover to reproduce new egg, $y_i$
        Convert $y_i$ to item permutation, $Y_i$
        $F(Y_i)$ = Evaluate fitness of $Y_i$
        if $(F(Y_i) > F(X_i)$ or $F(Y_i) > F(Z_i))$ then
                replace $x_i$ or $Z_i$ with the new solution $Y_i$;
        end if
        Cuckoo eggs mutate and are laid in other hosts' nests
        Low quality eggs are rejected by host birds
        Convert the new solutions to item permutations;
        Replace the new solutions by the item permutations;
        Evaluate their fitness;
        Rank the solutions and find the current best
end while
```

The objective function and the fitness function of the new solution are computed. Greedy selection is performed; if the fitness function of the new solution is better than that of the current solution, the current solution is replaced with the new solution. Otherwise, the trial of the current solution is increased. The fitness value may be calculated using the following Equation (17):

$$fit_i(x_i) = \begin{cases} \dfrac{1}{1+f_i(x_i)} & if\ f_i(x_i) \geq 0 \\ \dfrac{1}{1+abs(f_i(x_i))} & if\ f_i(x_i) < 0 \end{cases} \tag{17}$$

where $f_i(x_i)$ is the objective function. Before the onlooker bee phase, employed bees share their information with onlooker bees.

3) ONLOOKER BEE PHASE

In the onlooker bee phase, onlooker bees choose a food source (solution) to explore based on a probability. The probability of each food source in the population is calculated as follow:

$$p_i = \frac{fit_i(x_i)}{\sum_{i=1}^{SN} fit_i(x_i)} \tag{18}$$

After the food source x_i is chosen, a neighborhood food source is generated v_i and if the fitness function of the v_i is better than that of x_i, greedy selection is performed. The procedure is the same as that in the employed bee phase.

4) SCOUT BEE PHASE

In scout bee phase, all the employed bees that have solutions that cannot be improved enter the scout bee phase. This is determined by their trial; if the trial of the solution is greater than the selected threshold, the solution x_i is discarded and is replaced with a new solution v_i.

5) IMPROVEMENTS ON THE ARTIFICIAL BEE COLONY ALGORITHM

Variations of the artificial bee colony have been proposed, based on the modifications of the traditional algorithm by minimizing the disadvantages, e.g. early convergence. This will improve the performance of the algorithm. These new variations can be used in order to improve the current results.

6) IMPLEMENTATION

Generating the initial population for the artificial bee colony is achieved by randomizing the items in the list each time before each solution is generated. Each solution is generated by packing the randomized items using the better fit algorithm and the best fit algorithm. To generate the initial population, we randomized the items in the list, then packed them using better fit algorithm and the best fit algorithm. The objective function is determined by the number of bins and the wastage. The best solution is one which has the least number of bins and the least amount of wastage.

7) ADVANTAGES

- Its ability to generate a global optimum solution.
- It is simple to implement, strong robustness.
- Flexible.
- Consists of very few parameters [41].

8) DISADVANTAGES

- The algorithm has premature convergence, which can lead it into local optimum.
- The rate of convergence at a later stage is very slow.

The parameter settings used are shown in Table 5.

F. GENETIC ALGORITHM

Genetic algorithms are a heuristic search and optimization technique that is inspired by the evolutionary process

TABLE 5. Parameter settings.

Swarm size	40
Cycles	20
Limit	200
Population Size	20

found in nature. It was originally proposed by John Holland in 1960 and has evolved into a powerful algorithm that can solve difficult optimization problems [51]. It has more recently been used to create Hybrid Algorithms with other metaheuristics and optimization algorithms [52]. The genetic algorithm is a classic metaheuristic that has been covered many times over the last 50 years; as such, we decided to implement it as a comparator to the most popular and modern meta-heuristics.

The basic steps for a genetic algorithm are as follows [51]:

1. Initialise a population.
2. Evaluate the population.
3. Evaluate fitness of population.
4. Implement a selection method.
5. Crossover is performed.
6. Mutation is performed.
7. Determine the new population and new fitness.
8. Repeat steps 4 to 7 until termination criteria is met.

Each generation of a new population is created in the hope of finding a better population than the last. A solution is chosen based on its fitness, and the fitter candidates are used as the genes for the next generation, i.e. the 'fitter' a solution is, the higher its chance of being selected and the higher its chance of reproducing. This process is repeated until a stop condition is met. The Genetic Algorithm borrows terminology from the field of genetics and as such the components of a population in a GA are termed "chromosomes". These are n-dimensional arrays, of which each component is called a gene [53]. The components of a GA can be broken into the following: Chromosome Encoding, Fitness Function, Selection, Crossover/mutation, and solution space.

In the 1-D Bin Packing Problem, a chromosome is encoded with the Weight of the Item that position represents. The fitness function of a chromosome is evaluated to determine the quality of the chromosome as compared to other chromosomes [52]. The fitness function allocates a score (fitness) to each chromosome in the population to determine how well it can solve the problem. Based on this fitness function, a selection process decides which chromosomes to use as the basis of the next generation. The selection method will generally pick the fittest candidates, and these chromosomes will act as the parents for the next generation. This genetic algorithm uses k-tournament selection.

1) OPERATORS

Crossover then takes the chromosome pairs that the selection method has chosen and merges together their 'genetic material' to create a child for the new population. This is representative of reproduction in nature and ensures the passing on of good genetic materials into the new generation. This genetic algorithm uses a single-point crossover.

A mutation, as in biology, allows a child to exhibit characteristics that its parent never had, i.e. it did not inherit it from the parent population [52], [53]. The following is an example of a crossover operation.

$$\begin{aligned}
\text{Parent 1} &\rightarrow \mathbf{7\ 11\ 6\ 3\ 4\ 8}\\
\text{Parent 2} &\rightarrow \mathbf{11\ 6\ 8\ 7\ 4\ 3}\\
\text{Offspring 1} &\rightarrow \mathbf{7\ 11\ 6\ 8\ 4\ 3}\\
\text{Offspring 2} &\rightarrow \mathbf{11\ 6\ 8\ 7\ 3\ 4}
\end{aligned}$$

The following occurs after mutation:

$$\begin{aligned}
\text{Offspring 1} &\rightarrow \mathbf{7\ 4\ 6\ 8\ 11\ 3}\\
\text{Offspring 2} &\rightarrow \mathbf{11\ 7\ 8\ 6\ 3\ 4}
\end{aligned}$$

2) IMPLEMENTATION

The genetic algorithm is quite standard and uses the same conventions as most other GAs. It does, however, differ in the sense that it is used in conjunction with an underlying heuristic [38]. We represent our population as Items with weights e.g. {74, 32, 15, 23, 25, 36, 45} for which an initial population is made. The population is then randomized into X (user defined) amount of permutations of that set which makes up the initial generation. For example, in a case with a population of 5 the following could be the first generation:

$$\begin{gathered}
\{74, 15, 32, 23, 25, 36, 45\}\\
\{15, 74, 32, 23, 45, 25, 36\}\\
\{32, 23, 45, 15, 74, 36, 25\}\\
\{23, 32, 74, 15, 25, 36, 45\}\\
\{36, 32, 23, 15, 25, 74, 45\}
\end{gathered}$$

Fitness (the wastage in a bin) is determined for each of those permutations using the chosen packing method (best fit or better fit) after which standard crossover and mutations occur. This process is repeated for every generation until the stop criteria is met [51], [56], [58]. The stop criteria in our implementation occurs when the program iterates to the number of generations defined by the user.

3) ADVANTAGES

- Easy to conceptualize and understand.
- Quick to implement.
- Ideal for when a near optimal solution is acceptable, and an exact solution is unnecessary.
- Convergence speed is fast, and versatility is strong [57], [58].

4) DISADVANTAGES

- The increased computational time for complex problems.
- Slower than some other newer methods.
- Premature convergence can occur.
- Heavily dependent on the initial population [57], [58].

5) EXTENSIONS

The genetic algorithm can be combined with various other meta-heuristics to create hybrid metaheuristics. In this paper, we combined parts of the Genetic Algorithm to create the Cuckoo Search Genetic Algorithm Hybrid and the Firefly Hybrid. The parameter settings used are shown in Table 6.

TABLE 6. Parameter settings.

Mutation rate	0.2
Generations	50
Population size	50

IV. EXPERIMENTATION, RESULTS AND DISCUSSION

The following experiments were carried out using a 3.75 GHz AMD Ryzen 7 Processor and 16GB 2666Mhz memory. Algorithms were implemented in Java using the Eclipse integrated environment.

A. DATASETS

The datasets used for the 1-dimensional bin packing problem were obtained from [62]. There are 3 dataset categories with over 1,210 instances in total, each with differing capacities, number of items and distributions between item weights.

TABLE 7. Dataset 1.

Dataset 1 (Easy)			
Instance	Number of items	Capacity	Best known
N1C1W1_A	50	100	25
N1C1W1_D	50	100	28
N1C1W1_F	50	100	27
N2C1W2_P	100	100	68
N2C1W2_N	100	100	64
N2C1W2_O	100	100	64
N4C1W2_T	500	100	323
N4C1W4_A	500	100	368
N4C1W4_B	500	100	349
N4C1W4_D	500	100	359

Each of the following algorithms: Firefly Algorithm (FA), Firefly Hybrid algorithm (FAH), Adaptive Cuckoo Search Algorithm (ACSA), Cuckoo Search Genetic Algorithm Hybrid (CSGA), Artificial Bee Colony Algorithm (ABC), and the Genetic Algorithm (GA) were tested using best fit and better fit as underlying heuristics. In total, 30 instances were chosen, 10 instances from Dataset 1 (see Table 7), known as the easy dataset with 100 capacity; 10 instances from Dataset 2 (see Table 8), known as the medium dataset with 1000 capacity; and 10 instances from Dataset 3 (see Table 9), known as the hard dataset with 100000 capacity. Instances were chosen such that there was a variety of the number of items even though the capacity remained the same. Each dataset was run 10 times per metaheuristic and underlying heuristic to ensure accuracy of reported results.

A summary of the results is given in the next section, and key observations and results are discussed in detail. Furthermore, we specifically look at three instances from each dataset in more detail: N1C1W1_D, N2C1W2_N, N4C1W4_A, N1W1B1R9, N2W1B1R3, N4W2B1R3, HARD2, HARD5 and HARD8. These datasets were focused on specifically as they provide variation in terms of the number of items and capacity.

TABLE 8. Dataset 2 characteristics.

Dataset 2 (Medium)			
Instance	Number of items	Capacity	Best known
N1W1B2R1	50	1000	17
N1W1B1R9	50	1000	17
N1W1B2R0	50	1000	17
N2W1B1R3	100	1000	34
N2W3B3R7	100	1000	13
N2W4B1R0	100	1000	12
N4W3B3R7	500	1000	74
N4W4B1R0	500	1000	56
N4W2B1R3	500	1000	100
N4W4B1R1	500	1000	56

TABLE 9. Dataset 3.

Dataset 3 (Hard)			
Instance	Number of items	Capacity	Best known
HARD0	200	100000	56
HARD1	200	100000	57
HARD2	200	100000	56
HARD3	200	100000	55
HARD4	200	100000	57
HARD5	200	100000	56
HARD6	200	100000	57
HARD7	200	100000	55
HARD8	200	100000	57
HARD9	200	100000	56

B. RESULTS AND DISCUSSION

The results of the experiments conducted in this study and discussions on the outcome are presented in this section. The result of experiments conducted on Dataset 1 is presented in Table 10 where the bin capacity was set at 100, and the number of items varied at 50, 100 and 500, respectively. The graphical representation of results for N1C1W1_D is shown in Figure 1. Similarly, the results for N2C1W2_N and N4C1W4_A are as shown in Figure 2 and Figure 3, respectively. It can be observed from the Table and the Figures above that every algorithm performed well for each of the instances. Each algorithm was able to attain the optimal number of bins using the better fit algorithm, and the FAH and GA were only

TABLE 10. Results obtained from datasets 1.

Instance	Number of items	Capacity	Best known	ACSA		CSGA		FA		FAH		GA		ABC	
				*	**	*	**	*	**	*	**	*	**	*	**
N1C1W1_A	50	100	25	25	25	25	25	25	25	26	25	25	25	26	25
N1C1W1_D	50	100	28	28	28	28	28	28	28	28	28	28	28	28	28
N1C1W1_F	50	100	27	27	27	27	27	27	27	27	27	27	27	27	27
N2C1W2_P	100	100	68	68	68	68	68	68	68	68	68	68	68	68	68
N2C1W2_N	100	100	64	64	64	64	64	64	64	64	64	64	64	65	64
N2C1W2_O	100	100	64	64	64	64	64	64	64	64	64	64	64	64	64
N4C1W2_T	500	100	323	323	323	323	323	323	323	323	323	323	323	323	323
N4C1W4_A	500	100	368	368	368	368	368	368	368	369	368	369	368	368	368
N4C1W4_B	500	100	349	350	349	350	349	350	349	350	349	350	349	349	349
N4C1W4_D	500	100	359	359	359	359	359	359	359	359	359	359	359	359	359

Key: * → This algorithm was tested with the best fit heuristic
** → This algorithm was tested with the better fit heuristic

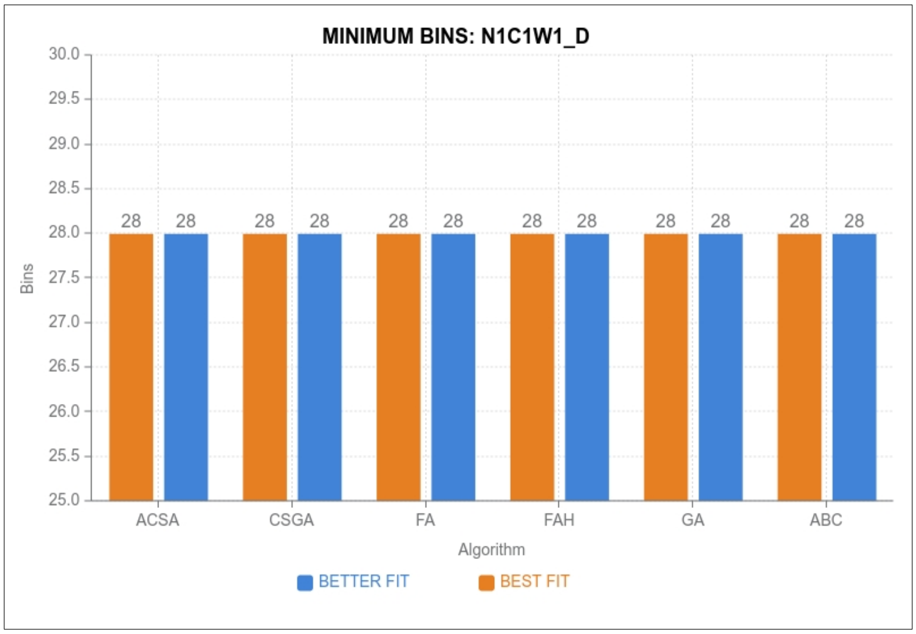


FIGURE 1. A graphical representation of the results of the N1C1W1_D instance.

one bin off when using the best fit algorithm. This is to be expected for Dataset 1 as the instances in this dataset are relatively basic.

The computational time comparison for the experiments on Dataset 1 is given in Figures 4-6. We noticed a trend occur across all three instances. Although the time taken to complete the algorithm changed, the general shape of the graph remained the same. Looking at the best fit first, we saw that the ABC performed the best across all three instances. In Figures 4 and 5 the difference between every algorithm was extremely small (less than a millisecond) and as such was negligible.

In Figure 6, for instance, with 500 items, the difference was more significant with the ABC and GA performing far better

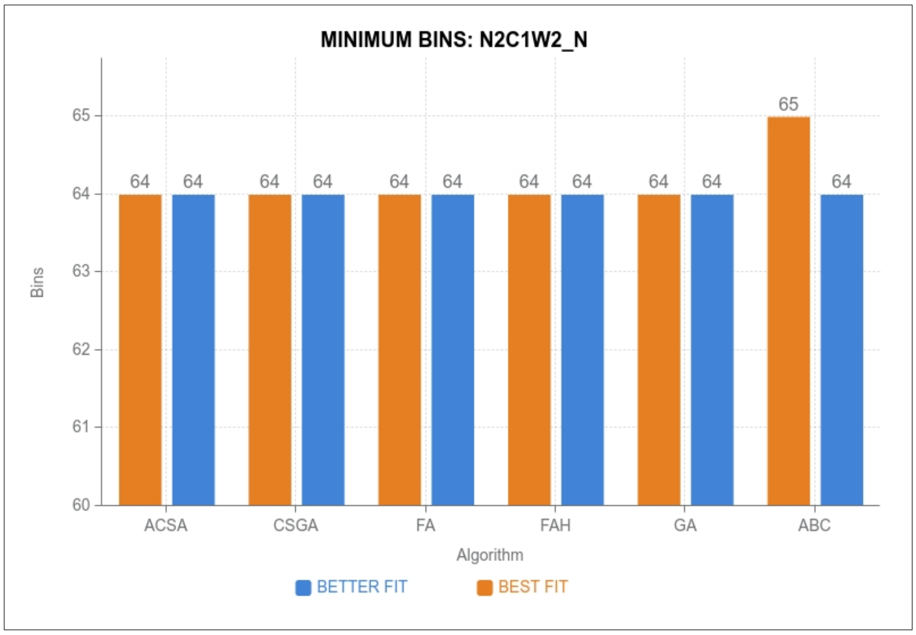


FIGURE 2. A graphical representation of the results of the N2C1W1_N instance.

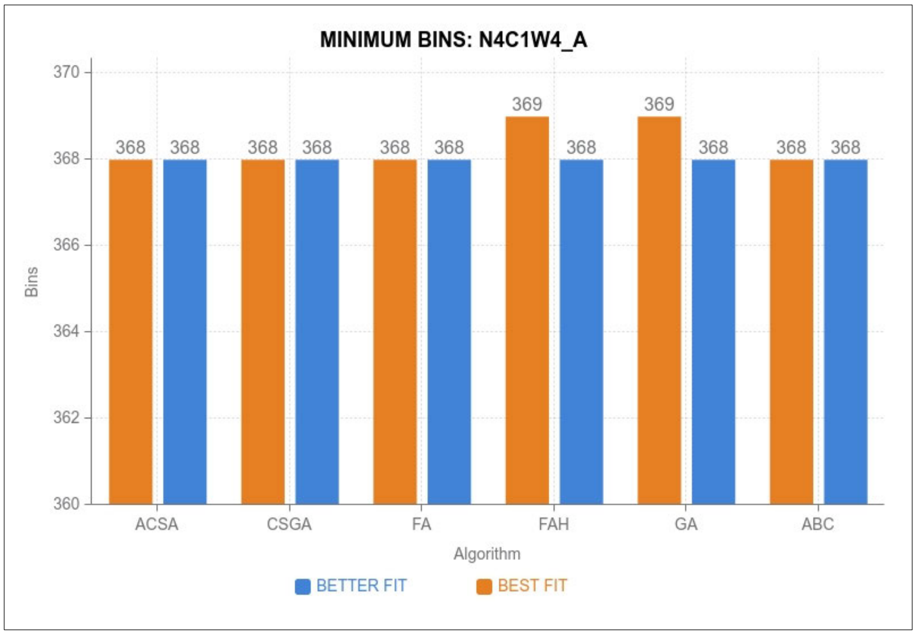


FIGURE 3. A graphical representation of the results of the N4C1W4_A instance.

than the other algorithms. The ACSA and CSGA were not far behind whilst the FA and FAH performed the worst. The difference, however, was still only a few milliseconds. In the Better fit heuristic, which is more computationally complex than the best fit, the ABC performed the best. The trends for the better fit were consistent throughout for all three instances with the ACSA, CSGA and GA performing worse than the other three algorithms. The differences in the third instance followed the same pattern, however, the actual time difference was staggering. The ACSA performed extremely badly when compared to the ABC, FA, and FAH. The genetic algorithms took up to 30 milliseconds longer on average whilst the ACSA took approximately 100 milliseconds longer. The results suggest that the GA, ACSA and CSGA have trouble with increasing item numbers, especially when using the better fit algorithm. The ABC, on the other hand, showed little to no difference in performance when the number of items increased.

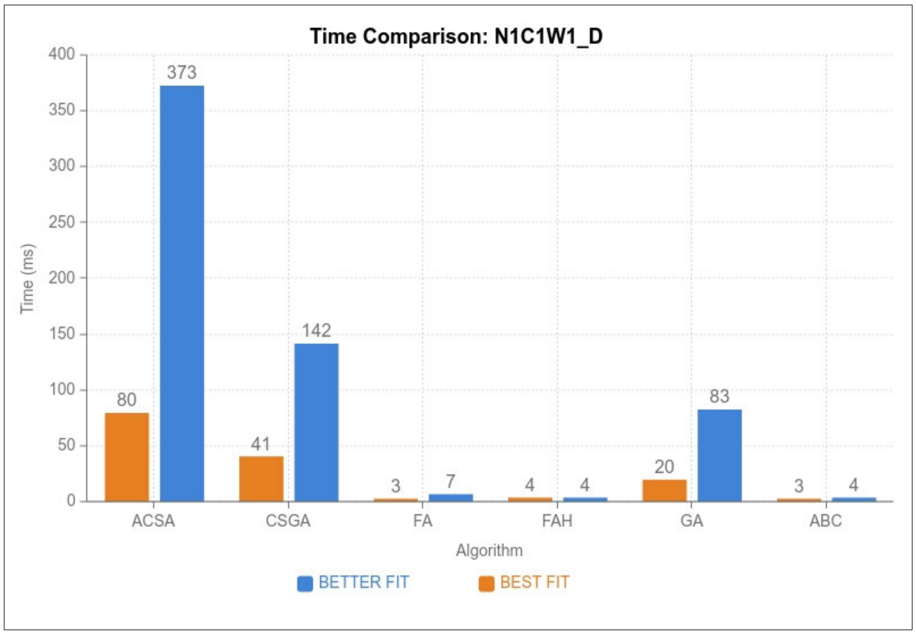


FIGURE 4. **Computational time results for the N1C1W1_D instance.**

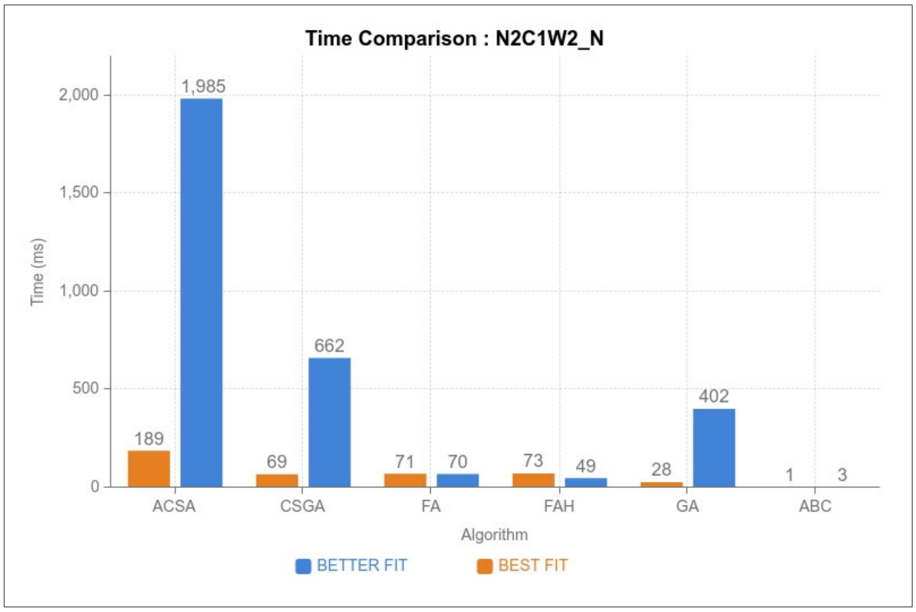


FIGURE 5. **Computational time results for the N2C1W2_N instance.**

Table 11 shows all results obtained from experiments on Dataset 2. These are the results for bins with a capacity of 1000 but with a varying number of items (50, 100 and 500 in Figure 8, Figure 9, and Figure 10, respectively.). As the number of items increased with the increased capacity, we began to see the effect of the better fit and best fit as underlying heuristics. Figure 7 only had 50 items, and as such, the best fit and better fit both performed well. The best fit was able to attain the optimal solution in all but two algorithms whilst the better fit attained it in all the algorithms.

In Figure 8 we begin to see what the increased complexity of increased items and capacity resulted in. Better fit attained the optimal solution for each metaheuristic whilst the best fit was unable to do the same. The ACSA and CSGA were able to perform slightly better than the other algorithms when using best fit in Figure 8. The metaheuristics performed identically for both the best fit and better fit methods in Figure 9 which showed an instance of 500 items and a capacity of 1000. Even the better fit heuristic was unable to obtain the optimal results for that specific instance. The results for this dataset were

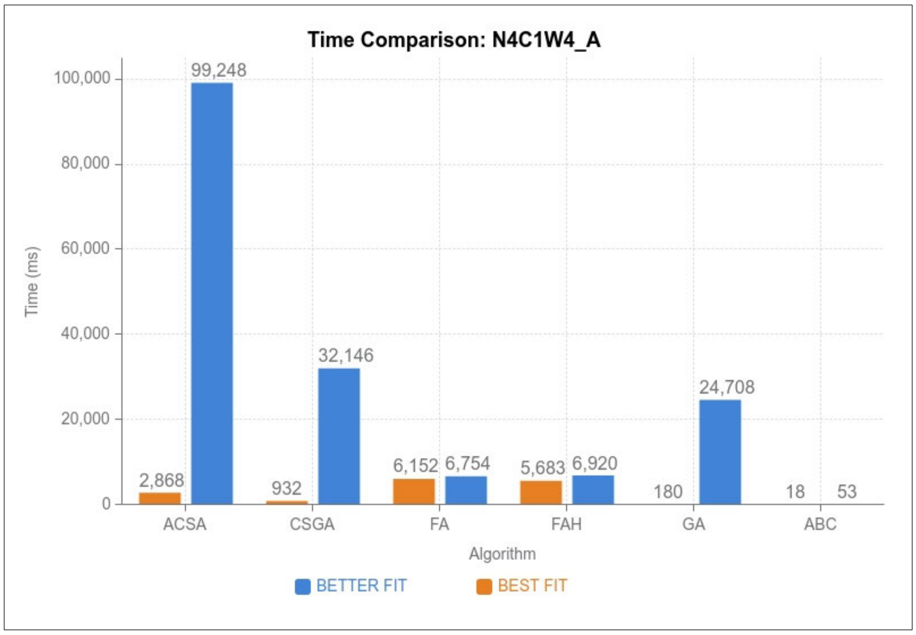


FIGURE 6. Computational time results for the N4C1W4_A instance.

TABLE 11. Results obtained from datasets 2 (Medium dataset).

Instance	Number of items	Capacity	Best known	ACSA		CSGA		FA		FAH		GA		ABC	
				*	**	*	**	*	**	*	**	*	**	*	**
N1W1B2R1	50	1000	17	17	17	17	17	18	17	17	17	17	17	18	17
N1W1B1R9	50	1000	17	17	17	17	17	18	17	18	17	17	17	17	17
N1W1B2R0	50	1000	17	17	17	17	17	18	17	18	17	17	17	18	17
N2W1B1R3	100	1000	34	35	34	35	34	36	34	36	34	37	34	36	34
N2W3B3R7	100	1000	13	13	13	13	13	13	13	13	13	13	13	13	13
N2W4B1R0	100	1000	12	12	12	12	12	12	12	12	12	12	12	12	12
N4W3B3R7	500	1000	74	74	74	74	74	74	74	74	74	74	74	74	74
N4W4B1R0	500	1000	56	57	56	57	56	57	56	57	56	57	56	57	56
N4W2B1R3	500	1000	100	105	101	105	101	105	101	105	101	105	101	105	101
N4W4B1R1	500	1000	56	57	56	57	56	58	56	58	56	57	56	58	56

Key: * → This algorithm was tested with the best fit heuristic
** → This algorithm was tested with the better fit heuristic

as expected. We saw an increase in disparity between the best fit and better fit as the complexity of the instance increased. The metaheuristics themselves all performed ideally and were able to find the optimal solution with a better fit in all but one instance (Figure 9). Discussing the time results will give us more clarity of the results.

The computational time comparison for the experiments conducted on Dataset 2 is given here. As with the previous time comparison, we noticed a trend occur across all three instances. The shape of the graph was the same, although the numbers changed. Figure 10's instance contained only 50 items, and as such, we saw that the time taken to pack

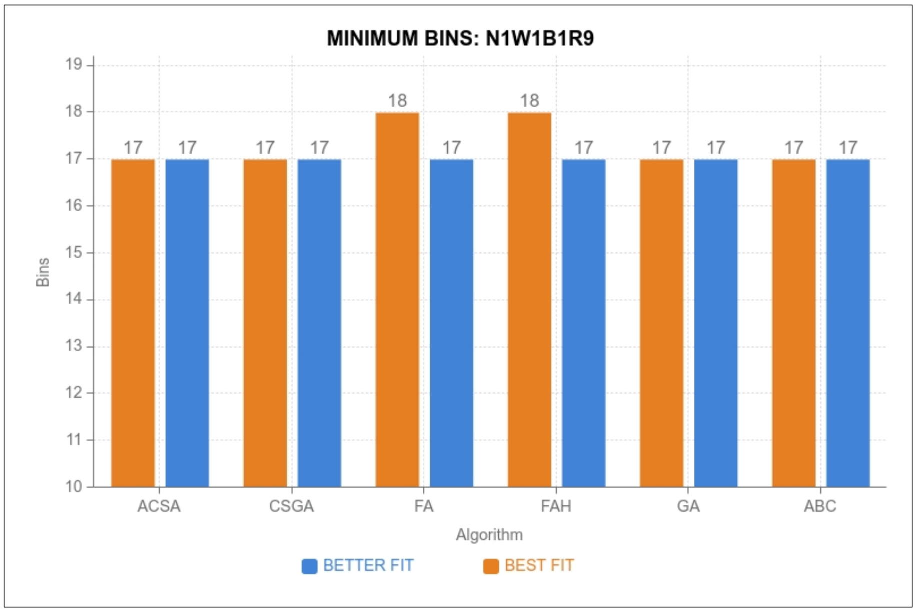


FIGURE 7. A graphical representation of the results of the N1W1B1R9 instance.

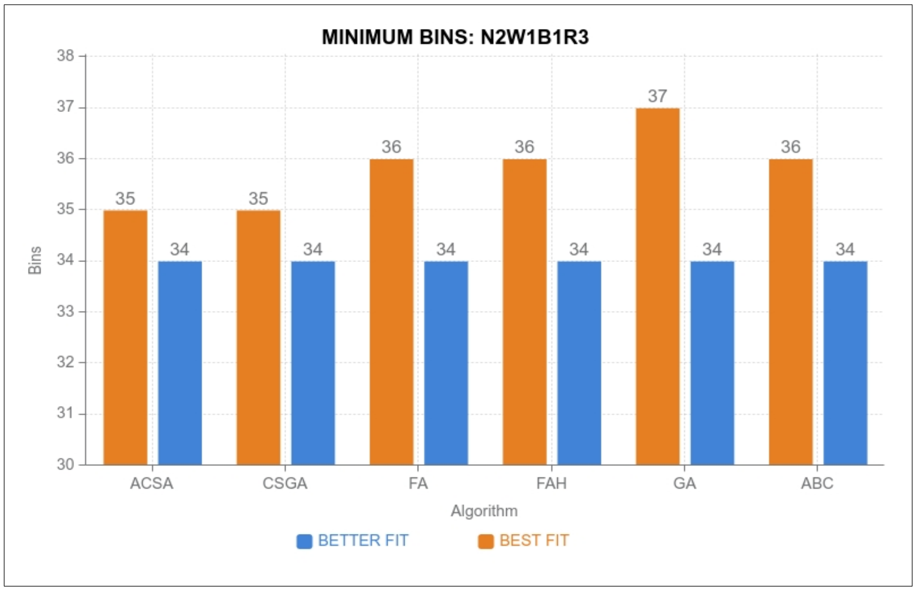


FIGURE 8. A graphical representation of the results of the N2W1B1R3 instance.

the bin was relatively short across all the algorithms. ABC was the fastest for both best fit and better fit. The difference between the algorithms was negligible as it was less than a second.

Best fit: Across all 3 instances, the results for the best fit were extremely close. The ACSA best fit in Figures 10, 11 and 13 was the only one to rise above 1 millisecond, and as such we can suggest that all the algorithms performed adequately when using best fit for the following instances. ABC was still the fastest, and ACSA was still the slowest.

Better fit: The trends of GA, CSGA and ACSA persisted throughout all three instances. The results tended to get worse in the same ratio as the increase in items, i.e. if the number of items doubled, so did time per item. From 50 to 100 items the time for the ACSA almost quadrupled even though the items only doubled. The time per item, i.e. 5.76 in Figure 11, doubled to 11.27 milliseconds per item. This suggests that ACSA, CSGA and GA perform substantially worse than the other 3 algorithms with respect to time. The difference in time between the FA, FAH and ABC was almost negligible. This is in contrast to the previous dataset where ABC came out

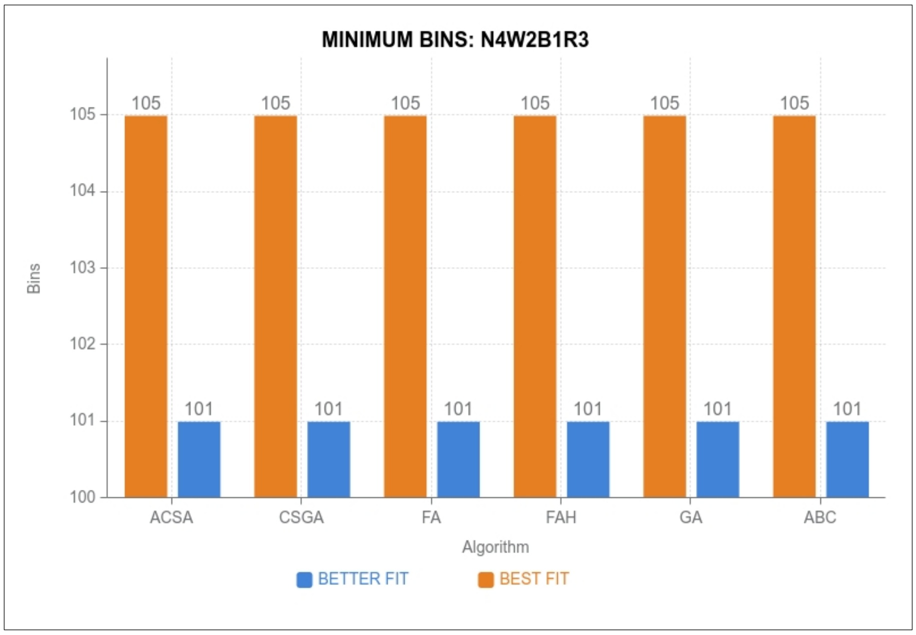


FIGURE 9. A graphical representation of the results of the N4W2B1R3 instance.

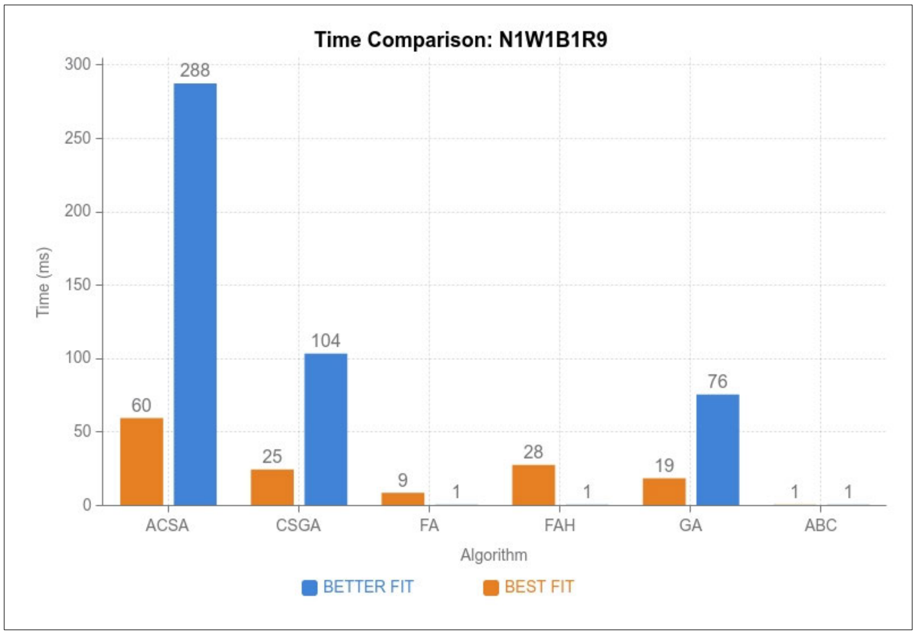


FIGURE 10. Computational time results for the N1W1B1R9 instance.

the clear winner. It is possible that the increase in capacity aided the firefly algorithms as the number of items increased. This could possibly be expanded on and tested in future work.

The results of the experiments obtained from Dataset 3 are presented in Table 12. These are the results for bins with a capacity of 100,000 but with a set number of items (200). This dataset contained the instances with the most complexity as it contained extremely large items with a lot of variance in item weight. As such, we expected to see the best fit struggle, as shown in Figures 13, 14 and 15.

The algorithms performed almost identically, although in Figure 13 the GA was the only heuristic to obtain the optimal solution. This was the only instance in the dataset where this occurred, and better fit found the optimal solution with the remaining heuristics in every other instance. We saw the best fit struggle in every instance to find the optimal solution and whilst four or five boxes may not

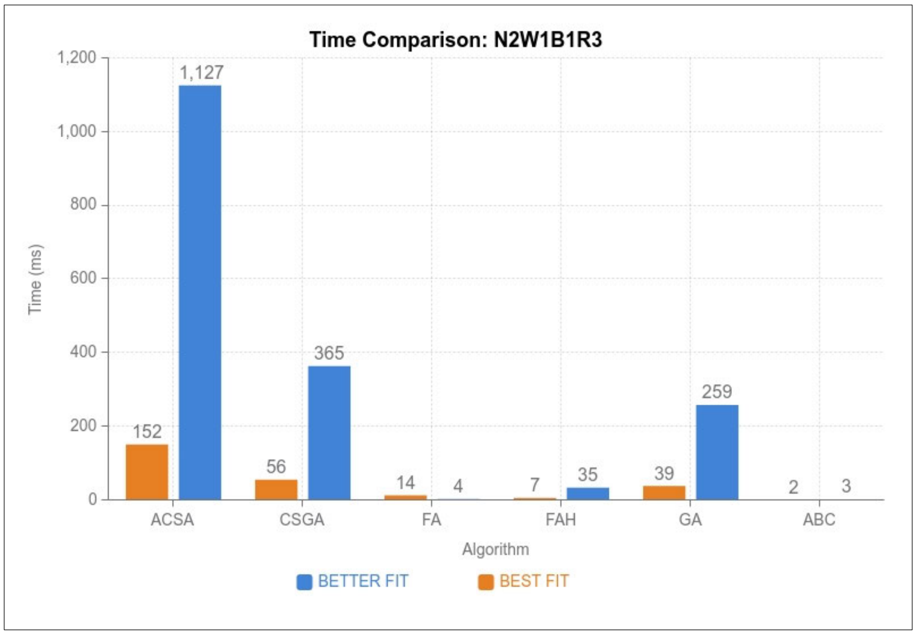


FIGURE 11. Computational time results for the N2W1B1R3 instance.

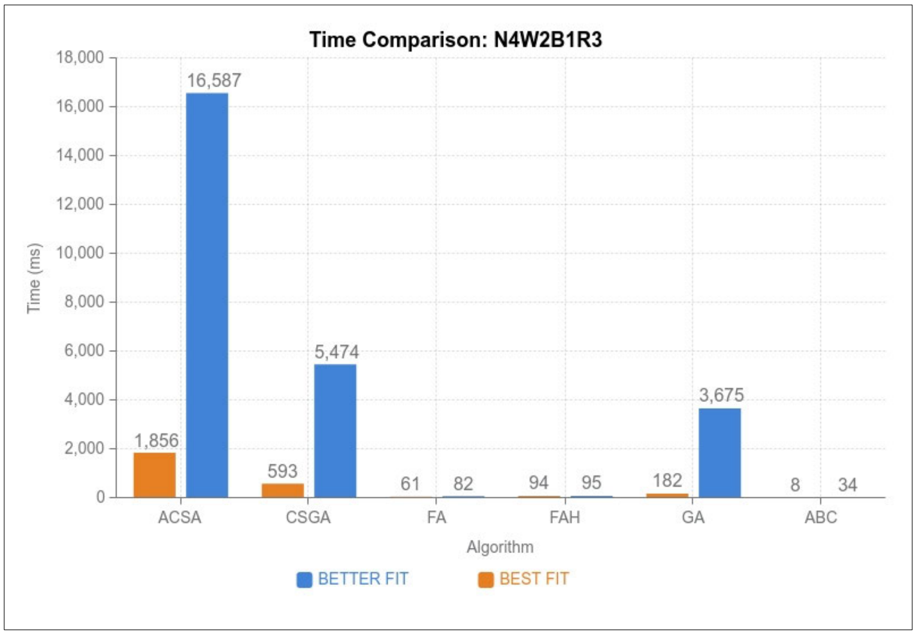


FIGURE 12. Computational time results for the N4W2B1R3 instance.

seem like a lot the ideal items per bin for these instances is between 3 and 5 and as such the wastage per box was high compared to the recommended solution. The ACSA, CSGA and GA appeared to slightly outperform the other algorithms, for instance, HARD5. All algorithms performed ideally under the better fit heuristic even for these complex problems. The performance in terms of time will be discussed further.

As with the previous datasets, we present the computational time comparisons in Figures 16, 17 and 18. We notice a trend occurring across all three instances. The shape of the graph was the same, although the numbers changed. The number of items and the capacity across all the instances in Dataset 3wa the same, and as such, we expected the results to be fairly similar across all three instances shown above. We saw the same trend as the previous tests. The ABC

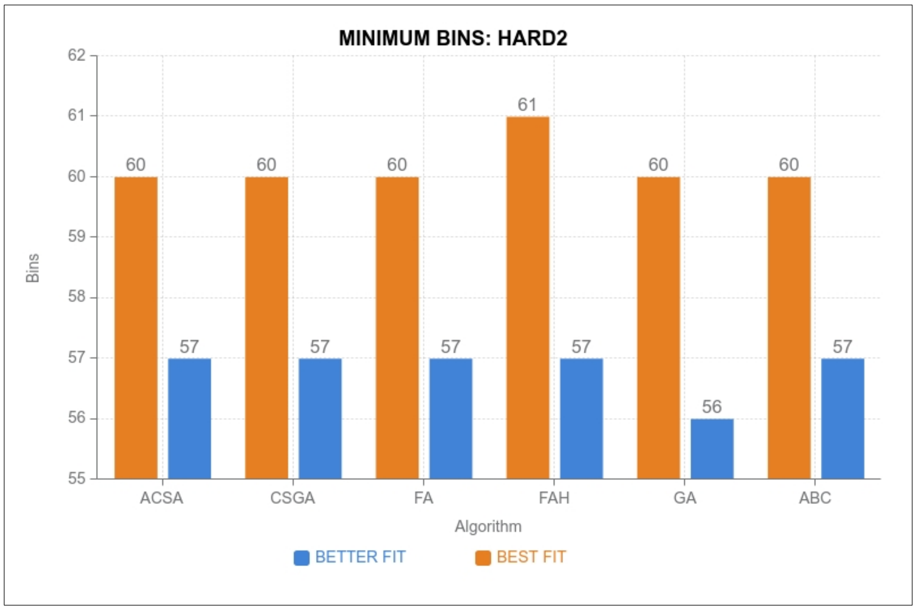


FIGURE 13. A graphical representation of the results of the HARD2 instance.

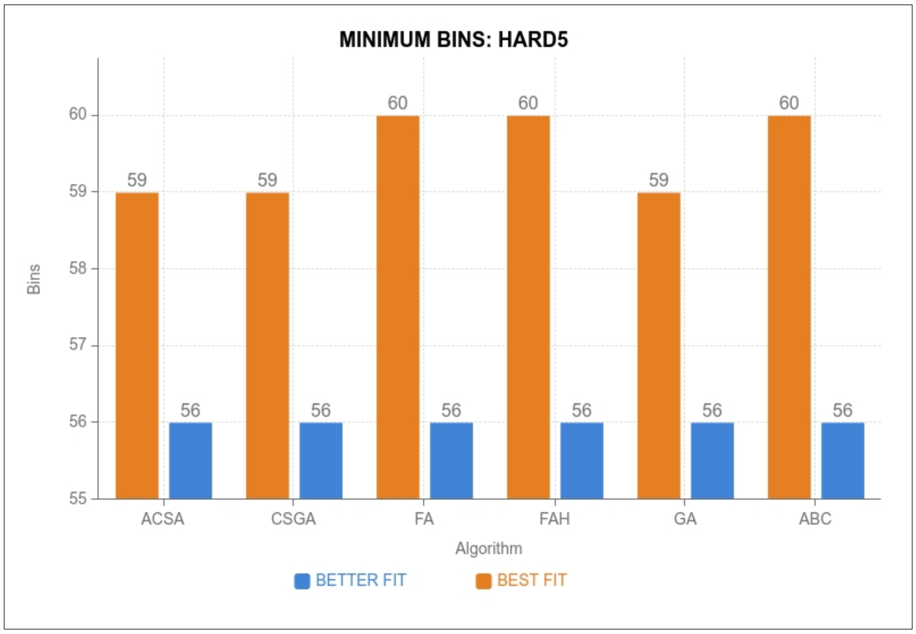


FIGURE 14. A graphical representation of the results of the HARD5 instance.

performed best in both the best fit and better fit methods. The ACSA performed the worst.

Best fit: The time differences across all six algorithms was negligible. The ACSA performed the worst by a long way in terms of a ratio but in actual the difference was less than half a millisecond and such we cannot consider it substantial.

Better fit: The ACSA, CSGA and GA were once again the three slower algorithms when using better fit. We can suggest that as the number of items increases the amount of time substantially increases for these algorithms (as discussed before). The increase in time from 100 to 500 items is in line with the number observed with 200 items (these current instances). This suggests that these three algorithms are heavily affected by the item number and not capacity. The FA and FAH, however, did appear to be affected by the capacity and perform extremely well even with 200 items. The ABC performed the best, but the difference between the ABC and the Firefly-based algorithms was negligible. However, there was a significant difference between the three worst performing algorithms and the three best performing.

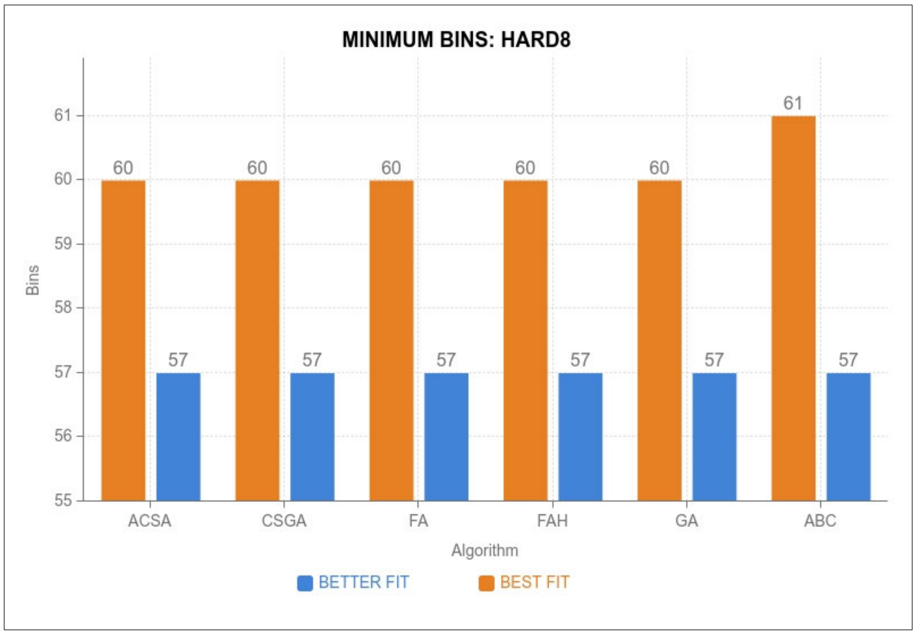


FIGURE 15. A graphical representation of the results of the HARD8 instance.

TABLE 12. Results obtained from datasets 3 (Hard dataset).

Instance	Number of items	Capacity	Best known	ACSA		CSGA		FA		FAH		GA		ABC	
				*	**	*	**	*	**	*	**	*	**	*	**
HARD0	200	100000	56	59	56	59	56	60	56	59	56	59	56	59	56
HARD1	200	100000	57	60	57	60	57	60	57	60	57	60	57	60	57
HARD2	200	100000	56	60	57	60	57	60	57	61	57	60	56	60	57
HARD3	200	100000	55	58	55	59	55	59	56	59	56	59	55	59	55
HARD4	200	100000	57	60	57	60	57	61	57	61	57	60	57	61	57
HARD5	200	100000	56	59	56	59	56	60	56	60	56	59	56	60	56
HARD6	200	100000	57	60	57	60	57	60	57	60	57	60	57	61	57
HARD7	200	100000	55	58	55	58	55	59	55	58	55	58	55	59	55
HARD8	200	100000	57	60	57	60	57	60	57	60	57	60	57	61	57
HARD9	200	100000	56	59	55	59	56	60	56	60	56	59	56	60	56

Key: * → This algorithm was tested with the best fit heuristic
** → This algorithm was tested with the better fit heuristic

C. HYBRID VS NON-HYBRID CUCKOO SEARCH

Looking at Tables 10, 11 and 12, both the hybrid and non-hybrid Cuckoo Search (ACSA and CSGA) attained very similar results when it came to the number of bins required. We, therefore, cannot conclude anything in that respect. In terms of time it is, however, a different story. When looking at both Figures 19 and 20, we can see a substantial improvement in the time for the CSGA compared to the ACSA. The CSGA was consistently 2.5 to 4 times faster than the ACSA. This was a massive improvement on the standard cuckoo search algorithm, that can most likely be attributed to the less complex operators in the CSGA compared to the Lévy

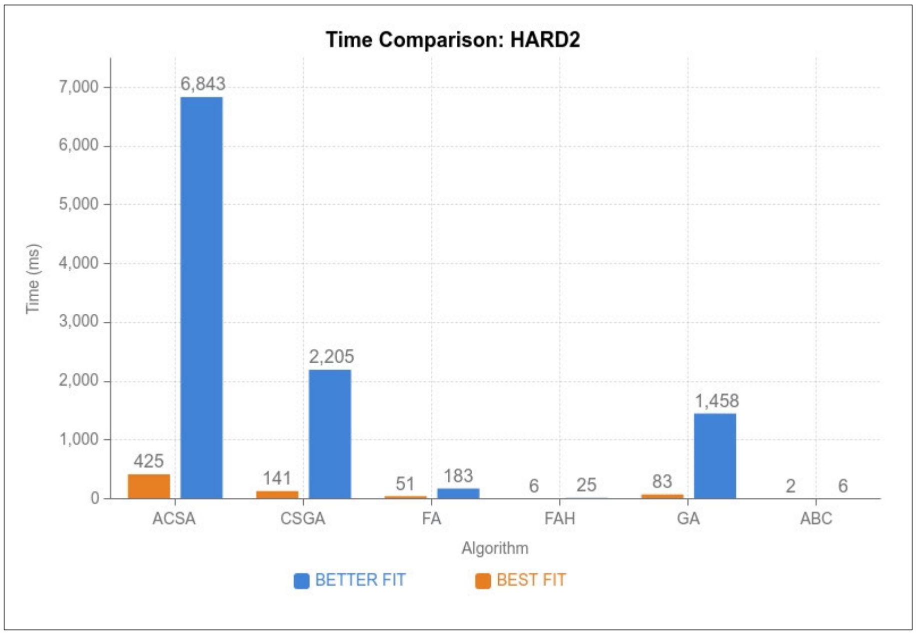


FIGURE 16. Computational time results for the HARD2 instance.

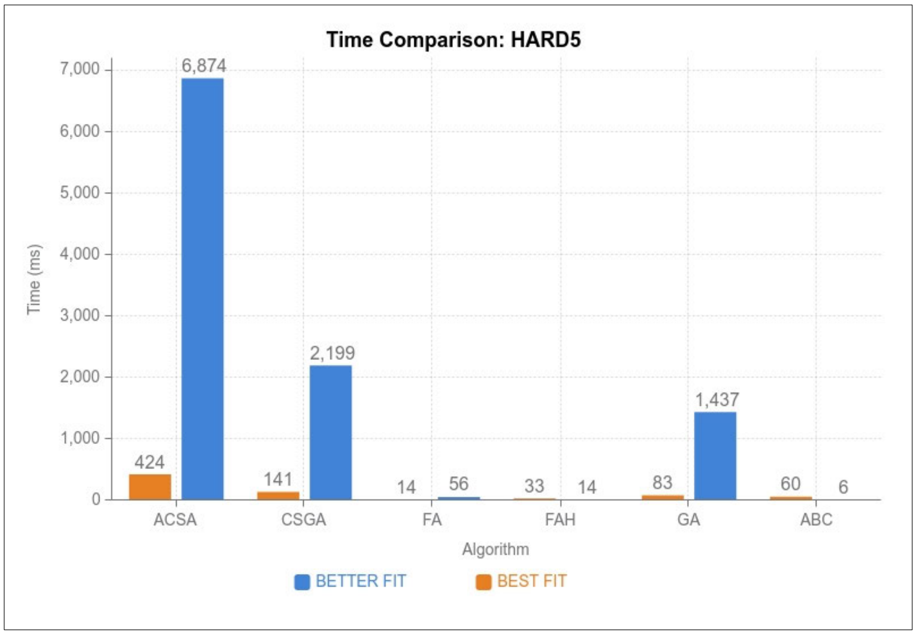


FIGURE 17. Computational time results for the HARD5 instance.

walk operators in the ACSA. Even with the improvement in time, the CSGA and ACSA were the two worst performing algorithms for all 3 datasets in almost every instance.

Both the ACSA and CSGA performed relatively well. For easy dataset, both algorithms achieved optimal results. However, the time required differed significantly, not just for both algorithms but also for the underlying heuristics. The ACSA algorithm with the best fit heuristic required approximately 70 milliseconds for the instances with 50 items. This steadily increased as items reached 500 whereas the better fit heuristic started off at approximately 341 milliseconds for 50 items and escalated to approximately 96,000 milliseconds for 500 items. Therefore, for the easy dataset with the Adaptive Cuckoo Search algorithm, the best fit heuristic produced optimal results with far less computational time and was the preferable underlying heuristic. Similarly, the CSGA hybrid required between 30 to 920 milliseconds with the best fit heuristic and 130 to 32,000 milliseconds for the better

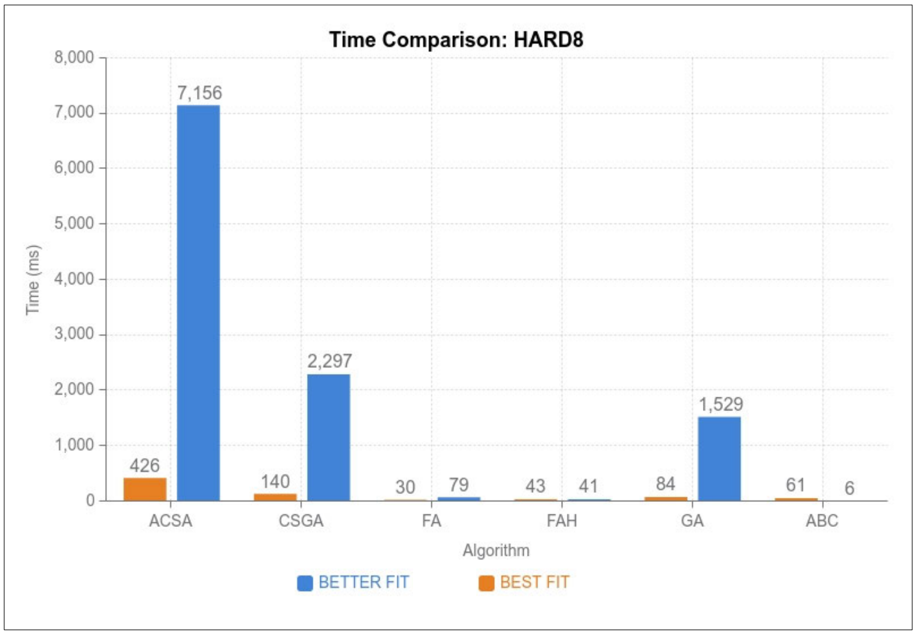


FIGURE 18. Computational time results for the HARD8 instance.

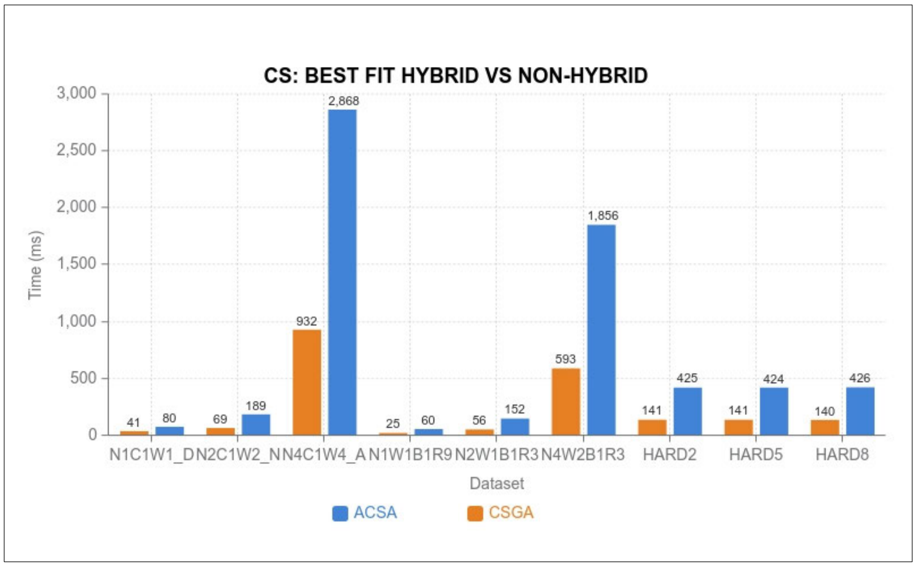


FIGURE 19. Computational time results for CS Hybrid and Non-Hybrid (best fit).

fit heuristic. Hence, the same can be said for the hybrid algorithm; the best fit was the superior underlying heuristic.

Both algorithms performed equivalently for the medium dataset. A better fit was able to produce optimal results for three instances that best fit failed to, and an improved packing schema for one instance that best fit was not able to. However, better fit still required between 5 to 10 times more milliseconds to achieve these results. In this case, a trade-off between the quality of solution and time required must be considered to determine which underlying algorithm is preferable.

Finally, for the hard dataset, all instances were tested. The ACSA algorithm with the best fit did not achieve any optimal results. An average of $n + 3$ bins was produced, where n is the optimal number. With the better fit heuristic, however, optimal results were achieved for 9 out of 10 of the instances. With respect to time, better fit required approximately 16 times more milliseconds. The CSGA produced a similar number of bins for all instances of the hard dataset with both the underlying heuristics, and this resulted in a similar difference in computational time. Therefore, once again,

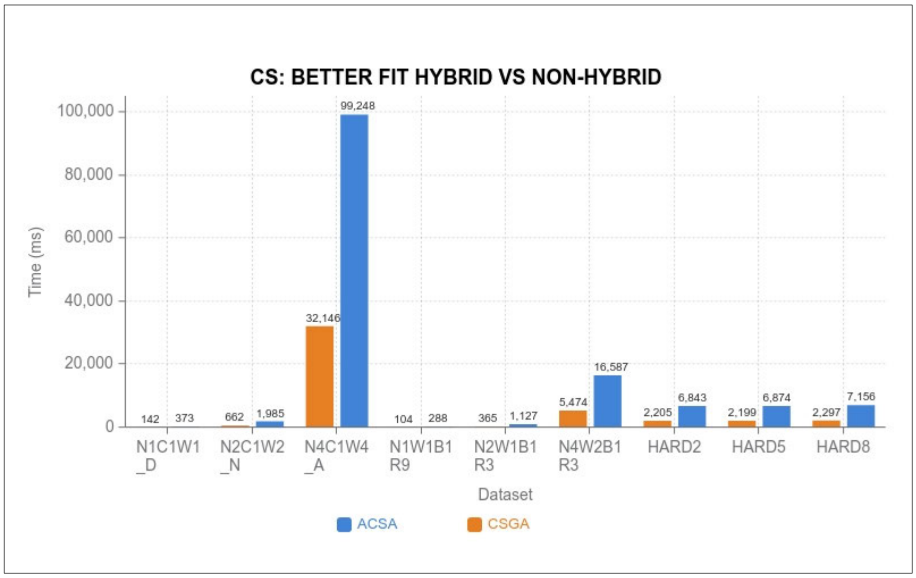


FIGURE 20. Computational time results for CS Hybrid and Non-Hybrid (Better fit).

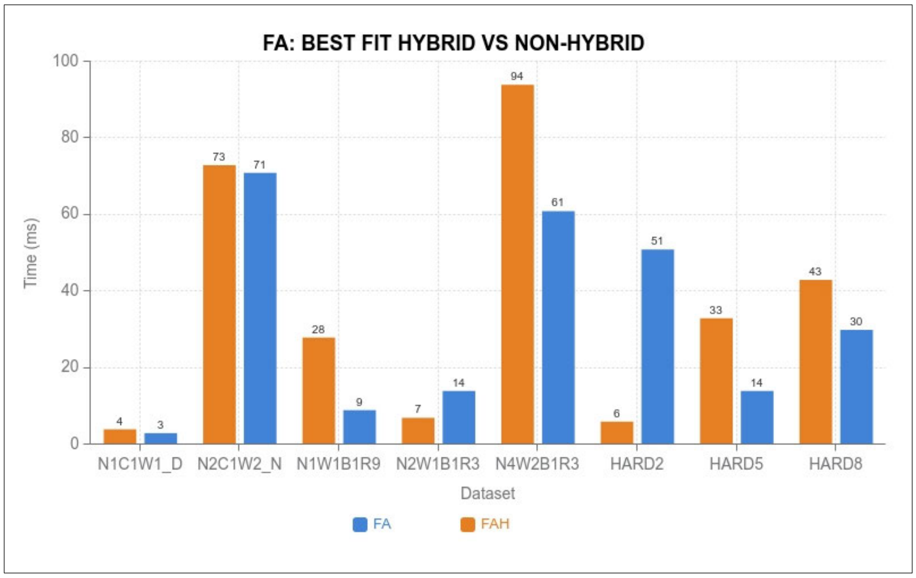


FIGURE 21. Computational time results for FA and FAH (best fit).

the trade-off between quality of solution and time required must be considered.

D. HYBRID VS NON-HYBRID FA

In terms of bin packing results, both algorithms performed equally well with both achieving optimal results with a better fit and achieving close to optimal with the best fit (when optimal was not achieved). No statement can be made about the effectiveness of the hybrid in terms of bin packing. In terms of computational time consumed by each algorithm to find the desired solutions, both the FAH and FA obtained extremely similar results with no pattern emerging from any of the results other than HARD instances when using better fit. For these instances, the FAH outperformed the FA. The time difference was less than a millisecond and was, therefore, negligible. We can suggest that the FAH provides no meaningful improvement to the FA both in terms of bin packing results and time. The Firefly algorithms both performed extremely well overall for all instances second only to the ABC.

As mentioned above, FA and FAH were implemented with two underlying heuristics: best fit and better fit. First, we considered the performance for the easy data set wherein the number of items ranged from 50 to 500. On the whole, the best fit FA and FAH (Figure 21) produced optimal or near optimal results. The better fit FA and FAH (Figure 22)

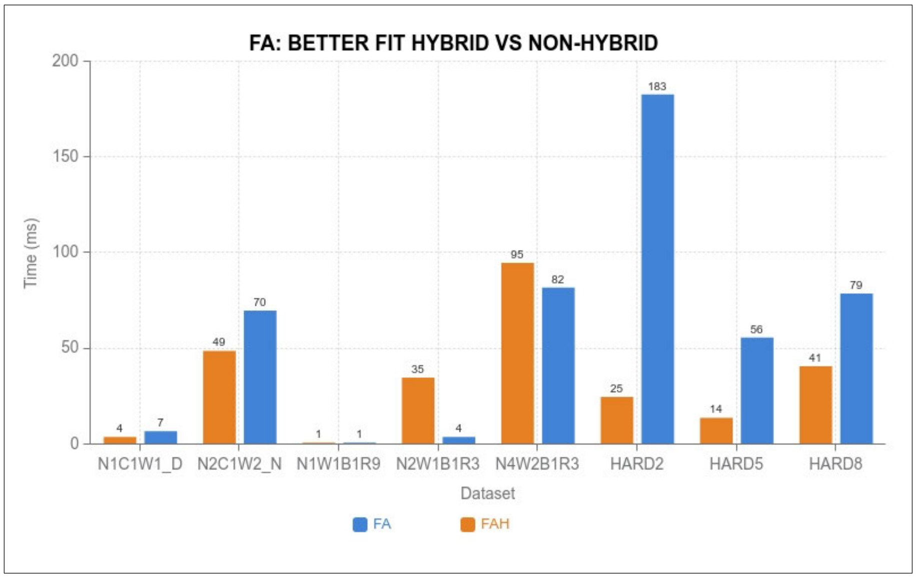


FIGURE 22. Computational time results for FA and FAH (Better fit).

produced only optimal results with the minimum number of bins required being packed. It should be noted that best fit FAH did not see any improvement over best fit FA. It is likely that increasing the number of iterations in the FAH would lead to an improvement.

With regards to time, the best fit FA was more efficient than better fit FA. This is due to the fact that the better fit heuristic is computationally more expensive than the best fit heuristic. The time complexity of better fit is worse than that of the best fit since better fit involves more search operations. Best fit FAH and better fit FAH follow similarly with best fit FAH usually performing faster for the same reasons.

There are a small number of cases where better fit FAH performs faster than best fit FAH, and this is likely to be due to the existence of highly effective mutations occurring, leading to all fireflies having similar or the same brightness, so solution updates were minimal. The behaviour of this nature is, however, anomalous. It should be noted that, on the whole, the best fit FAH is slower than the best fit FA and better fit FAH is slower than the better fit FA. This is because the hybridized algorithm involves more operations when mutations occur, thereby increasing the computational expense of the algorithm. The best fit FA and best fit FAH almost never produce optimal results, mostly settling with near optimal results. Better fit FA and better fit FAH almost always produce optimal results only occasionally outputting near optimal results.

V. CONCLUSION

This study investigated six nature-inspired, population-based metaheuristic algorithms to solve the classical one-dimensional bin-packing problem. The representative algorithms considered for the problem at hand included FA, FAH, CSA, CSGA, ABC and GA, respectively. Solving the NP-hard optimization problem considered in this paper required a yardstick to assess the quality of the generated solutions. The best fit and better fit heuristics were selected to serve this purpose through their role as underlying heuristics. Solutions themselves were a unique ordering of items to be passed to these underlying heuristics.

These metaheuristics were implemented and tested on a set of standard benchmark datasets, of which specific instances were chosen non-trivially. In the comparison of these algorithms, several observations regarding the solution and the underlying heuristics were made. It is worth noting that the best fit heuristic attained optimal solutions for instances of the easy dataset requiring smaller capacity bins. However, as the complexity of the instances increased, the ability to produce high quality results decreased. Nevertheless, this heuristic is able to produce near-optimal solutions and a good packing schema for most instances. On the other hand, utilizing better fit as the underlying heuristic results in optimal solutions almost all the time, regardless of capacity and number of items. The downfall of this heuristic is the computational time required to produce these solutions, especially with the ACSA, CSGA, and GA metaheuristics. Therefore, a trade-off between the quality of solution and computational time required must be made when choosing which underlying heuristic performs better. For the FA, FAH, and ABC metaheuristics, the better fit heuristic proved to be more beneficial because the increase in computational time was not substantial and the quality of the results was still sufficiently maintained.

Throughout the testing of this study, the ABC metaheuristic was the fastest performed algorithm, followed by the FA and FAH. Each metaheuristic contains features worth exploring, and therefore, for future work, it would be interesting to investigate how to lower the computational cost of the ACSA, CSGA, and GA when applying it to the bin-packing problem. In addition, beside the methods used in this paper, other existing state-of-the-art representative computational intelligence algorithms can also be employed to solve the one-dimensional BPP, like the monarch butterfly optimization (MBO) [69], earthworm optimization algorithm (EWA) [70], elephant herding optimization (EHO) [71], moth search (MS) algorithm [72], symbiotic organisms search (SOS) algorithm [73], [74], and many more. Moreover, investigating alternative methods to generate the initial population of the metaheuristics may prove beneficial, as this study involved simply randomizing initial positions.

CONFLICT OF INTERESTS

The authors declare that there is no conflict of interests regarding the publication of the paper.

REFERENCES

[1] A. M. Connor and K. Shea, "A comparison of semi-deterministic and stochastic search techniques," in *Evolutionary Design and Manufacture*, I. C. Parmee, Ed. London, U.K.: Springer, 2000, doi: 10.1007/978-1-4471-0519-0_23.

[2] S. I. Gass and C. M. Harris, "Best-fit decreasing algorithm," in *Encyclopedia of Operations Research and Management Science*, S. I. Gass and C. M. Harris, Eds. New York, NY, USA: Springer, 2001, doi: 10.1007/1-4020-0611-X_70.

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

[4] M. Gourgand, N. Grangeon, and N. Klement, "An analogy between bin packing problem and permutation problem: A new encoding scheme," *IFIP Adv. Inf. Commun. Technol.*, vol. 438, pp. 572–579, 2014, doi: 10.1007/978-3-662-44739-0_70.

[5] I. Fister, I. Fister, Jr., X. S. Yang, and J. Brest, "A comprehensive review of firefly algorithms," *Swarm Evol. Comput.*, vol. 13, pp. 34–46, 2013.

[6] X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*. York, U.K.: Luniver Press, 2010.

[7] M. Crepinšek, M. Mernik, and S. H. Liu, "Analysis of exploration and exploitation in evolutionary algorithms by ancestry trees," *Int. J. Innov. Comput. Appl.*, vol. 3, no. 1, pp. 11–19, 2011, doi: 10.1504/IJICA.2011.037947.

[8] X. S. Yang, "Firefly algorithm, Levy flights and global optimization," in *Research and Development in Intelligent Systems XXVI*. London, U.K.: Springer, 2010, pp. 209–218.

[9] R. Yesodha and T. Amudha, "Effectiveness of firefly algorithm in solving bin packing problem," *Int. J. Adv. Res. Effectiveness FireFly Algorithm Solving Bin Packing Problem*, vol. 3, no. 5, pp. 1003–1010, 2013.

[10] X.-S. Yang and X. He, "Firefly algorithm: Recent advances and applications," *Int. J. Swarm Intell.*, vol. 1, no. 1, pp. 36–50, 2013, doi: 10.1504/IJSI.2013.055801.

[11] A. Layeb and Z. Benayad, "A novel firefly algorithm based ant colony optimization for solving combinatorial optimization problems," *Int. J. Comput. Sci. Appl.*, vol. 11, no. 2, pp. 19–37, 2014.

[12] W. Long, S. H. Cai, J. Jiao, Y. Chen, and Y. Huang, "Firefly algorithm for solving constrained optimization problems and engineering applications," *J. Central South Univ. Sci. Technol.*, vol. 46, no. 4, pp. 1260–1267, 2015, doi: 10.11817/j.issn.1672-7207.2015.04.013.

[13] X.-S. Yang, S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, Dec. 2009, pp. 210–214.

[14] Z. Zendaoui and A. Layeb, "Adaptive cuckoo search algorithm for the bin packing problem," in *Modelling and Implementation of Complex Systems*. Cham, Switzerland: Springer, 2016, pp. 107–120.

[15] A. S. Joshi, O. Kulkarni, G. M. Kakandikar, and V. M. Nandedkar, "Cuckoo search optimization—A review," *Mater. Today, Proc.*, vol. 4, no. 8, pp. 7262–7269, 2017, doi: 10.1016/j.matpr.2017.07.055.

[16] M. Mareli and B. Twala, "An adaptive Cuckoo search algorithm for optimization," *Appl. Comput. Inform.*, vol. 14, no. 2, pp. 107–115, 2018, doi: 10.1016/j.aci.2017.09.001.

[17] E. Valian, S. Mohanna, and S. Tavakoli, "Improved cuckoo search algorithm for feed forward neural network training," *Int. J. Artif. Intell. Appl.*, vol. 2, no. 3, pp. 36–43, 2011, doi: 10.5121/ijaia.2011.2304.

[18] A*Comparative Study of Cuckoo Algorithm and Ant Colony Algorithm in Optimal Path Problems Guanyu Wang*. Accessed: Jul. 28, 2020. [Online]. Available: https://www.matec-conferences.org/articles/matecconf/pdf/2018/91/matecconf_eitce2018_03003.pdf

[19] X. Li and M. Yin, "Modified cuckoo search algorithm with self-adaptive parameter method," *Inf. Sci.*, vol. 298, pp. 80–97, Mar. 2015, doi: 10.1016/j.ins.2014.11.042.

[20] W. C. E. Lim, G. Kanagaraj, and S. G. Ponnambalam, "A hybrid cuckoo search-genetic algorithm for hole-making sequence optimization," *J. Intell. Manuf.*, vol. 27, no. 2, pp. 417–419, 2014, doi: 10.1007/s10845-014-0873-z.

[21] N. B. Davies and M. D. L. Brooke, "Cuckoos versus reed warblers: Adaptations and counteradaptations," *Animal Behaviour*, vol. 36, no. 1, pp. 262–284, 1988.

[22] M. Shehab, A. T. Khader, and M. A. Al-Betar, "A survey on applications and variants of the cuckoo search algorithm," *Appl. Soft Comput.*, vol. 61, pp. 1041–1059, Dec. 2017.

[23] Wikipedia Contributors. (Aug. 7, 2020). *NP-Hardness. In Wikipedia, the Free Encyclopedia*. Accessed: Aug. 7, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=NP-hardness&oldid=971690665

[24] E. López-Camacho, H. Terashima-Marín, G. Ochoa, and S. E. Conant-Pablos, "Understanding the structure of bin packing problems through principal component analysis," *Int. J. Prod. Econ.*, vol. 145, no. 2, pp. 488–499, Oct. 2013, doi: 10.1016/j.ijpe.2013.04.041.

[25] K. Fleszar and C. Charalambous, "Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem," *Eur. J. Oper. Res.*, vol. 210, no. 2, pp. 176–184, Apr. 2011, doi: 10.1016/j.ejor.2010.11.004.

[26] 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.

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

[28] J. Levine and F. Ducatelle, "Ant colony optimization and local search for bin packing and cutting stock problems," *J. Oper. Res. Soc.*, vol. 55, no. 7, pp. 705–716, Jul. 2004, doi: 10.1057/palgrave.jors.2601771.

[29] M. Gendreau and J.-Y. Potvin, "Metaheuristics in combinatorial optimization," *Ann. Oper. Res.*, vol. 140, no. 1, pp. 189–213, 2005.

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

[31] S. Martello, *Knapsack Problems: Algorithms and Computer Implementations*. (Wiley-Interscience Series in Discrete Mathematics and Optimization). New York, NY, USA: Wiley, 1990.

[32] D. S. Abdul-Minaam, W. M. E. S. Al-Mutairi, M. A. Awad, and W. H. El-Ashmawi, "An adaptive fitness-dependent optimizer for the one-dimensional bin packing problem," *IEEE Access*, vol. 8, pp. 97959–97974, 2020.

[33] J. M. Abdullah and T. Ahmed, "Fitness dependent optimizer: Inspired by the bee swarming reproductive process," *IEEE Access*, vol. 7, pp. 43473–43486, 2019.

[34] M. Abdel-Basset, G. Manogaran, L. Abdel-Fatah, and S. Mirjalili, "An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems," *Pers. Ubiquitous Comput.*, vol. 22, nos. 5–6, pp. 1117–1132, Oct. 2018.

[35] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[36] W. H. El-Ashmawi and D. S. A. Elminaam, "A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem," *Appl. Soft Comput.*, vol. 82, Sep. 2019, Art. no. 105565.

[37] M. Jain, V. Singh, and A. Rani, "A novel nature-inspired algorithm for optimization: Squirrel search algorithm," *Swarm Evol. Comput.*, vol. 44, pp. 148–175, Feb. 2019.

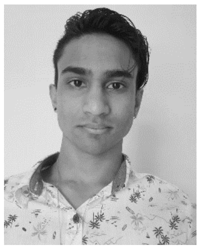
[38] A. K. Bhatia, M. Hazra, and S. K. Basu, "Better-fit heuristic for one-dimensional bin-packing problem," in *Proc. IEEE Int. Advance Comput. Conf.*, Mar. 2009, pp. 193–196, doi: 10.1109/IADCC.2009.4809005.

[39] D. Karaboga, "Artificial bee colony algorithm," *Scholarpedia*, vol. 5, no. 3, p. 6915, 2010, doi: 10.4249/scholarpedia.6915.

[40] J.-H. Liang and C.-H. Lee, "A modification artificial bee colony algorithm for optimization problems," *Math. Problems Eng.*, vol. 2015, pp. 1–14, Mar. 2015. Accessed: Aug. 27, 2020. [Online]. Available: https://www.hindawi.com/journals/mpe/2015/581391/

[41] M. Gourgand, N. Grangeon, and N. Klement, ''An analogy between bin packing problem and permutation problem: A new encoding scheme,'' in *Proc. IFIP Int. Conf. Adv. Prod. Manage. Syst.* Berlin, Germany: Springer, Sep. 2014, pp. 572–579.
[42] N. Bansal, A. Lodi, and M. Sviridenko, ''A tale of two dimensional bin packing,'' in *Proc. 46th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Oct. 2005, pp. 657–666.
[43] G. Scheithauer, ''LP-based bounds for the container and multi-container loading problem,'' *Int. Trans. Oper. Res.*, vol. 6, no. 2, pp. 199–213, 1999.
[44] J. Terno, G. Scheithauer, U. Sommerweiß, and J. Riehme, ''An efficient approach for the multi-pallet loading problem,'' *Eur. J. Oper. Res.*, vol. 123, no. 2, pp. 372–381, Jun. 2000.
[45] (2020). *Fe.up.pt*. Accessed: Aug. 27, 2020. [Online]. Available: PopulationalMetaheuristics/SimulatedAnnealingAndrypintoInes Domingues_LuisRocha_HugoAlves_SusanaCruz.pptx
[46] D. Bertsimas and J. Tsitsiklis, ''Simulated annealing,'' *Statist. Sci.*, vol. 8, no. 1, pp. 10–15, 1993.
[47] E. Aarts, J. Korst, and W. Michiels, ''Simulated annealing,'' in *Search Methodologies*. Boston, MA, USA: Springer, 2005, pp. 187–210.
[48] R. L. Rao and S. S. Iyengar, ''Bin-packing by simulated annealing,'' *Comput. Math. Appl.*, vol. 27, no. 5, pp. 71–82, 1994.
[49] E. Sonuc, B. Sen, and S. Bayir, ''Solving bin packing problem using simulated annealing,'' in *Proc. 65th ISERD Int. Conf.*, Mecca, Saudi Arabia, 2017.
[50] E. Falkenauer and A. N. D. Delchambre, ''A genetic algorithm for bin packing and line balancing,'' in *Proc. IEEE Int. Conf. Robot. Automat.*, May 1992, pp. 1186–1192, doi: 10.1109/robot.1992.220088.
[51] C. F. A. Alvim, C. C. Ribeiro, F. Glover, and J. D. Aloise, ''A hybrid improvement 715 heuristic for the one-dimensional bin packing problem,'' *J. Heuristics*, vol. 10, no. 2, pp. 205–229, 2004.
[52] N. Mohamadi, ''Application of genetic algorithm for the bin packing problem with a new representation scheme,'' *Math. Sci.*, vol. 4, no. 3, pp. 253–266, Oct. 2010.
[53] (Aug. 21, 2020). *A Comprehensive Review of Swarm Optimization Algorithms*. [Online]. Available: https://openi.nlm.nih.gov/detailedresult?img=PMC4436220_pone.0122827.g001
[54] J. G. M. Coffman and D. Johnson, ''Approximation algorithms for bin-Packing, an updated survey,'' in *Algorithm Design for Computer Systems Design*, G. Ausiello, M. Lucertini, P. Serafini, Eds. New York, NY, USA: Springer, 1984, pp. 49–106.
[55] K. Fleszar and K. S. Hindi, ''New heuristics for one-dimensional bin-packing,'' *Comput. Oper. Res.*, vol. 29, no. 7, pp. 821–839, Jun. 2002.
[56] Electricalvoice. (Aug. 10, 2017). *Genetic Algorithm: Advantages & Disadvantages*. Accessed: Aug. 21, 2020. [Online]. Available: https://electricalvoice.com/genetic-algorithm-advantages-disadvantages/
[57] X. Yang, ''Genetic algorithms,'' in *Nature-Inspired Optimization Algorithms*. London, U.K.: Elsevier, 2014, pp. 77–87, doi: 10.1016/b978-0-12-416743-8.00005-1.
[58] E. G. Coffman, J. Y.-T. Leung, and D. W. Ting, ''Bin packing: Maximizing the number of pieces packed,'' *Acta Inf.*, vol. 9, no. 3, pp. 263–271, Sep. 1978.
[59] K. L. Krause, V. Y. Shen, and H. D. Schwetman, ''Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems,'' *J. ACM*, vol. 22, no. 4, pp. 522–550, Oct. 1975.
[60] *BPP Datasets*. Accessed: Aug. 30, 2020. [Online]. Available: https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm
[61] T. Kucukyilmaz and H. E. Kiziloz, ''Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem,'' *Comput. Ind. Eng.*, vol. 125, pp. 157–170, Nov. 2018.
[62] T. Dokeroglu and A. Cosar, ''Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms,'' *Comput. Ind. Eng.*, vol. 75, pp. 176–186, Sep. 2014.
[63] A. Gherboudj, ''African buffalo optimization for one dimensional bin packing problem,'' *Int. J. Swarm Intell. Res.*, vol. 10, no. 4, pp. 38–52, Oct. 2019.
[64] S. A. G. Shirazi and M. B. Menhaj, ''A new genetic based algorithm for channel assignment problems,'' in *Computational Intelligence, Theory and Applications*. Berlin, Germany: Springer, 2006, pp. 85–91.
[65] C. Zhao, L. Jiang, and K. L. Teo, ''A hybrid chaos firefly algorithm for three-dimensional irregular packing problem,'' *J. Ind. Manage. Optim.*, vol. 16, no. 1, p. 409, 2020.
[66] G. G. Wang, S. Deb, and Z. Cui, ''Monarch butterfly optimization,'' *Neural Comput. Appl.*, vol. 31, no. 7, pp. 1995–2014, 2019.
[67] G.-G. Wang, S. Deb, and L. dos S. Coelho, ''Earthworm optimisation algorithm: A bio-inspired metaheuristic algorithm for global optimisation problems,'' *Int. J. Bio-Inspired Comput.*, vol. 12, no. 1, pp. 1–22, Jan. 2018.
[68] G.-G. Wang, S. Deb, and L. D. S. Coelho, ''Elephant herding optimization,'' in *Proc. 3rd Int. Symp. Comput. Bus. Intell. (ISCBI)*, Dec. 2015, pp. 1–5.
[69] G.-G. Wang, ''Moth search algorithm: A bio-inspired Metaheuristic algorithm for global optimization problems,'' *Memetic Comput.*, vol. 10, no. 2, pp. 151–164, Jun. 2018.
[70] M.-Y. Cheng and D. Prayogo, ''Symbiotic organisms search: A new metaheuristic optimization algorithm,'' *Comput. Struct.*, vol. 139, pp. 98–112, Jul. 2014.
[71] A. E. Ezugwu and D. Prayogo, ''Symbiotic organisms search algorithm: Theory, recent advances and applications,'' *Expert Syst. Appl.*, vol. 119, pp. 184–209, Apr. 2019.
[72] A. E. Ezugwu, V. Pillay, D. Hirasen, K. Sivanarain, and M. Govender, ''A comparative study of meta-heuristic optimization algorithms for 0–1 knapsack problem: Some initial results,'' *IEEE Access*, vol. 7, pp. 43979–44001, 2019.
[73] A. E.-S. Ezugwu, M. B. Agbaje, N. Aljojo, R. Els, H. Chiroma, and M. A. Elaziz, ''A comparative performance study of hybrid firefly algorithms for automatic data clustering,'' *IEEE Access*, vol. 8, pp. 121089–121118, 2020.
[74] M. B. Agbaje, A. E. Ezugwu, and R. Els, ''Automatic data clustering using hybrid firefly particle swarm optimization algorithm,'' *IEEE Access*, vol. 7, pp. 184963–184984, 2019.
[75] A. E. Ezugwu and F. Akutsah, ''An improved firefly algorithm for the unrelated parallel machines scheduling problem With sequence-dependent setup times,'' *IEEE Access*, vol. 6, pp. 54459–54478, 2018.

CHANALEÄ MUNIEN received the B.Sc. degree in computer science and information technology *(summa cum laude)* from the University of KwaZulu-Natal, in 2019. She is currently pursuing the Honours degree in computer science. Her honours research work is focused on deep learning and computer vision for medical image analysis, specifically the classification of breast cancer histology images. She intends to explore this research domain further in the near future.



SHIV MAHABEER received the B.Sc. degree in computer science and information technology from the University of KwaZulu-Natal, in 2019. He is currently pursuing the Honours degree in computer science. His honours research is focused on multi-agent deep reinforcement learning and its application to the field of social sciences. He intends furthering this research into his master's degree.

ESTHER DZITIRO received the B.Sc. degree in computer science and information technology from the University of KwaZulu-Natal, in 2019. She is currently pursuing the B.Sc. honours degree in computer science, with research focus on computer vision and machine learning. She intends pursuing her master's degree in this area.

SHARAD SINGH received the B.Sc. degree in computer science and information technology from the University of KwaZulu-Natal, in 2019. He is currently pursuing the B.Sc. honours degree in computer science, with his research in data analytics and machine learning. He is pursuing a career in this field.

SILULEKO ZUNGU received the B.Sc. degree in computer science and information technology from the University of KwaZulu-Natal, in 2019. He is currently pursuing the Honours degree in computer science, with his research area focusing on epidemiology using machine learning. He intends pursuing his master's degree in this area.

ABSALOM EL-SHAMIR EZUGWU (Member, IEEE) received the B.Sc. degree in mathematics with computer science and the M.Sc. and Ph.D. degrees in computer science from Ahmadu Bello University, Zaria, Nigeria. He is currently a Senior Lecturer with the School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, South Africa. He has published articles relevant to his research interest in internationally refereed journals and edited books, conference proceedings, and local journals. His research interests include parallel algorithms design in cloud and grid computing environments, artificial intelligence with specific interest in computational intelligence, and metaheuristic solutions to real-world global optimization problems. He is also a member of IAENG and ORSSA.