Research Article

Chanaleä Munien, Absalom E. Ezugwu*

Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications

https://doi.org/10.1515/jisys-2020-0117
received November 26, 2020; accepted March 19, 2021

Abstract: The bin-packing problem (BPP) is an age-old NP-hard combinatorial optimization problem, which is defined as the placement of a set of different-sized items into identical bins such that the number of containers used is optimally minimized. Besides, different variations of the problem do exist in practice depending on the bins dimension, placement constraints, and priority. More so, there are several important real-world applications of the BPP, especially in cutting industries, transportation, warehousing, and supply chain management. Due to the practical relevance of this problem, researchers are consistently investigating new and improved techniques to solve the problem optimally. Nature-inspired metaheuristics are powerful algorithms that have proven their incredible capability of solving challenging and complex optimization problems, including several variants of BPPs. However, no comprehensive literature review exists on the applications of the metaheuristic approaches to solve the BPPs. Therefore, to fill this gap, this article presents a survey of the recent advances achieved for the one-dimensional BPP, with specific emphasis on population-based metaheuristic algorithms. We believe that this article can serve as a reference guide for researchers to explore and develop more robust state-of-the-art metaheuristics algorithms for solving the emerging variants of the bin-parking problems.

Keywords: bin-packing problem, one-dimensional, nature-inspired, metaheuristic

1 Introduction

The bin-packing problem (BPP) is a combinatorial optimization problem that deals with packing a finite set of items with weights into a finite number of bins without exceeding the specified maximum capacity of the bins. This must be done such that the total number of required bins is minimized. The BPP can be viewed as a special case of the one-dimensional (1D) cutting-stock problem [1]. Research into the classical version of the BPP dates back into the early 1970s; however, the structure and applications of this problem have been studied since the 1930s [2]. In terms of computational complexity theory, this problem is known to be NP-hard [3]. Therefore, achieving optimal solutions can be time-consuming, specifically for problem instances with large sets of items.

The classical 1D BPP can be formally defined as: for a non-negative bin capacity, C, and a list of n items, $L = (a_1, a_2, \ldots, a_n)$, where a_i has a value (size or weight) $s(a_i)$ satisfying the constraint $0 \leq s(a_i) < C$ (size of item must not be larger than capacity of bin), the requirement is to determine the smallest integer t such

* **Corresponding author: Absalom E. Ezugwu,** School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Pietermaritzburg, 3201, South Africa, e-mail: ezugwua@ukzn.ac.za
Chanaleä Munien: School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Private Bag Box X54001, Durban 4000, South Africa, e-mail: 217013433@stu.ukzn.ac.za

that there is a partition of $L = W_1 \cup W_2 \ldots \cup W_t$. Here, the sum of the sizes of the items $a_i \in W_j$ must not exceed the maximum capacity. The set W_j can be seen as the contents of a bin with capacity, C [4].

There are several variations of the 1D BPP that fit different applications and have various types of constraints – a few of which will be briefly mentioned here. Coffman et al. [5] introduced the concept of maximizing the number of items that are packed into a bin, with the intention of modelling processor and storage allocation problems. Given a fixed number of bins, m, and a list of items, L, the aim of this variant is to pack the maximum subset of L into the bins in such a way that the maximum bin capacity is never exceeded. Krause et al. [6] proposed a variant of the BPP problem that places a restriction on the number of items allowed to be packed into each bin. Hence, for a given non-negative number, k, each bin may contain at most k items. The authors investigated this problem with task scheduling in mind. Similarly, Coffman et al. [7] described a generalization of the classical version of the BPP, where arrival and departure times were added to each item. Hence, the items in a bin occupy the space according to the specified time interval. This variant is more practical in particular applications such as problems associated with computer storage allocation.

Other research introduced multidimensional variations. The two-dimensional BPP requires allocating a finite set of rectangles into a minimized number of rectangular bins [4]. The three-dimensional problem (3D) proposed by Martello [8] requires packing a finite set of 3D rectangular items into a minimized number of bins. In both problems, the bins must be of a fixed size and overlapping must be avoided.

The classical 1D BPP can be modelled as an Integer Linear Program [9]. Firstly, let u be an upper bound on the minimum number of bins required to pack all the items. Assume that these bins are numbered as $1, 2, \ldots, u$. Let n be the number of items to pack. Two binary decision variables are introduced:

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

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

Minimize

$$\sum_{i=1}^{u} y_i$$

s.t.

$$\sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad (1 \le i \le u)$$

$$\sum_{i=1}^{u} x_{ij} = 1 \qquad (1 \le j \le n)$$

$$y_i \in \{0, 1\} \qquad (1 \le i \le u)$$

$$x_{ij} \in \{0, 1\} \qquad (1 \le i \le u; 1 \le j \le n).$$

The constraints enforce that the maximum capacity of the bin must not be exceeded, and that each item must only be packed into one bin. The goal of the 1D-BPP is to minimize the total number of bins, N, that are used to pack all n items, this can be estimated as follows:

$$N \ge \left\lceil \left(\sum_{i=1}^{n} S_i \right) \Big/ C \right\rceil,$$

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

Due to the complexity of this problem and the variety of real-world applications that can benefit from good quality solutions, many approaches have been used to solve the BPP. These approaches include approximate methods, metaheuristics, and hybrid metaheuristics. In addition, recently, the use of metaheuristic approaches for solving the BPP has become popular. These are incredibly powerful algorithms

that have successfully been applied to various combinatorial optimization problems [10], which include the population-based genetic algorithm (GA) [11], particle swarm optimization (PSO) [12], the tabu search [13], and many others. Some main benefits of utilizing metaheuristics are that they easily handle complex constraints present in real-life applications, and they produce high-quality solutions while requiring shorter computational time [14]. However, metaheuristics cannot randomly be applied to optimization problems, but rather, a significant amount of knowledge on the subject is required to create a proper metaheuristic implementation [10] – an example of this is choosing a good representation scheme to fit the problem at hand.

The aim of this study is to highlight recent nature-inspired metaheuristics that have been applied to solve the classical 1D BPP and to understand the benefits of this approach, which includes efficiency and high-quality solutions. This survey would invariably provide the much needed guidance to future enthusiastic researchers and industrial practitioners alike, who have interest in adapting the nature-inspired metaheuristic algorithms to solve the well-known BPP. Specifically, we may summarize the technical contributions of this article as follows:

- A state-of-the-art review of widely used metaheuristic algorithms for the 1D BPP.
- Critical analysis of design concepts and solution representation for the implementation of BPP using nature-inspired population-based metaheuristic algorithms.
- A comparative evaluation of existing literature results for the BPP.
- Suggestions of trending practical application areas for the BPP.

The rest of this work is structured as follows: Section 2 provides an overview of approximate algorithms and the recent metaheuristics that have been used as the foundation of the adapted nature-inspired metaheuristics that solve the BPP. Section 3 provides insights into the aforementioned nature-inspired metaheuristics by elaborating on their key features. Section 4 provides a comparison of results for each of these metaheuristics. Section 5 presents some of the most intriguing recent application areas of the BPP. Finally, the concluding remarks and future research direction are discussed in Section 6.

2 Literature review

2.1 Approximate algorithms

Approximate algorithms were the first approach to solve the classical 1D BPP. There are two types of approximation algorithms: online and offline. In online algorithms, the items are considered one by one, with no knowledge of succeeding items. Hence, the placement of the current item is based solely on the size of the current item and the already packed items [15]. A compelling concept that is introduced in the context of online algorithms is that of bounded-space algorithms. Bins can either be open or closed, and items must only be packed into open bins. When a new bin is added, it is given the status of "open." In some algorithms, the bin may be closed due to the inability to add items to the contents (as it may result in exceeding the maximum capacity). Therefore, a bin-packing algorithm is considered a k-bounded-space algorithm. This means that if there are k bins open and a single item is packed into a new bin, then one of the open bins must be closed [15].

The Next Fit (NF) heuristic packs the maximum number of items that the current bin can contain, and when there is no available space for the next item, the current bin is closed, a new bin is opened, and this bin becomes the current bin [16]. The First Fit (FF) heuristic differs from NF in that the bins are not closed. Each item is packed into the first bin that has sufficient space. If an item cannot fit into any of the available bins, a new bin is opened [16]. The Best Fit (BF) heuristic packs an item into the best, or most suitable, bin – that is, the bin with the smallest available space after including the current item into it. As in FF, if no bin can fit the current item, a new bin is opened [16].

On the other hand, offline algorithms have also been used to solve this problem. Famous examples are the NF-Decreasing, BF-Decreasing, and FF-Decreasing algorithms. This type of heuristic processes the data before packing the items into the bin. Generally, this means that the items are rearranged and sorted in increasing or decreasing order [4]; thereafter, these algorithms fill the bins just as the original NF, BF, and FF algorithms do. These offline algorithms are not guaranteed to return optimal solutions every time.

An example of an algorithm that is both online and offline is the Better-Fit heuristic algorithm (BFH) [17], which removes an existing item from a bin and replaces it with the current item if the current item better fills the bin. Moreover, if the packing of the current item results in a smaller available space than the packing of the existing item, then the latter item is removed from the bin it is in. This "replaced item" is then packed using the BFH. This continues for all items until better-fit cannot pack a replaced item. In this case, the BF heuristic is used to pack the object. This algorithm outperforms the BF heuristic in every instance. The performance of the algorithm is based on the sequence of items. If the items are arranged in decreasing order, then the results are equivalent to that of BF-Decreasing, and if the items are arranged in increasing order, the algorithm will place small items into the initial bins. Larger items cannot replace these. Therefore, the best performance is the average case in which the item weights (or sizes) are random [17].

2.2 Metaheuristic approaches

2.2.1 The fitness-dependent optimizer (FDO)

The FDO was developed in 2019 by Abdullah and Ahmed [18]. It is a swarm intelligent algorithm that models the characteristics of the reproductive process of bee swarms, along with their collective decision-making behaviour. Although the FDO is considered a PSO-based algorithm, it calculates the velocity of a particle differently – it uses the problem fitness function value to produce weights. These weights then guide the algorithm's search agents during the exploitation and exploration phases. Additionally, the FDO algorithm requires fewer computations than the PSO algorithm for updating the velocity and particle positions [19]. Bees are social insects who reside in small caves or hollow trees. They work in colonies that are generally called hives. The three types of bees in a colony include the queen, worker, and scout bees. Each of these bees has specific functions according to their traits. The FDO algorithm focuses on the function of the scout bees. These bees explore their environment in order to locate a suitable place for the colony to build a hive (in other words, they exploit preferable hives). Once this is found, the bees perform a sort of "dance" to communicate with the swarm [20]. In the algorithm, each hive that is exploited by a scout bee is seen as a candidate solution, and the best hive represents a global optimum solution, according to the relative fitness weight.

The advantages of this algorithm are that it is stable in both the exploitation and exploration phases and that it requires fewer computations than other algorithms. The disadvantage is that it uses a weight factor in order to control the fitness weights which are, in most cases, ignored [21]. Abdul-Minaam et al. [19] took into account these strengths and weaknesses. They adapted the FDO to solve the 1D BPP by replacing the original way of generating the first population with a random generation of the initial population using an improved FF heuristic and by updating the operating strategies in the original algorithm to improve the exploration and exploitation phases.

2.2.2 Cuckoo search via Lévy flights

In 2009, Yang and Deb developed the population-based Cuckoo Search via Lévy Flights (CS) metaheuristic algorithm [22]. This algorithm was inspired by the breeding strategy of certain cuckoo species, as they tend to lay their eggs in nests belonging to birds of other species 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 throwing it away, or they may abandon the entire nest and

rebuild elsewhere. Some cuckoos have evolved such that the female parasite cuckoos can mimic the colours and patterns of the eggs of the host birds, thereby reducing the probability of their eggs being abandoned. This technique leads to an increase in reproductivity [23]. 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 eggs by blindly propelling these eggs out of the nest. This action leads to an increased portion of food provided by the host bird to the cuckoo chick. The Lévy Flights mechanism replaces the simple random walk to improve the performance of the algorithm [22]. This mechanism flight is described as a "random walk in which the step-lengths are calculated with a heavy-tailed probability distribution" [23].

The advantages of this algorithm include that it uses far fewer parameters than most metaheuristics; therefore, not much tuning is required, it is straightforward to implement, and it is easily hybridized with other algorithms. A significant disadvantage is that fast convergence cannot be guaranteed because it is dependent on random walks [24]. Zendaoui and Layeb [25] used these strengths to aid their attempt to adapt the CS algorithm to solve the 1D BPP by introducing an interesting mechanism, Ranked Order Value (ROV), in order to convert the results of the original algorithm from a continuous space to a discrete space.

2.2.3 Whale optimization algorithm (WOA)

The WOA was developed in 2016 by Mirjalili and Lewis [26]. This algorithm is a swarm intelligence-based metaheuristic, and it imitates the hunting strategy of humpback whales. These whales are known for their ability to use an uncommon hunting strategy, also called the "bubble net" strategy. This is where the whales cleverly cooperate to trap their prey, fish, into a ring of air. The whales accomplish this by emitting a stream of bubbles. These bubbles take two different shapes: the first is a spiral, and the second is a shrinking circle. After trapping their prey in these bubbles, the fish are swallowed by the humpback whales. In the algorithm, the exploration phase is represented by the hunt for prey, while the hunting strategy utilizing the shrinking and spiral shapes represents the exploitation phase. The positions of the prey are randomly initialized – this represents the initial population, which is evaluated to find the current best solution.

The advantages of this algorithm include the simplicity of implementation and low computational cost. On the other hand, the disadvantage is that the algorithm sometimes results in stagnation in local optima, and that the exploitation phase would be more beneficial if it were improved [27]. Abdel-Basset et al. [28] utilized these strengths in their adaptation of WOA to solve the 1D BPP, called the Improved Lévy-based Whale Optimization Algorithm (ILWOA), developed to solve the BPP. This was accomplished by integrating various mechanisms including the Largest Order Value (LOV) technique (in order to convert the continuous results to discrete ones) Lévy Flights, and Chaotic maps to overcome the weaknesses of the original WOA.

2.2.4 Squirrel search algorithm

The Squirrel search nature-inspired optimization algorithm (SSA) was developed in 2018 by Jain et al. [29]. This algorithm mimics the behaviour of flying squirrels and their efficient movement. The flying squirrels do not fly, but rather they use a special technique known as gliding. This technique is considered highly efficient because it is energetically cheap and enables small animals to cover great distances quickly [30]. There are three primary assumptions in the SSA [29]. First, there is a certain number of flying squirrels in the forest. Second, each of these squirrels forages for food and utilizes the available food resources dynamically. Finally, there are three types of trees in the aforementioned forest – normal, acorn, and hickory trees.

An advantage of this algorithm is that there is an efficient search space exploration. The disadvantages, on the other hand, is that it suffers from premature convergence and that it is possible, when solving complex problems, that the algorithm will get trapped in a local optimum [31]. El-Ashmawi and Abd Elminaam [32] modified the SSA to solve the 1D BPP with these weaknesses in mind, aiming to improve on them. The modified algorithm incorporated randomization in the initial population and tweaks to the operating strategies to enhance the new generations during exploration.

2.2.5 GA

The GA was initially proposed by Holland [33] and has been applied to a vast range of optimization problems. This algorithm mimics the natural selection process of evolution, based on the teachings of Charles Darwin [34]. In GA, solutions are modelled as chromosomes, which can take many different forms – for example, binary strings or real number vectors. A population consists of multiple chromosomes. A fitness function must be defined so these chromosomes can be decoded and evaluated, and a fitness value can be assigned to this chromosome. This value is an indication of the strength of that individual in the current population and dictates the probability of being selected for the creation of the next population. The newer populations are generated by performing techniques on the chromosomes such as mutation and crossover. Stronger, or more fit, chromosomes are more likely to be chosen to be carried into the new population after the genetic operators have been applied, so that the new populations contain better quality solutions.

The advantages of this algorithm include that it has been developed to deal well with complex problems and parallelism, and it is easy to understand and implement. However, naturally, there are disadvantages to this metaheuristic – this includes the difficulty in formulating the correct fitness function for the given problem, determining the best population size and parameters such as the crossover and mutation rate, which governs how often the operators must occur during the iterations of the algorithm. An inappropriate selection of these parameters and functions may cause the GA to produce results that do not have meaning, or it may be increasingly difficult for the GA to converge [34]. Quiroz et al. [35] used the GA as a foundation for their grouping GA, adding grouping mechanisms and a fascinating selection mechanism to solve the BPP.

3 Optimization methods

3.1 Adapting metaheuristics for the 1D BPP

Figure 1 illustrates a general overview of metaheuristic algorithms that are used to solve the 1D BPP. The algorithm begins by defining parameters, such as population size and any other values that must be specified to ensure the algorithm functions as intended. Next, the initial population is generated. This step is important, and the method chosen to do this impacts the efficiency of the algorithm greatly. Next, the solutions go through the "fit and evaluate" phase. This refers to using approximate algorithms (discussed in Section 2) to pack the items into bins. This packing is then evaluated by the objective function so that the solution is assigned a fitness value that will determine it's position in the population. This phase additionally helps to determine the best solution in the population. This best solution is then stored in memory. The algorithm will then perform the specific local and global walks so that the population is improved by adding solutions with more favourable fitness values and generally discarding of unfavourable or weak solutions. The "fit and evaluate" phase must be repeated at this stage so that the best solution can be checked against, and if a better solution than the current best is found, then this solution replaces the best solution in memory and becomes the current best. Some algorithms may require an update to parameters, but this is not always the case; therefore, this is optional. Thereafter, the termination criterion is checked. Usually, termination criterion refers to the number of iterations (generations of population) the algorithm has gone through. However, the criterion may also be to check if the current best solution is an optimal solution. It is possible that either one or both of these conditions are checked at this point in the algorithm. If the criterion has been met, then the algorithm may end and produce the final solution. However, if the criterion has not been met, the algorithm returns to update the population using exploration and exploitation mechanisms, so that a better solution may be found, and the cycle continues.

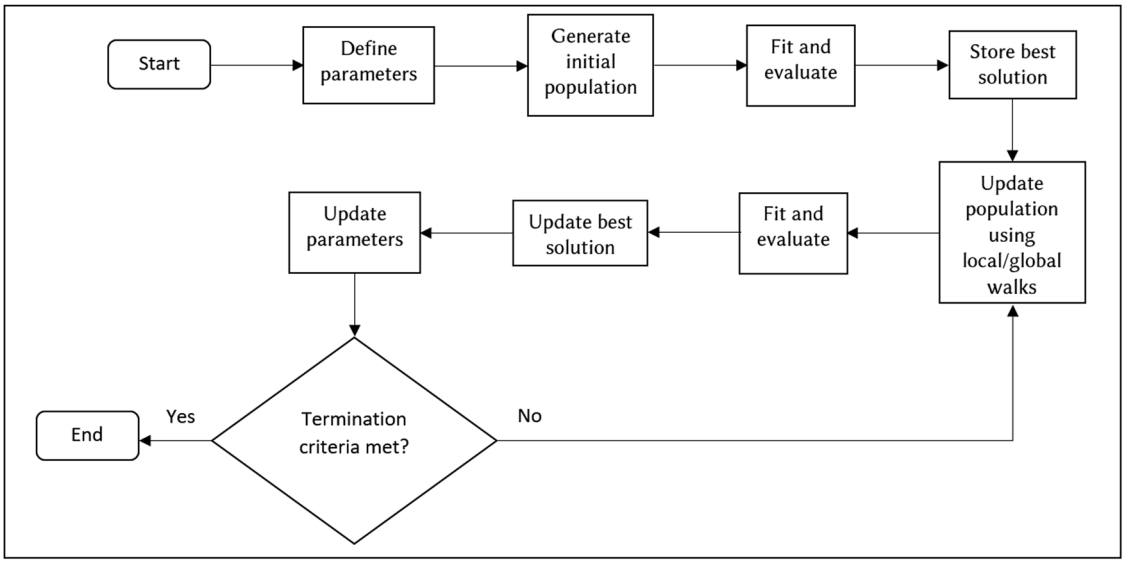

Figure 1: A generalized metaheuristic algorithms flowchart for the 1D BPP.

3.2 Adapted metaheuristics for the 1D BPP

All equations have been taken directly from the surveyed papers, and all figures and algorithms have been adapted from these papers.

3.2.1 Objective function

If this function is set to be just the number of bins, the algorithm may eventually experience stagnation [19]. This is due to several possible solutions having the same number of bins, yet different representations, and therefore, various amounts of unused space. In order to avoid such stagnation and move towards finding the optimal solutions, the objective function should consider the amount of unused space, as well as the number of bins for a specific representation. Hence, the following functions are used:

$$\text{Minimize } f(X) = 1 - \frac{\sum_{k=1}^{N}(\text{fill}_k/C)^z}{N}$$

or

$$\text{Maximize } f(X) = \frac{\sum_{k=1}^{N}(\text{fill}_k/C)^z}{N},$$

where

$$\text{fill}_k = \sum_{i \in X_k} S_i,$$

which essentially entails giving more value to the solution using the least bins and having the least amount of unused space. So, fill_k is the capacity of bin k (it is the sum of the sizes (S_i) of all items in the bin), N represents the number of bins in the solution, C represents the fixed bin capacity, and z represents a constant that is used to define the equilibrium of the filled bin (generally kept at $z = 2$) [19].

3.3 Adaptive fitness-dependent optimizer (AFDO)

In 2020, Abdul-Minaam et al. [19] adapted the FDO algorithm to solve discrete optimization problems and tested it on the BPP, and this was named AFDO. This algorithm aims to improve the solution quality for the BPP. For the adaptation, the authors suggested using a random initial population and added an update to the algorithm procedure. In this algorithm [19], an artificial bee scout represents a candidate solution to the BPP. Each solution is a vector of n dimensions where one dimension represents the index of an item, and another represents the index of the bin that the corresponding item is assigned to. Abdul-Minaam et al. [19] found this representation to aid in minimizing the number of bins used while allowing multiple items to be packed in one bin without exceeding the maximum bin capacity constraint. This can be seen in Figure 2. Figure 2(a) represents the list of items with their indexes and corresponding weights.

Since the initial population influences the execution time and quality of the result [19], the authors modified the FF heuristic such that the algorithm takes in a shuffled list of items and randomly assigns the items to the first available bin. This allows for the generation of diverse, feasible solutions in the initial population. Figure 2(b) and (c) represents possible solutions using the modified FF heuristic.

Item	1	2	3	4	5	6	7	8	9	10
Weight	6	1	8	2	4	7	1	3	9	5

(a)

Item index	1	5	6	8	9	7	10	4	2	3
Bin	1	1	2	2	3	3	4	4	4	5

(b)

Item index	10	5	2	3	4	1	8	7	6	9
Bin	1	1	1	2	2	3	3	3	4	5

(c)

Figure 2: An example of solutions in the population illustrated in figure (a)–(c).

An additional variable to be considered in this algorithm is the weight factor, wf, that is either set to 0 (neglect the factor) or 1 (lowers the possibility of coverage and increases the level of convergence). This weight controls the stability of the search and dramatically influences the movement of the bees towards the next position. Therefore, it is a parameter that requires testing. The fitness weight is directly impacted by the weight factor, as shown in equation (1), where $f(X^*_{i,t})$ represents the global best solution's fitness value and $f(X_{i,t})$ represents the current solution's fitness value.

$$fw = \left|\frac{f(X^*_{i,t})}{f(X_{i,t})}\right| \times wf. \tag{1}$$

The new position of a scout bee relies on the movement rate, pace, and direction. Only a positive direction is considered in the AFDO algorithm. The pace is computed by considering the current solution ($X_{i,t}$), a random number, r, generated with uniform distribution (rate), and the fitness weight (the last two are critical parameters that improve the exploration of the search space), as can be shown in equation (2):

$$\text{pace} = \begin{cases} X_{i,t} \otimes r & \text{if } fw \in \{0, 1\} \text{ or } (X_{i,t}) = 0 \\ (X_{i,t} \ominus X^*_{i,t}) \otimes fw & \text{if } 0 < fw < 1. \end{cases} \tag{2}$$

The cases in this equation represent the influences on the movement direction. In the first case, a random sequence of raw vectors is generated from the current solution, based on the number of bins and items in the specific problem instance. As can be shown in Figure 3, the final solution of the first case is raw vectors, of the form ⟨**item**, **old bin**, **new bin**⟩. The random number, r, is set to 0.2 and represents the probability that a task (assignment of an item to a bin) is selected for an update in the aforementioned vector. If a task is

selected, a new bin is randomly chosen for the item to be assigned to. This modification to the FDO algorithm provides the possibility of obtaining multiple solutions through the use of randomization. The number of items, I, is 10.

Therefore, $0.2 \times 10 = 2$, so two raw vectors are produced.

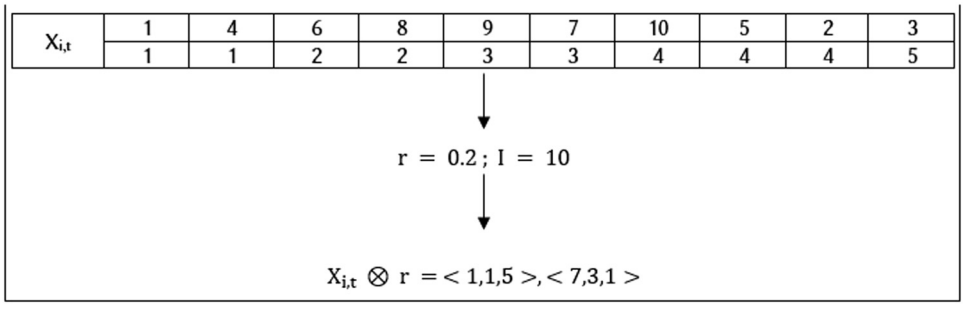

Figure 3: An example of using $X_{i,t} \otimes r$.

In the second case, the result of the $\ominus$ operator is computed by taking the difference between the current solution and the global best solution thus far. This is given in terms of raw vectors, as shown in Figure 4. The form of the vectors is the same as mentioned previously. If an item is assigned to the same bin in both solutions, then no change is made. If an item is in two different bins in the solutions, then the raw vector is added to the set. The $\otimes$ operator represents the chance of *fw* choosing each generated raw vector. Since $fw = 0.3$, three of the raw vectors are randomly chosen in the final output.

After that, the solution's pace is calculated as in equation (3), and the new solution is introduced, as shown in Figure 5.

$$X_{i,t+1} = X_{i,t} \oplus \text{pace}. \tag{3}$$

It is worth noting that the reassignment of an item to a bin will only occur if the suggested reassignment does not cause the bin to overflow (in other words, exceed capacity). This is the reason that the first and third raw vectors in the pace in Figure 5 do not affect the resulting solution.

The algorithm continues until a predefined number of iterations is reached or the most optimal solution is found.

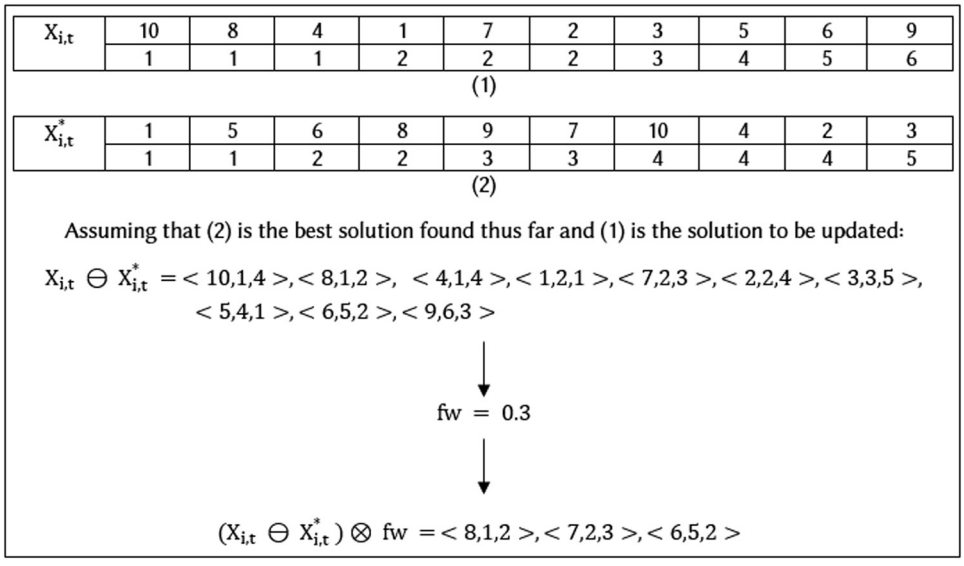

Figure 4: An example of using $(X_{i,t} \ominus X^*_{i,t}) \otimes fw$.

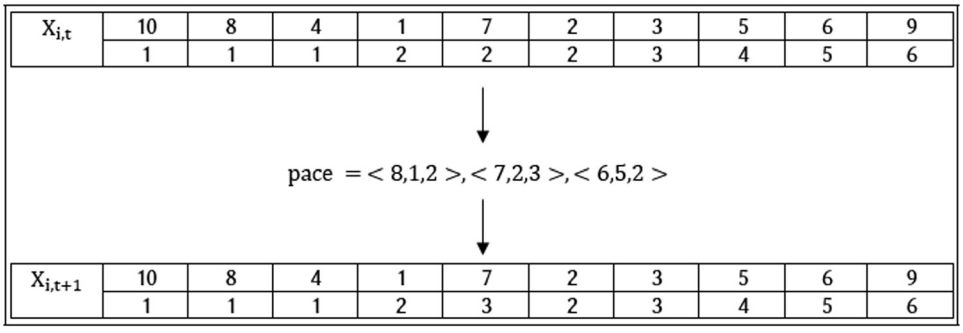

Figure 5: An example of updating a solution with pace.

The pseudocode for this algorithm can be seen in Algorithm 1.

Algorithm 1: Pseudocode of adaptive fitness dependent optimizer

initialize: number of items (I), bin capacity (C), number of iterations (tmax), population size (pmax);
generate initial population by using a modified FF heuristic:
For p = 1 to p = pmax:
Shuffle the list of items;
while *list is not empty* **do**
Select an item from the list;
Assign the item to the first available bin;
If the item cannot fit in any available bin, open a new bin;
Update the remaining capacity of the bin that the item was added to
(remove the size of the item from the bin's current capacity);
end
improve solutions
while $t \leq tmax$ **do**
(**evaluate the artificial bee scout**);
For each bee scout, $X_{i,t}$ in the population,
{
Evaluate according to objective function;
$X^*_{i,t}$ is set the bee scout with the lowest fitness value;
} Set b^* to the current number of bins in the best solution. ;
(**movement of the artificial bee scout**);
For each bee scout, $X_{i,t}$ in the population,
{
Compute fw;
if ($fw == 1$ *or* $fw == 0$ *or* $f(X_{i,t}) = 0$) **then**
set r
$pace = X_{i,t} \otimes r$;
$X_{i,t+1} = X_{i,t} \oplus pace$;
else
$pace = (X_{i,t} \ominus X^*_{i,t}) \otimes fw$;
$X_{i,t+1} = X_{i,t} \oplus pace$
end
}
$t = t + 1$;
If b^* is the optimal number of bins, stop.
end

3.4 Adaptive cuckoo search (ACS) algorithm

The CS algorithm follows three cardinal rules: First, each cuckoo must lay one egg at a time and randomly chose a nest to dump this egg in. Second, the nests that contain eggs of the highest quality will proceed to the next generation. Third, there is a fixed number of available hosts, and the host bird discovers the cuckoo egg with a probability, p_a, between 0 and 1 [34]. Additionally, this probability governs the balance between the local and global explorative walks.

In 2016, Zendaoui and Layeb [25] adapted the cuckoo search algorithm to solve the 1D BPP problem (ACS). Since the original CS algorithm was designed to solve continuous optimization problems, it could not directly be applied to solve the discrete BPP. Therefore, the authors proposed utilizing a method that linked the continuous output of the algorithm to integer representations that correspond to the index of the items in a given list. The ROV method was employed for this reason – it guarantees the feasibility of new solutions and avoids creating any additional overhead. This can be shown in Figure 6.

Continuous Solution	3.26	7.11	4.62	6.15	8.00	1.78	5.40	2.91
ROV	3	7	4	6	8	1	5	2

Figure 6: An example of employing ROV on a continuous solution.

For the representation of solutions in ACS [25], the authors assume that only one egg is laid in one nest, and each nest contains a single egg. The egg in this nest is a candidate solution represented by an item permutation, determined by the ROV. The FF method was used to assign the items to the bins and evaluate the fitness of a solution. Two essential movements are used to generate new solutions in the ACS. Firstly, a local random move which can be seen in equation (4). This move is used for exploitation and produces a new solution by utilizing two randomly selected solutions (x_j^t and x_k^t), a Heaviside function, a random number, ε, drawn from uniform distribution, and s which represents the step size. An example is provided in Figure 7.

$$x_i^{t+1} = x_i^t + \alpha s \otimes H(p_a - \varepsilon) \otimes (x_j^t - x_k^t). \tag{4}$$

Second, a global random move via Lévy Flights is used for exploration. This is shown in equations (5) and (6). A scaling factor, α, related to the scale of the problem to be solved must be set. Once the result of the Lévy Flight is calculated, it is added to the continuous version of the item permutation, and the solution is ranked once again with ROV, and then evaluated by the objective function. An example is provided in Figure 8.

Current Solution	3	7	4	6	8	1	5	2
Result of local move	2.32	7.55	4.32	6.74	8.16	1.75	3.19	5.42
New Solution	2	7	4	6	8	1	3	5

Figure 7: An example of employing the local walk on a current solution.

Current Solution	3	7	4	6	8	1	5	2
Result of global move	1.32	8.10	7.32	4.11	6.43	5.91	3.17	5.88
New Solution	1	8	7	3	6	5	2	4

Figure 8: An example of employing the global walk on a current solution.

$$x_i^{t+1} = x_i^t + \alpha L(s, \lambda), \tag{5}$$

where

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) \sin(\pi\lambda/2)}{\pi} \frac{1}{s^{1+\lambda}}, (s \gg s_0 > 0). \tag{6}$$

A fraction, equal to p_a (the switching parameter that is predefined), of nests with unacceptable fitness values are abandoned. In other words, these solutions are replaced with new solutions generated using the local walk movement. The algorithm tracks the best solution and only replaces this position if a solution with a lower fitness value is found (since the aim is minimization).

This algorithm continues until the number of iterations has reached a predefined number or if an optimal solution is found.

The pseudocode for this algorithm can be seen in Algorithm 2.

Algorithm 2: Pseudocode of adaptive cuckoo search algorithm

initialize: number of host nests (n), maximum number of iterations (t), probability of host bird discovering a nest (p_a);
Generate the initial population of n host nests, x_i where $i = 1, \ldots, n$;
while $t < tmax$ **do**
  X_i = Generate a new solution by using Lévy Flights;
  P_i = Use ROV to convert X_i into an item permutation;
  Replace X_i by P_i;
  F_i = Evaluate P_i;
  P_j = Choose random solution from population;
  if ($F_i > F_j$) **then**
   | Replace P_j with P_i;
  end
  A fraction of worse solutions are abandoned andnew solutions are generated using local walk (use $P_a \times n$;)
  Convert these new solutions to item permutations with ROV;
  Replace the new solutions by the item permutations;
  Evaluate the fitness of the new solutions;
  Rank the solutions according to fitness value to find the current best solution and store current best;
end

3.5 ILWOA

Although the convergence rate and performance of the original WOA [26] were good and obtained a high-quality solution in a negligible time when applied to a continuous search space, Abdel-Basset et al. [28] observed that WOA also performed well on combinatorial problems. This observation motivated the attempt to adapt this algorithm to solve the BPP. Therefore, in 2018, the ILWOA was developed. Three enhancements were made in this algorithm. The deployment of Lévy flights for whale movements improved the exploration capabilities of WOA, the original algorithm was embedded with a mutation phase in order to enhance convergence speed, and a logistic chaotic map was utilized to swap between exploitation and exploration phases efficiently.

Since it was necessary to discretize the search space, the authors used the LOV technique. As shown in Figure 9, this technique first ranks the descending order of continuous values, $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$, where 1 to n represents the dimensions of the solution. This results in a sequence, $\varphi_i = [\varphi_{i,1}, \varphi_{i,2}, \ldots, \varphi_{i,n}]$. To decode the solution, the formula used is $j_{i,\varphi_{i,k}=k}$, where k represents the corresponding dimension [36].

Dimension (k)	1	2	3	4	5
$x_{i,k}$	0.32	2.81	1.45	1.62	2.00
$\varphi_{i,k}$	5	1	4	3	2
$j_{i,k}$	2	5	4	3	1

Figure 9: An example of LOV.

The BF heuristic was employed for packing bins with the assigned items. The representation used in this study was simply a list of items. The Lévy distribution has an infinite mean and infinite variance. This causes a longer movement from the solution's current position to the new position, which is much more efficient for the exploration phase.

For improved diversification of the given search space, the WOA [26] combines a couple of exploration search mechanisms. This is carried through and slightly modified in the ILWOA. The probability of these search mechanisms being employed on a candidate solution depends on a switching parameter.

Chaotic maps can be described as: "evolutionary functions that produce a deterministic bounded sequence of random numbers depending on initial condition" [28]. The authors decided to use logistic chaotic maps to determine the switching parameter value due to the random-like parameter adaptation [28]; this value is known as p. The value of a coefficient vector from the original WOA, $\vec{C}$, is replaced by a random step that is drawn from Lévy distribution (equation (6)). A random variable, r, and p dictate how the solutions are updated. $\vec{A}$ is calculated as in equation (7). This coefficient vector includes $\vec{a}$ which represents the shrinking feature of the spiral and decreases from 2 to 0, and $\vec{r}$ which represents a random number between 0 and 1.

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a}. \tag{7}$$

The use of two exploration mechanisms introduces improved diversification of the search space. Therefore, the value of $\vec{A}$ influences whether the best solution or a randomly selected solution will affect the positions of solutions in the search space. This is explained below.

If $r < p$ and $\vec{A} < 1$, then the update is calculated using equation (8) and the solution is generated using equation (9). In these equations, the current best solution and the current solution are represented by $\vec{x}^*(t)$ and $\vec{x}(t)$, respectively.

$$\vec{D} = |\vec{C} \cdot \vec{x}^*(t) - \vec{x}(t)|, \tag{8}$$

$$\vec{x}(t+1) = \vec{x}^*(t) - \vec{A} \cdot \vec{D}. \tag{9}$$

However, if $r < p$ and $\vec{A} \geq 1$, then the update is calculated by equation (10), where $\vec{x}_r(t)$ is a random solution chosen from the current population, and the updated solution is generated by equation (11).

$$\vec{D} = |\vec{C} \cdot \vec{x}_r(t) - \vec{x}(t)|, \tag{10}$$

$$\vec{x}(t+1) = \vec{x}_r(t) - \vec{A} \cdot \vec{D}. \tag{11}$$

If $r \geq p$, then the update is calculated by equation (12) and the position is updated spirally by equation (13). These equations simulate the whales' movements around the prey, specifically, the best solution found thus far. In equation (13), b is a pre-specified constant that defines the shape of the spiral and l is a randomly selected number in the range $[-1, 1]$.

$$\vec{D'} = |\vec{x}^*(t) - \vec{x}(t)|, \tag{12}$$

$$\vec{x}(t+1) = \vec{D'} \cdot e^{bl} \cdot \cos(2\pi l) + x^*(t). \tag{13}$$

The additional mutation phase occurs before and after each iteration. In this phase, the current best number of bins is tested for optimality. If the test is successful, the search stops. However, if the test fails, the solution is randomly modified by three operators, namely, swap, displacement, and reversion. The swap operator randomly chooses and swaps two indexes in the solution. The displacement operator randomly chooses a subset of items and inserts it into another place in the solution. Finally, the reversion operator randomly selects a subset of the solution and reverses the order of the items within that subset. These mechanisms further improve the exploration capabilities of the algorithm. Figure 10 illustrates these operators. This occurs until the maximum number of iterations has been reached.

The pseudocode for this algorithm can be seen in Algorithm 3.

Algorithm 3: Pseudocode of improved Lévy-based whale optimization algorithm

Generate the initial population;
Use LOV to map continuous population to combinatorial population;
Evaluate population;
Set x^* to the current best solution;
Set $perm^*$ to the current best permutation;
Set $nbin^*$ to the current best number of bins;
while termination criteria has not been met **do**
  For each whale position
  { Update a, A, l, r, C with Lévy distribution, and p with logistic map;
  if $r < p$ **then**
    if $|A| < 1$ **then**
      Calculate $\vec{D} = |\vec{C} \cdot \vec{x^*}(t) - \vec{x}(t)|$;
      Update position with: $\vec{x}(t+1) = \vec{x^*}(t) - \vec{A} \cdot \vec{D}$;
    else
      Randomly select a position, $\vec{x_r}(t)$;
      Calculate $\vec{D} = |\vec{C} \cdot \vec{x_r}(t) - \vec{x}(t)|$;
      Update position with : $\vec{x}(t+1) = \vec{x_r}(t) - \vec{A} \cdot \vec{D}$;
    end
  else
    Calculate $\vec{D'} = |\vec{x^*}(t) - \vec{x}(t)|$;
    Update position with : $\vec{x}(t+1) = \vec{D'} \cdot e^{bl} \cdot \cos(2\pi l) + x^*(t)$;
  end
  Adjust position inside search space boundaries;
  Use LOV to map the new solution;
  Evaluate new combinatorial solution, $perm^n$;
  Update x^*;
  if $nbin^n < nbin^*$ **then**
  | Update $perm^*, nbin^*$
  else
  | Apply mutation for $perm^n$;
  end
  Evaluate new permutations;
  Update $perm^*, nbin^*$;
end
return $perm^*, nbin^*$

Swap	1	3	2	5	4
	1	4	2	5	3
Displacement	1	3	2	5	4
	2	5	4	1	3
Reversion	1	3	2	5	4
	1	5	2	3	4

Figure 10: Illustration of mutation operators.

3.6 Modified squirrel search algorithm

In 2019, El-Ashmawi and Abd Elminaam [32] modified the SSA algorithm in order to apply it to the BPP. This version of the algorithm generates random but feasible initial solutions. Additionally, various operating strategies are employed to update the locations of the squirrels. This aids in generating new populations during the exploration phase of the algorithm. The feasible initialization step ensures that the initial population's randomness explores the solution space in an efficient manner. This was accomplished by improving the BF heuristic – randomly assigning items from the given list of items to the BF bin (such that total waste is minimized).

The evaluation of the population depends on the values of the fitness function, as is with most metaheuristics. The population must be sorted according to the ascending order of fitness value. The best squirrel (the one with the lowest fitness value) is assigned to the hickory tree. Being on this tree indicates that this particular squirrel is at the optimal food source. In other words, it is a current best solution. The next three squirrels with relatively low fitness values are assigned to the acorn tree. This implies that those squirrels are at the normal food source. In other words, these solutions are near-current-best. Finally, the rest of the squirrels are assigned to the normal tree. This means that these squirrels have no food source – implying that these solutions are quite far from the current best [32].

Individual solutions are updated according to the tree that they are assigned to. If a solution is on an acorn tree, it moves towards the hickory tree, and if it is on a normal tree, it can be moved towards either the acorn or the hickory tree. The probabilities of these updates are controlled by a random number ($r1$, $r2$, $r3$) drawn from a uniform distribution, between 0 and 1, and a constant, Pdp. This constant is known as the "probability of a predator" and it is set beforehand. If the random number is not greater than the specified constant, the squirrel's position is randomly updated – this procedure achieves good exploration of the search space. These updates are shown in equations (14), (15), and (16). The α, β, and γ values are random numbers between 0 and 1. These values denote the gliding distance of each squirrel.

The $\ominus$ operator calculates the difference between the two solutions and results in a set of swap operators – difference means to look at each item–bin entry in the solutions and, if the two solutions have the same item in the same bin, no swap operator is created, but if the two solutions have the same item in different bins, a swap operator of the form $\langle$**item number**, **bin number**$\rangle$ is added to the set. An example of this can be seen in Figure 4.

The $\otimes$ operator computes the probability that the swap operators in the set are selected and applied in the update (the probability is given by α, β, and γ). Based on the result (a number) of this operation, random solutions are selected from the set. This can also be seen in Figure 4; however, the example uses fw in place of the random values in this algorithm.

The $\oplus$ operator updates the solution. In other words, it changes the location of the squirrel. This is done by replacing the item–bin pairs in the chosen solution with the selected swap operators. An example of this is shown in Figure 5; however, instead of a "set of swap operators," "pace" is used. These functions are essentially the same.

$$(\mathrm{FS}_{i,j}^{t+1})_{\mathrm{at}} = \begin{cases} (\mathrm{FS}_{i,j}^{t})_{\mathrm{at}} \oplus \alpha \otimes ((\mathrm{FS}_{i,j}^{t})_{\mathrm{ht}} \ominus (\mathrm{FS}_{i,j}^{t})_{\mathrm{at}}) & \text{if } r1 \geq Pdp \\ \text{random location} & \text{otherwise} \end{cases}, \tag{14}$$

$$(\mathrm{FS}_{i,j}^{t+1})_{\mathrm{nt}} = \begin{cases} (\mathrm{FS}_{i,j}^{t})_{\mathrm{nt}} \oplus \beta \otimes ((\mathrm{FS}_{i,j}^{t})_{\mathrm{at}} \ominus (\mathrm{FS}_{i,j}^{t})_{\mathrm{nt}}) & \text{if } r2 \geq Pdp \\ \text{random location} & \text{otherwise} \end{cases}, \tag{15}$$

$$(\mathrm{FS}_{i,j}^{t+1})_{\mathrm{nt}} = \begin{cases} (\mathrm{FS}_{i,j}^{t})_{\mathrm{nt}} \oplus \gamma \otimes ((\mathrm{FS}_{i,j}^{t})_{\mathrm{ht}} \ominus (\mathrm{FS}_{i,j}^{t})_{\mathrm{nt}}) & \text{if } r3 \geq Pdp \\ \text{random location} & \text{otherwise} \end{cases}. \tag{16}$$

If the random number ($r1$, $r2$, $r3$) is less than Pdp, this indicates that a squirrel was not able to reach a source of food, but survives regardless and moves to a location that is randomly generated. This is done by generating swap operators randomly and applying the $\oplus$ operator to the solution. This is shown in equation (17). O_i represents the swap operators, and $1 \le i \le m$ where m is the number of swap operators generated. These random walks introduce improved exploration in the space.

$$(\mathrm{FS}_{i,j}^{t+1}) = (\mathrm{FS}_{i,j}^{t}) \oplus \mathrm{rand}\,(O_i). \tag{17}$$

The stopping criterion is based on the number of iterations reached compared to a pre-specified maximum number or until the optimal value has been found.

The pseudocode for this algorithm can be seen in Algorithm 4.

Algorithm 4: Pseudocode of modified squirrel search algorithm

Initialize the population, FS with n individuals and the dimension of the problem, m;
Set tmax to the maximum number of iterations;
Compute $m*$ according to the equation: $m* = \left[\left(\sum_{i=1}^{n} S_i\right)/C\right]$ where S_i is the size of each item and C is the fixed bin capacity;
For $i = 1, \ldots, n$ do
  For $j = 1, \ldots, m$ do
    Generate $FS_{i,j}^{t}$ using the improved BF heuristic;
while $t \le tmax$ **do**
  Evaluate the fitness, $f(FS^t)$ according to objective function;
  Sort the locations of $f(FS^t)$ in ascending order of fitness value;
  Set FS_{ht}^{t} to the best global solution ($f(FS_{ht}^{t}) = min(f(FS^t))$) - this squirrel is located on the hickory tree;
  Set FS_{at}^{t} to the next three best global solutions - these squirrel are located on the acorn tree;
  Set FS_{nt}^{t} to the remaining solutions - these squirrels are located on the normal tree;
  Set $FS_{nt}^{t}*$ to a random subset from FS_{nt}^{t} that moves towards the hickory tree;
  Set $nbest$ to the current best solution′s number of bins;
  For $k = 1, \ldots, sizeOf(FS_{at}^{t})$ do
  if $r1 \ge Pdp$ **then**
    Determine $(FS_{i,j}^{t+1})_{at}$ with equation (14);
  else
    Random location using equation (17)
  end
  For $k = 1, \ldots, sizeOf(FS_{nt}^{t})$ do
  if $r2 \ge Pdp$ **then**
    Determine $(FS_{i,j}^{t+1})_{nt}$ with equation (15);
  else
    Random location using equation (17)
  end
  For $k = 1, \ldots, sizeOf(FS_{nt}^{t}*)$ do
  if $r3 \ge Pdp$ **then**
    Determine $(FS_{i,j}^{t+1})_{nt}$ with equation (16);
  else
    Random location using equation (17)
  end
  Set $t = t + 1$;
  If($nbest \le m*$) then stop;
end
Return the best solution

3.7 Grouping genetic algorithm with controlled gene transmission (GGA-CGT)

In 2015, Quiroz et al. [35] published the GGA-CGT to solve the 1D BPP. The authors propose transmitting the best genes in a chromosome while maintaining the balance between diversity and selective pressure in the current population.

The algorithm consists of the following features: a novel technique for generating a high-quality initial population, grouping operators that aid in exploiting the search space of the problem, a rearrangement operator that aids in exploring the search space, and the balance of the evolution is ensured by the use of a novel reproduction technique that is capable of preventing premature convergence and governing the exploration phase.

The initial population is generated with a modified FF heuristic. Until a predefined number of iterations is reached or the optimal solution is found, the individuals in the population are recombined and mutated in a couple of phases. First, some individuals are selected with "controlled selection," and the gene-level crossover and FF decreasing heuristic are employed on the selected individuals. Afterwards, "controlled replacement" is utilized to introduce the offspring. The second phase considers a predefined "lifespan" of individuals – according to this lifespan, elite individuals with superior fitness values are cloned, and several individuals are chosen to apply genetic alterations to. These alterations are accomplished with the use of "adaptive mutation" and "RP" (Rearrangement of Pairs). As in the first phase, controlled replacement is applied to individuals in the second phase, before they are added to the new population. Algorithm 5 shows the pseudocode for this algorithm.

Algorithm 5: Pseudocode of grouping genetic algorithm with controlled gene transmission

Generate initial population, P, with proposed modified FF heuristic (FF-$\tilde{n}$);
while *generation* < *max-generation and solution is not optimal* **do**
  Select n_c individuals to crossover using controlled selection;
  Apply *gene-level crossover and first fit decreasing heuristic* to the n_c selected individuals;
  Apply *controlled replacement* to introduce offspring;
  Select n_m individuals and clone elite solutions using *controlled selection*;
  Apply *adaptive mutation and rearrangement by pairs* to the best n_m individuals;
  Apply *controlled replacement* to introduce clones;
  Update the global best solution;
end

GGA-CGT uses a group-based encoding scheme – so, there could be various lengths of chromosomes based on the number of bins used in the specific solution. Therefore, every gene in a chromosome is a "group" of items. Each group is packed into a single bin.

The authors introduced a rather interesting method of generating an initial population from a set N of n items by first finding a subset, $\tilde{N}$, of $\tilde{n}$ items that have weights greater than half of the fixed bin capacity. This subset represents items that cannot be packed into a bin with items of the same subset, as this would cause a bin to overflow. The items of $\tilde{N}$ are packed first into different bins, and then the remaining items of N (that are not part of $\tilde{N}$) are randomly packed into the bins using the FF heuristic.

The grouping crossover operator called the gene-level crossover can be described as follows: A novel crossover for group-based representation was introduced, whereby the two selected parent solutions are compared on a gene-level. Two children are produced, as both of the selected solutions are given the opportunity to be the "first father." First, the operator looks at the bins in each solution according to the descending order of how full each bin is. This is done so that the offspring have a better chance of inheriting

the best bins in the parent solutions. To create the offspring, for every pair of bins, the bin that is more filled is inherited first. The bins are compared in parallel – so, the first bin of the first father and the first bin of the second father are considered a pair. After that, the other bin in the pair is inherited. This continues until all the bins in both parent solutions have been added. In the event of a pair of bins being equally filled (have the same amount of available space), the first father's bin is inherited first. It can be observed that this method would result in a duplication of items since every bin from both solutions is added to the child solution. Therefore, if a bin contains an item that already exists in the solution (i.e., the item can be found in a previous bin), then the bin containing the duplicate item is eliminated from the child solution. This elimination may cause the child solution to be invalid since it would not contain all the items from the problem instance. The method resolves this issue by determining the missing items and placing these items into the existing bins by using the FF decreasing heuristic. This can be seen in Figure 11.

The grouping mutator operator permits introducing random changes into the population, resulting in more diversity in the solutions, so there is more exploration of the search space. The proposed mutator operator promotes transmitting the "best genes" in a chromosome and is applied at the gene level to avoid the loss of high-quality genes (which are essentially well-filled bins). This operator eliminates the bins that have the most available space (are not well-filled) and reinserts the items from the eliminated bins into the solution with the help of a rearrangement technique that is discussed later. The operation is called adaptive because the number of bins that will be eliminated in a solution, n_b, is determined by equation (18) considering: the solution size (m) and how many bins in the solution and not completely filled (τ), the elimination proportion (ε) as calculated in equation (19), and the probability of elimination (P_ε) as calculated in equation (20).

$$n_b = \lceil \tau \cdot \varepsilon \cdot P_\varepsilon \rceil, \tag{18}$$

$$\varepsilon = \frac{2 - (\tau/m)}{\tau^{1/k}}, \tag{19}$$

$$P_\varepsilon = 1 - \text{Uniform}\left(0, \frac{1}{\tau^{1/k}}\right), \tag{20}$$

where k defines the rate of change of the elimination proportion, ε, and probability, P_ε, concerning the number of incomplete bins, τ.

The rearrangement heuristic, called "rearrangement by pairs," or RP, is a technique that must be implemented in two steps. First, every bin is reviewed to determine the possibility of improving its packing schema by swapping pairs of already-packed and free items (free items are also known as "missing items" which are items that need to be placed in a bin but are not present in the solution being repaired). Second, the remaining items are inserted into the solution through the FF heuristic. This technique has a significant impact on the exploration and exploitation of the search space. Algorithm 7 explains the rearrangement by pairs procedure. S' denotes the result of swapping the items s and p from the bin Bj with an item i from F. $\text{Feasible}(S')$ is true if the swap that has occurred results in a bin with a capacity that does not exceed the fixed maximum capacity, otherwise it is false. Both the items from S and F are considered in pairs. There are two possibilities for the two pairs of items, as shown in the IF-statements in the algorithm. These statements say two things: (1) if it can be done, replace the pair of items p and s from the bin with one of the free items from the pair that have a weight greater than or equal to the sum of the weight of p and s, as long as the replacement does not cause the bin to overflow; (2) if replacing the pair of items from the bin (p and s) with the free items (i and k) results in the free items filling the bin equal to or better than the items from the bin, swap the items p and s with i and k. Then, apply the FF heuristic to reinsert the items in F to make the solution valid.

Figure 12 shows an example of the adaptive mutator with the rearrangement heuristic.

Item	0	1	2	3	4	5	6	7	8
Weight	6	3	7	8	5	2	2	5	2

(a)

For a bin capacity of 10, (b) and (c) are possible solutions.

Fullness {	10	9	8	8	5
Items {	3, 5	8, 2	0, 6	1, 7	4
	A	B	C	D	E

(b)

Fullness {	10	8	8	7	7
Items {	1, 2	3	0, 8	4, 6	7, 5
	a	b	c	d	e

(c)

Each gene is compared in parallel → bin **A** is compared with bin **a**, bin **B** is compared with bin **b**, and so forth.
The bin that is inherited first is the bin that is more filled, immediately afterwards, the other bin in the pair is inherited. For instance: bin **B** is fuller than bin **b**, so bin **B** is inherited first and then bin **b** follows. An example of what occurs when the bins in a pair have the same capacity can be seen in the first pair, bin **A** and bin **a**. Bin **A** belongs to the first father, and so is given preference and is inherited first. The result of this ordering can be seen in (d).

10	10	9	8	8	8	8	7	7	5
3, 5	1, 2	8, 2	3	0, 6	0, 8	1, 7	4, 6	7, 5	4
A	a	B	b	C	c	D	d	e	E

(d)

Then eliminate bins with duplicate items. For instance, bin **B** has item 2, but this item has already been seen in bin **a**. Also, bin **c** has items 0 and 8 which has already been seen in bins **C** and **B**, respectively. After elimination, determine the missing, or free, items.

10	10	8	5
3, 5	1, 2	0, 6	4
A	a	C	E

(e)

Now, we can see that items 7 and 8 are missing. The authors suggest using the first fit decreasing heuristic to reinsert these free items to complete the child solution, as seen in (f).

10	10	10	10
3, 5	1, 2	0, 6, 8	4, 7
A	a	C	E

(f)

Figure 11: An example of the gene-level crossover operator. (a) An example of the list of items in a BPP instance. (b) An example of a first father. (c) An example of a second father. (d) The result of the offspring from the first and second father in the examples. (e) Partial solution after removing duplicates. (f) The completed child solution. Source: [35].

Algorithm 6: Pseudocode of RP (Rearrangement by Pairs)

RP(S, F) – S is the solution to be rearranged and F is the set of free items
Randomly sort the bins of S to obtain a sequence of b bins: $S = (B_1, \ldots, B_b)$;
Randomly sort the items of F;
$j \leftarrow 1$;
while $j \leq b$ **do**
  for all $(p, s) \in B_j$ **do**
    for all $(i, k) \in F$ **do**
      if $w_i \geq w_p + w_s$ *and Feasible*$(S' \leftarrow$ *Swap* $(S, F, (p, s), i))$ **then**
      $S \leftarrow S'$;$j \leftarrow j + 1$; go back to while statement;
  end
      if $w_k \geq w_p + w_s$ *and Feasible*$(S' \leftarrow$ *Swap* $(S, F, (p, s), k))$ **then**
      $S \leftarrow S'$;$j \leftarrow j + 1$; go back to while statement;
  end
      if $w_i + w_k \geq w_p + w_s$ *and Feasible*$(S' \leftarrow$ *Swap* $(S, F, (p, s), (i, k)))$ **then**
      $S \leftarrow S'$;$j \leftarrow j + 1$; go back to while statement;
  end
end
Apply FF to reinsert the items in F to S;

The authors additionally propose a "controlled selection" mechanism that balances the population diversity and selective pressure. Population diversity permits exploring new areas of the search space, which positively impacts solution quality and helps avoid premature convergence. The selective pressure awards fitter individuals in the population with a better chance of being chosen for mutation, reproduction, and survival. If these attributes are missing from the algorithm, the search process will not be guided but instead be random, which will result in promising regions of the search space being overlooked. Therefore, balancing these features greatly improves the GGA-CGT's performance. Every individual is given the opportunity to contribute to the succeeding generation of the population; however, the proposed selection scheme ensures that the best individuals survive.

The controlled selection for crossover is as follows: for n_c parents to generate n_c children, create two sets of individuals, G and R, that will serve as parents. B is known as the elite group, which consists of the best solutions. The first set, G, which stands for "good," contains $n_c/2$ solutions randomly chosen from the best n_c solutions in the current population. In contrast, the second set, R, which stands for "random," also contains $n_c/2$ solutions randomly selected but from the set: population minus B. The random selections are made with uniform probability. Moreover, because this crossover occurs pairwise between the two sets (hence, G_i is crossed with R_i, where $0 \leq i < n_c/2$), the manner in which the sets are chosen avoids a solution being crossed with itself.

The controlled selection for mutation is as follows: a fixed number of solutions, n_m, will be mutated. The operator is applied to n_m of the best solutions in the population. In this case, lifespan plays a role in determining if cloning occurs. If a chosen solution exists within a specific lifespan, then it is cloned. After this, solutions are selected from the population to mutate, according to decreasing fitness values.

The replacement strategy dictates which solutions are eliminated and replaced by the new progeny. The authors aim to promote exploration by always permitting new solutions to be added. There are two types of controlled replacement: one for crossover and one for mutation. In the controlled replacement for crossover, the first $n_c/2$ children generated from the crossover operator replace their parent from the random set, R (their second father). The remaining children replace solutions from a subset of the population that does not include the set of best (B) and random (R) solutions. In this latter case, the children either replace

solutions from the described subset of the population with duplicate fitness (if they exist) or worst solutions. In the controlled replacement for mutation, the cloned solutions either replace solutions with duplicate fitness (if they exist), or the worst solution in the current population.

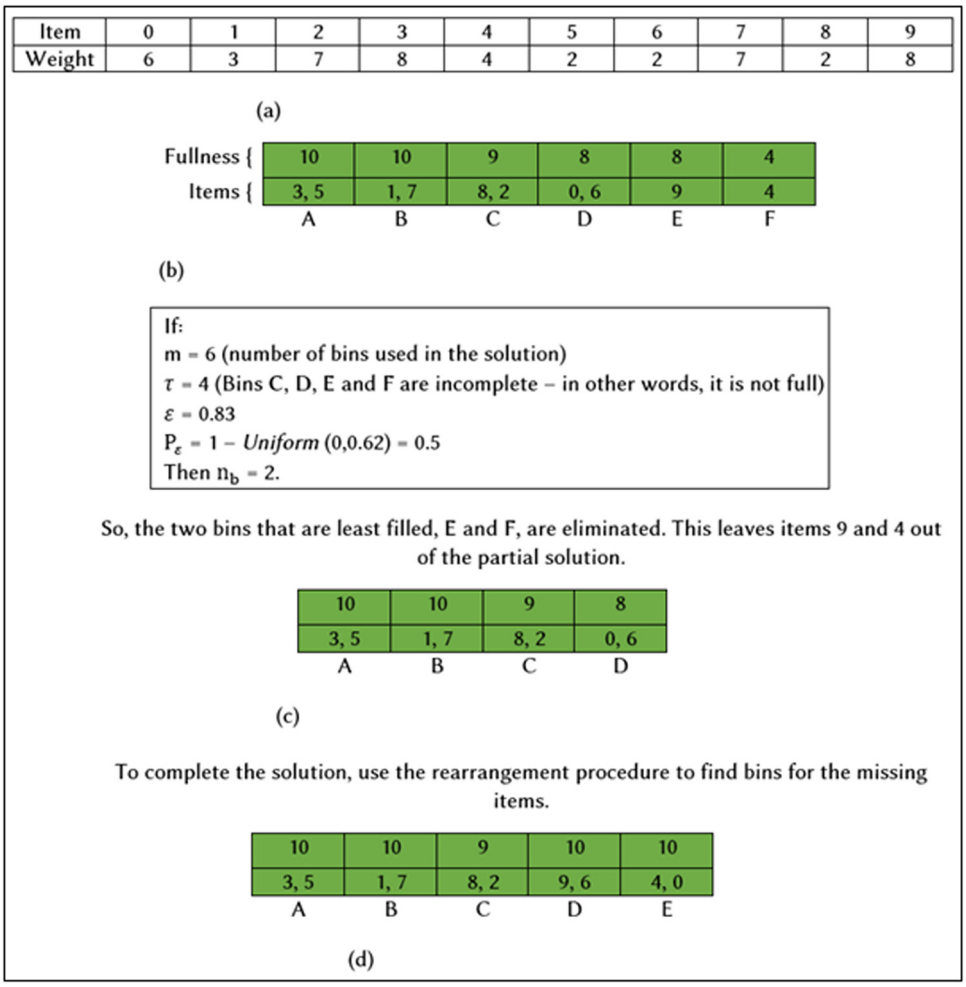

Figure 12: An example of the adaptive mutator with RP operator. (a) An example of the list of items in a BPP instance. (b) An example of an individual in the population with bin capacity of 10. (c) Partial solution after elimination of two bins. (d) The completed child solution. Source: [35].

4 Comparison of results

Since every algorithm described in this article used different instances of datasets, and were tested on hardware with varying hardware specifications, it would not be possible to conduct a valid comparison of the metaheuristics with each other. However, the researchers have provided results and compared their proposed algorithms with various other popular algorithms; therefore, this will be analysed in this section.

The benchmark datasets used in all of the reviewed papers were made available by Scholl et al. [37] and are divided into easy, medium, and hard classes. These classes have various numbers of items, weights, and maximum bin capacities. For the easy class, there are 720 instances, and the number of items in each instance can either be 50, 100, 200, or 500. The bin capacities are 100, 120, or 150. In the medium class, there are 480 instances, and the number of items in each instance is the same as the easy class; however, the maximum bin capacity for each instance is 1,000. Finally, for the hard class, there are ten instances, and there are 200 items in each instance, and the capacity is 100,000. These sets have been used to test several

algorithms, and it provides a way to compare algorithms based on their success in solving these instances. For the following results, each researcher selected particular instances of these classes to demonstrate the successes of their algorithms. However, these instances do not necessarily overlap between papers.

The AFDO [19] algorithm was tested on three classes of diverse difficulty, and the results were compared to three other well-known metaheuristics – namely, the PSO, Jaya, and Crow Search (CS) algorithms. In the easy and medium classes, the AFDO was able to display high efficiency in solving the 1D BPP in terms of the average fitness values. Out of 15 instances of the easy class, the AFDO successfully found all optimal values, whereas the other three algorithms found between 7 and 9 optimal values. Similarly, for the medium class, out of ten instances, all ten optimal values were found by the AFDO, whereas the other three algorithms found five or six optimal values. However, in the most challenging class (hard), the algorithm did not reach the optimal number of bins, but it was noted that a better item packing schema was achieved, yielding a lower number of bins compared to the other algorithms. Hence, AFDO proved to effectively explore the search space to locate an optimal or near-optimal solution within a reasonable time.

The ACS algorithm [25] was tested on the described dataset. The results were compared to the FF Decreasing heuristic, the Ant System algorithm, Firefly and Ant Colony algorithm, and the Quantum Cuckoo Search algorithm. For the easy class, the ACS produced results identical to the other four algorithms in the chosen instances – these were all optimal values. For the medium class, ACS outperformed other algorithms in all chosen instances – either achieving optimal or near-optimal results. Finally, for the hard class, the ACS successfully achieved better near-optimal results and outperformed the other four algorithms. Moreover, the ACS algorithm has fewer parameters to tune, and as a result, it is easier to implement.

The ILWOA [29] was tested against the ACS algorithm, quantum-inspired cuckoo search, firefly algorithm, firefly colony optimization, and ant system algorithm. For the selected instances in the easy class, the ILWOA performs just like other algorithms and finds the best-known number of bins. For the instances of the medium and hard classes, however, the results obtained using ILWOA were not always optimal but were superior in many cases. The authors also noted that the overall results of the ACS and ILWOA on this standard benchmark dataset were very similar. However, ILWOA was able to return reasonable solutions with fewer iterations and search agents.

The modified squirrel BPP (MSBPP) algorithm [32] was tested against popular metaheuristic algorithms such as PSO, African Buffalo Optimization, and Crow Search Algorithm (CRSA) to solve the BPP. The authors chose specific instances of the benchmark datasets from the easy, medium, and hard classes. For all of these easy class instances, the MSBPP algorithm was able to reach optimal values, whereas PSO and CRSA failed to reach optimal values for even half of the selected tests. For the medium class, the MSBPP successfully reached optimal values for 11 of the 12 instances, whereas PSO performed exceptionally poor, reaching optimal values for merely 3 of the 12 instances. For the instances of the hard class, the MSBPP performed well relative to other algorithms, achieving values much closer to the optimal number of bins.

For the GGA-CGT [35], the authors compared their algorithm to two state-of-the-art algorithms, namely, Perturbation-SAWMBS (PSAWMBS) and the hybrid improvement heuristic for the 1D BPP (HI_BP). The authors of GGA-CGT [35] tested their algorithm on a range of BPP datasets but did not report in detail their findings on the standard benchmark dataset described above. However, the report stated that optimal results for all 720 instances of the easy class, 480 instances of the medium class, and 10 instances of the hard class of the standard benchmark dataset described above were found, just as for the other two algorithms. It was worth noting that the PSAWMBS was a much faster algorithm and obtained similar results in the majority of the datasets that were tested. The most thought-provoking findings of this report were that for the Hard28 dataset, GGA-CGT was able to find optimal results for 16 of the 28 instances, while the other two algorithms were only able to find optimal results for 5 instances. This highlights the superiority of GGA-CGT as the Hard28 dataset is known to be one of the most challenging datasets known for the 1D BPP [35].

Finally, it is very important to highlight here that because the choice of parameter settings can significantly affect the quality of solutions generated by the individual metaheuristic optimization algorithms, several experiments need to be performed in order to find the best combination of parameters values that

would give the desired competitive results. Therefore, depending on the scale of the test problems for which the algorithms are being evaluated upon, it is suggested that different empirical and statistical analysis options should be explored and adopted to ensure the efficient selection of the parameter values. However, this approach might be more appropriate for those class of metaheuristics with varying control parameters. As for those variant of metaheuristics with little or no control parameter, such initial evaluation would not be necessary. For example, metaheuristic optimization algorithms such as the symbiotic organisms search algorithm [38,39] and teaching-learning-based optimization [40] are known to have few or no control parameters beside the initialized population size, which is a default control parameter setting for all the global metaheuristic algorithms.

5 Complexity analysis of metaheuristics

There are three challenging areas in algorithm analysis: complexity, convergence, and no-free-lunch theory [41]. For the traditional algorithms, complexity analysis is well-established because the algorithms are deterministic. However, for metaheuristic algorithms, complexity analysis remains a challenging task to date, partly due to their nondeterministic or stochastic nature. In addition, no generic framework exists for them. Although, in recent times, dynamic systems and Markov processes have been used to study complexity analysis of metaheuristics algorithms, convergence analysis still remains one of the active research areas with many encouraging results [41–43].

Furthermore, there is a common belief among domain enthusiasts that metaheuristic algorithms tend to be less complex for implementation. In many cases, the problem sizes are not directly linked with the algorithm complexity [41]. Although most metaheuristics can often solve immensely challenging NP-hard combinatorial optimization problems, our understanding of their performance in terms of efficiency and convergence lacks far behind. Moreover, because of the algorithms, diverse application and capability of solving complex problems like the BPP, metaheuristics can be considered as an efficient way to produce acceptable solutions in a reasonably practical time. In addition, because metaheuristics are random search-based techniques, the complexity of the problem of interest makes it impossible to search for every possible solution or combination, specifically for large graph size BPPs. Therefore, in most cases, the aim is to find reasonable, feasible solutions in an acceptable timescale. Hence, the general pursuit and idea are to have efficient but practical algorithms that will work most of the time and produce reasonable quality solutions.

Finally, based on the notion of no-free-lunch theorem that Wolpert and Macready proposed in 1997 [44], which states that given any two algorithms A and B, the algorithms will on average perform equally, that is, if algorithm A performs better than algorithm B for some problems, then algorithm B will outperform algorithm A for other problems [41]. This implies that no single universally accepted superior algorithm can solve all types of problems, including the BPPs. However, this does not mean that some algorithms would not perform better than other algorithms for some specific kinds of problems. We do not need to measure performance on average for all functions, but instead, we need to measure how an algorithm performs for a given class of problems.

6 Recent application areas of BPP

The BPP is an interesting research domain that has over the years received recognition and more comprehensive coverage because of its relevance to industrial applications, especially in cutting (wood and glass industries), packing (transportation and warehousing), and supply chain management (vehicle, container, pallet or cargo loading, cutting stock and trim loss problems, packaging design, resource allocation, load balancing, scheduling, project management, and financial budgeting). Therefore, in this section, we present more recent and general application areas of the BPP, and we tried to classify articles that are

published in these areas into their different categories. However, interested readers are directed to refs. [45–48] for older and more theoretical research coverage on the application areas of the BPP.

6.1 Scheduling and resource allocation

The application of BPP to the area scheduling and resource allocation basically deals with the problem of allocating limited resources or tasks with/without time consideration. Researchers have proposed different approaches or variation of the BPP in this regard. For example, Ojeyinka [49] applied the online and offline variants of the bin packing heuristics, implemented using the fill function, on the passenger-bus scheduling and the multiprocessor job scheduling problems. The goal of reducing the timespan needed for a multiprocessor to process all tasks using the BPP is exploited by Coffman et al. [50]. The performance of some scheduling algorithm was evaluated in ref. [51] using the BPP where the operating room time corresponds to the bin to be minimized. Leinberger et al. [52] showed how multi-capacity bin packing can lead to a highly sophisticated multi-resource allocation and scheduling. Furthermore, the scheduling of surgical operations using the BPP was exploited by Van Houdenhoven et al. [53].

6.2 Production

The BPP has several practical applications in the supply chain sectors, which cut across different manufacturing and service orientations. For instance, in manufacturing, the product to be manufactured can be considered to be a task to be accomplished within a time frame in a production cycle; the plant used for production is the bin to be allocated to the task (product), and finally, the goal of minimizing production cost is achieved by minimizing the number of bins [54]. Smart packing algorithm was designed by Khairuddin et al. [55] using the GA to solve the 3D bin or container packing problem. The authors in ref. [55] then applied their proposed method to reduce packaging waste due to ineffective packaging in the manufacturing process. Two-dimensional packing algorithm was similarly used in ref. [56] to solve the problem of composite material production in autoclave moulding. The aspect of product delivery using mobile robotics was exploited in ref. [57].

6.3 Health care community

The BPP is applied in this area to promote better cooperation between different components within the health care sector to provide an improved health care system. Surgical cases and resource scheduling using BPP were exploited in refs. [53,58,59]. The common goal here is to make optimal use of the operating rooms by scheduling surgeries that can be combined to achieve high throughput in the system. Patient scheduling for access to a practitioner was exploited in ref. [60], they used BPP for effective and optimal scheduling of patients to available practitioners. In the real sense, this problem is essentially a BPP with the modification that each bin might have a different total size. Each doctor forms a bin whose capacity is equal to the number of free hours in the doctor's schedule, and each patient is an object to be placed into a bin, whose size is equal to the number of hours for which she needs to be seen. The overall goal here is to find the most efficient way to schedule patients across doctors, while upholding certain measure of fairness in terms of workload across the doctors, and maximizing fairness while getting every patient seen by the prescribed doctors on duty. However, this task would become even more complex and difficult to solve when dealing with a scenario where the doctors are specialists in different subfields and each patient needs to be seen by a doctor with a certain set of skills under the different subfields depending on their medical practice.

Another interesting application of the BPP is in nurse scheduling, here a consideration of the capacity of the nurses and the various needs of the patients using BPP are taken into account. A nurse to patient allocation relative to the BPP was considered in ref. [61].

6.4 Distribution and inventory

Although some aspect of the distribution and inventory system can be classified as being part of the production category, distribution and inventory are well researched using the BPP. The BPP fits in here because of consideration of sizes and shape, technological and orientation constraints during delivery and so on. Gupta and Radovanovic [62] considered a scenario of items in the form of a sequence (for delivery or inventory) that must be packed on arrival. A similar scenario was considered in ref. [63] except that in this case, additional constraints of no buffering and readjustment were added. For an inventory, online order fulfilment is an integral part of e-commerce. Zhang et al. [64] proposed a solution to the dynamic stocking problem using a mixed integer programming. Besides, without adequate inventory in a warehouse, storage processing is interrupted. Ravichandran and Sasi [65] proposed a robust solution approach using the BPP-based farm optimization algorithm.

6.5 Logistics

The logistic process, in this case, implies packaging and routing of goods and services for different interests. This industry usually deals with the respective company rules on handling their products; however, this is by no means an easy task. Sajini Anand and Guericke [66] proposed a variable size BPP to achieve this objective. A smart packaging scheme was proposed by Khairuddin et al. [55]. A scenario where a decision must be made from a list of permissible actions upon the arrival of a stochastic request was exploited by Banerjee and Freund [67]. The authors in ref. [68] used BPP to address the problem of minimization of the distance of client's location from the requested item location. The electric cars used are in multi-depots.

6.6 Cloud computing

In cloud computing, the problem of energy efficiency and cost of operation can be effectively managed if the number of servers used for a given task is minimized. This approach greatly fits into BPP, hence, several studies have been published in this regard. The placements of the virtual machines were considered to be the bins in ref. [69]. A heuristic method was proposed to solve a smaller version of the problem, with the primary goal of minimizing the problem objective function. A BPP model that implements the reversed auction in cloud computing was proposed by Ye et al. [70]. In the same vein, Srirama et al. [71] proposed a dynamic BPP strategy with auto scaling. In order to solve the complications of scheduling and autoscaling caused by container layer, Wang et al. [72] proposed an elastic scheduling for microservices (ESMS), which minimizes the cost of the virtual machines without compromising autoscaling.

7 Concluding remark and future direction

This study has reviewed current nature-inspired, population-based metaheuristics that have been used to solve the classical 1D BPP. These algorithms were either adaptations or modifications of other

metaheuristics. The mechanisms discussed show how simple it is to discretize a solution space to adapt algorithms that were created for continuous optimization problems to algorithms that solve discrete optimization problems. Furthermore, the researchers of these reviewed papers focused on improving the stability of the exploration and exploitation phases of the metaheuristics, which greatly impacts the final solution quality and improved on the weaknesses of the original metaheuristics while leveraging the strengths. As can be seen, some algorithms are being tailored towards simplicity of implementation, such as the ACS and ILWOA, while others focus on decreasing computational expense. The former approach usually results in optimal solutions at the expense of time and resources, while the latter approach generally produces near-optimal solutions but takes far less time to compute a result.

In general, metaheuristics are widely used optimization tools for finding near-optimal solutions to large graph sized problem instances of the BPP in reasonable execution times. Moreover, the metaheuristic optimization algorithms are often simple to implement and flexible in terms of their adaptability in solving the 1D BPP in practice. In addition to these benefits, this class of optimization algorithms continue to be promising tools for numerous NP-Hard optimization problems where the exact solution methods tend to fail because of the exponential search spaces. However, some disadvantages of the metaheuristic algorithms include the issue of premature convergence, which can lead the algorithms into getting trapped in local optimum, and the aspect of parameter fine-tuning, as some of the algorithms would require having to set the control parameter to meet certain specified threshold.

For future research, it would be interesting to review hybridized versions of nature-inspired metaheuristic algorithms that have been used to solve the 1D BPP and other variants which involve additional constraints. Furthermore, it will be worth exploring the specific real-world applications of the metaheuristic algorithms for solving the BPP and the impact of these implementations. More so, a comparative performance analysis study of the state-of-the-art implementation of different nature-inspired metaheuristic algorithms can be considered so as to identify the best possible methods for handling the different variants of the BPP. Finally, it would be interesting to further conduct an extensive comparative analysis study to determine the success of the new generation metaheuristic algorithms presented in ref. [73] on the 1D BPP.

Conflict of interest: The authors declare that they have no financial nor personal interests that could have an influence on the work reported in this article.

References

[1] Valério de Carvalho JM. Lp models for bin packing and cutting stock problems. Eur J Oper Res. 2002;141:253–73.
[2] Coffman E, Leung J, Csirik J. Variants of classical one-dimensional bin packing. Handbook of approximation algorithms and metaheuristics. Taylor & Francis; 2007 May. p. 33.
[3] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness (series of books in the mathematical sciences). 1st ed., W. H. Freeman, editor. New York: W. H. Freeman and Company; 1979.
[4] Gass SI, Harris CM. Encyclopedia of operations research and management science. J Oper Res Soc. 1997;48(7):759–60.
[5] Coffman EG, Leung JY, Ting DW. Bin packing: Maximizing the number of pieces packed. Acta Inf. 1978 Sep;9(3):263–71.
[6] Krause KL, Shen VY, Schwetman HD. Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems. J ACM (JACM). 1975;22(4):522–50.
[7] Coffman Jr. EG, Garey MR, Johnson DS. Dynamic bin packing. SIAM J Comput. 1983;12(2):227–58.
[8] Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem. Oper Res. 2000;48(2):256–67.
[9] Martello S, Pisinger D, Toth P. New trends in exact algorithms for the 0–1 knapsack problem. Eur J Oper Res. 2000;123(2):325–32.
[10] Gendreau M, Potvin J-Y. Metaheuristics in combinatorial optimization. Annals OR. 2005 Nov;140:189–213.
[11] Luo F, Scherson ID, Fuentes J. A novel genetic algorithm for bin packing problem in jmetal. In 2017 IEEE International Conference on Cognitive Computing (ICCC); 2017. p. 17–23.
[12] Tlili T, Krichen S. On solving the double loading problem using a modified particle swarm optimization. Theor Comput Sci. 2015;598:118–28.
[13] Tadei R, Crainic TG, Perboli G. Ts2pack: A two-level tabu search for the three-dimensional bin packing problem. 2009 Jun;195:744–60.

[14] Ezugwu AE, Shukla AK, Nath R, Akinyelu AA, Agushaka JO, Chiroma H, et al. Metaheuristics: a comprehensive overview and classification along with bibliometric analysis. Artif Intell Rev. 2021. doi: 10.1007/s10462-020-09952-0.
[15] Sgall J. Online bin packing: Old algorithms and new results. In Conference on Computability in Europe. Cham: Springer; 2014. p. 362–72.
[16] Johnson DS, Demers AJ, Ullman JD, Garey MR, Graham RL. Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM J Comput. 1974;3(4):299–325.
[17] Bhatia AK, Hazra M, Basu SK. Better-fit heuristic for 1D BPP. In 2009 IEEE International Advance Computing Conference; 2009. p. 193–6.
[18] Abdullah JM, Ahmed T. Fitness dependent optimizer: Inspired by the bee swarming reproductive process. IEEE Access. 2019;7:43473–86.
[19] Abdul-Minaam DS, Al-Mutairi WMES, Awad MA, El-Ashmawi WH. An adaptive fitness-dependent optimizer for the one-dimensional bin packing problem. IEEE Access. 2020;8:97959–74.
[20] Schultz KM, Passino KM, Seeley TD. The mechanism of flight guidance in honeybee swarms: subtle guides or streaker bees?. J Exp Biol. 2008;211(20):3287–95.
[21] Muhammed DA, Saeed SAM, Rashid TA. Improved fitness-dependent optimizer algorithm. IEEE Access. 2020;8:19074–88.
[22] Yang X-S, S Deb. Cuckoo search via levy flights In 2009 World congress on nature & biologically inspired computing (NaBIC). IEEE; 2009. p. 210–14.
[23] Joshi AS, Kulkarni O, Kakandikar G, Nandedkar V. Cuckoo search optimization-a review. Mat Today. 2017 Jan;4:7262–9.
[24] Shehab M, Khader AT, Al-Betar M. A survey on applications and variants of the cuckoo search algorithm. Appl Soft Comput. 2017;61:1041–59.
[25] Zendaoui Z, Layeb A. Adaptive cuckoo search algorithm for the bin packing problem. In: Chikhi S, Amine A, Chaoui A, Kholladi M, Saidouni D. editors. Modelling and implementation of complex systems. Lecture notes in networks and systems, vol. 1. Cham: Springer; 2016. doi: 10.1007/978-3-319-33410-3_8.
[26] Mirjalili S, Lewis A. The whale optimization algorithm. Adv Eng Softw. 2016;95:51–67.
[27] Mohammed HM, Umar SU, Rashid TA. A systematic and meta-analysis survey of whale optimization algorithm. Comput Intell Neurosci. 2019;2019:8718571. doi: 10.1155/2019/8718571.
[28] Abdel-Basset M, Manogaran G, Abdel-Fatah L, Mirjalili S. An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems. Pers Ubiquitous Comput. 2018;22(5):1117–32.
[29] Jain M, Singh V, Rani A. A novel nature-inspired algorithm for optimization: Squirrel search algorithm. Swarm Evol Comput. 2019;44:148–75.
[30] Vernes K. Gliding performance of the northern flying squirrel (Glaucomys sabrinus) in mature mixed forest of eastern Canada. J Mammal. 2001 Nov;82:1026–33.
[31] Zheng T, Luo W. An improved squirrel search algorithm for optimization. Complexity. 2019;2019:6291968. doi: 10.1155/2019/6291968.
[32] El-Ashmawi WH, Abd Elminaam DS. A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem. Appl Soft Comput. 2019;82:105565.
[33] Holland JH. Adaptation in natural and artificial systems. 2nd ed., Ann Arbor, MI: University of Michigan Press; 1975. p. 1992.
[34] Yang XS. Nature-inspired metaheuristic algorithms. United Kingdom: Luniver press; 2010.
[35] Quiroz M, Reyes LC, Torres-Jimenez J, Santillán C, Fraire-Huacuja H, Alvim A. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Comput Operat Res. 2014 Oct;55:52–64.
[36] Qian B, Zhou H-B, Hu R, Xiang F-H. Hybrid differential evolution optimization for no-wait flow-shop scheduling with sequence-dependent setup times and release dates. In: Huang De-S, Gan Y, Bevilacqua V, Figueroa JC, editors. Advanced intelligent computing. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 600–11.
[37] Scholl A, Klein R, Jürgens C. Bison: A fast hybrid procedure for exactly solving the 1D BPP. Comput Operat Res. 1997;24(7):627–45.
[38] Ezugwu AE, Prayogo D. Symbiotic organisms search algorithm: theory, recent advances and applications. Expert Syst Appl. 2019;119:184–209.
[39] Ezugwu AE, Adeleke OJ, Akinyelu AA, Viriri S. A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems. Neural Comput Appl. 2020;32(10):6207–51.
[40] Rao RV, Savsani VJ, Vakharia DP. Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. Computer-Aided Design. 2011;43(3):303–15.
[41] Yang XS. Metaheuristic optimization: algorithm analysis and open problems. In: Pardalos PM, Rebennack S. editors. Experimental algorithms. SEA 2011. Lecture notes in computer science, vol. 6630. Berlin, Heidelberg: Springer; 2011. doi: 10.1007/978-3-642-20662-7_2.
[42] Clerc M, Kennedy J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans Evol Comput. 2002;6(1):58–73.
[43] Henderson SG, Nelson BL. editors. Handbooks in operations research and management science: simulation. Elsevier; 2006.
[44] Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Trans Evolut Comput. 1997;1(1):67–82.

[45] Eliiyi U, Eliiyi D. Applications of bin packing models through the supply chain. Int J Bus Manag Stud. 2009 Jan;1:11–19.
[46] Han B, Diehr G, Cook J. Multiple-type, two-dimensional bin packing problems: Applications and algorithms. Ann Oper Res. 1994 Dec;50:239–61.
[47] Gabay M, Zaourar S. Vector bin packing with heterogeneous bins: application to the machine reassignment problem. Ann Oper Res. 2016;242(1):161–94.
[48] Christensen HI, Khan A, Pokutta S, Tetali P. Approximation and online algorithms for multidimensional bin packing: A survey. Comput Sci Rev. 2017;24:63–79.
[49] Ojeyinka T. Bin packing algorithms with applications to passenger bus loading and multiprocessor scheduling problems. Commun Appl Electron. 2015 Sep;2:38–44.
[50] Coffman Jr. EG, Garey MR, Johnson DS. An application of bin-packing to multiprocessor scheduling. SIAM J Comput. 1978;7(1):1–17.
[51] Dexter F, Macario A, Traub RD. Which algorithm for scheduling add-on elective cases maximizes operating room utilization?: use of bin packing algorithms and fuzzy constraints in operating room management. Anesthesiol. 1999 Nov;91(5):1491–500.
[52] Leinberger W, Karypis G, Kumar V. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In Proceedings of the 1999 International Conference on Parallel Processing; 1999. p. 404–12.
[53] Van Houdenhoven M, van Oostrum JM, Hans EW, Wullink G, Kazemier G. Improving operating room efficiency by applying bin-packing and portfolio techniques to surgical case scheduling. Anesthesia & Analgesia. 2007;105(3):707–14.
[54] Angelelli E, Bianchessi N, Filippi C. Optimal interval scheduling with a resource constraint. Comput Oper Res. 2014;51:268–81.
[55] Khairuddin U, Razi N, Abidin M, Yusof R. Smart packing simulator for 3d packing problem using genetic algorithm. J Phys Conf Ser. 2020 Jan;1447:012041.
[56] Xie N, Zheng S, Wu Q. Two-dimensional packing algorithm for autoclave molding scheduling of aeronautical composite materials production. Comput Indust Eng. 2020;146:106599.
[57] Rhiat A, Aggoun A, Lachere R. Combining mobile robotics and packing for optimal deliveries. Procedia Manufacturing, The 1st International Conference on Optimization-Driven Architectural Design (OPTARCH 2019). 143 2020;44:536–42.
[58] Vijayakumar B, Parikh PJ, Scott R, Barnes A, Gallimore J. A dual bin-packing approach to scheduling surgical cases at a publicly-funded hospital. Eur J Oper Res. 2013;224(3):583–91.
[59] Laurent A, Klement N. Bin packing problem with priorities and incompatibilities using PSO: application in a health care community. IFAC-PapersOnLine. 2019 Jan;52:2596–601.
[60] Klein Kranenbarg PW. Patient scheduling optimization through an application of the cutting stock problem. Bachelor's thesis, University of Twente; 2020. p. 1–32. http://essay.utwente.nl/80499/1/KleinKranenbarg_BA_eemcs.pdf.
[61] Marzouk M, Kamoun H. Nurse to patient assignment through an analogy with the bin packing problem: Case of a tunisian hospital. J Oper Res Soc. 2020. doi: 10.1080/01605682.2020.1727300
[62] Gupta V, Radovanović A. Interior-point-based online stochastic bin packing. Oper Res. 2020;68(5):1474–92.
[63] Zhao H, She Q, Zhu C, Yang Y, Xu K. Online 3d bin packing with constrained deep reinforcement learning. arXiv preprint arXiv. 2020;2006.14978.
[64] Zhang J, Onal S, Das S. The dynamic stocking location problem – dispersing inventory in fulfillment warehouses with explosive storage. Int J Prod Econ. 2020;224:107550.
[65] Ravichandran SK, Sasi A. Effective storage of goods in a warehouse using farm optimisation algorithm. Int J Cloud Comput. 2020;9(2/3):207.
[66] Anand S, Guericke S. A bin packing problem with mixing constraints for containerizing items for logistics service providers. In: Lalla-Ruiz E, Mes M, Voß S, editors. Computational logistics. ICCL 2020. Lecture notes in computer science, vol 12433. Cham: Springer; 2020. doi: 10.1007/978-3-030-59747-4_22.
[67] Banerjee S, Freund D. Uniform loss algorithms for online stochastic decision-making with applications to bin packing. In Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 1–2
[68] Zhu X, Yan R, Huang Z, Wei W, Yang J, Kudratova S. Logistic optimization for multi depots loading capacitated electric vehicle routing problem from low carbon perspective. IEEE Access. 2020;8:31934–47.
[69] Aydın N, Muter İ, İlker Birbil Ş. Multi-objective temporal bin packing problem: An application in cloud computing. Comput Oper Res. 2020;121:104959.
[70] Ye D, Xie F, Zhang G. Truthful mechanism design for bin packing with applications on cloud computing. J Comb Optim. 2020 Jun. doi: 10.1007/s10878-020-00601-4.
[71] Srirama SN, Adhikari M, Paul S. Application deployment using containers with auto-scaling for microservices in cloud environment. J Netw Comput Appl. 2020;160:102629.
[72] Wang S, Ding Z, Jiang C. Elastic scheduling for microservice applications in clouds. IEEE Trans Parall Distrib Syst. 2021;32(1):98–115.
[73] Dokeroglu T, Sevinc E, Kucukyilmaz T, Cosar A. A survey on new generation metaheuristic algorithms. Comput Indust Eng. 2019;137:106040.