International Journal of Combinatorial Optimization Problems and Informatics, 16(4), Sep-Dec 2025, 194-211. ISSN: 2007-1558.
https://doi.org/10.61467/2007.1558.2025.v16i4.1004

www.editada.org

An Experimental Study on Grouping Mutation Operators within the GGA-CGT Applied to the One-Dimensional Bin Packing Problem

Alejandro Barojas-Vázquez[1], *Marcela Quiroz-Castellanos*[1], *Guadalupe Carmona-Arroyo*[1]
[1] Universidad Veracruzana, Instituto de Investigaciones en Inteligencia Artificial, México.
alejandrobv08@gmail.com, maquiroz@uv.mx, gcarmonaarroyo@gmail.com

Abstract. The Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT) is among the most effective algorithms for solving the one-dimensional Bin Packing Problem (1D-BPP), a classical NP-hard combinatorial optimisation problem with numerous industrial and logistical applications. This study aims to identify the characteristics that enable a mutation operator to perform better within this algorithm by implementing five state-of-the-art mutation operators: Elimination, Merge & Split, Swap, Insertion, and Item Elimination. Performance was evaluated in terms of the number of optimal solutions obtained. Our findings indicate that the GGA-CGT performs best with the least disruptive operators and that both the gene selection strategy and the item selection strategy can enhance the performance of mutation operators. These findings were validated by redesigning and improving a state-of-the-art item-oriented operator, achieving a 26% improvement over the best baseline version of the same operator.

Keywords: Bin packing problem, grouping genetic algorithm, combinatorial optimization, grouping problems, mutation operators.

Article Info
Received March 14, 2025
Accepted June 14, 2025

1 Introduction

The one-dimensional Bin Packing Problem (1D-BPP) can be defined as follows: Given an unlimited number of bins, each with a fixed capacity $c > 0$, and a set of n items, each with a weight $0 < w_i \leq c$, it intends to find the minimum number of bins necessary to pack all the items such that no bin has its capacity exceeded by the sum of the weights of the items it contains. Due to being an NP-hard problem, it is unlikely that there exists a deterministic algorithm that can solve all 1D-BPP possible cases optimally in polynomial time, given that the problem complexity grows exponentially as the size of the problem increases.

The Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT) can efficiently find the optimal solution for 1D-BPP instances with different features, prevailing as one of the best metaheuristics in the state-of-the-art (Quiroz-Castellanos et al., 2015). However, experimental studies of its performance on high-difficulty benchmark instances have shown that, for certain types of test cases, its operators can lead to premature convergence (Carmona-Arroyo et al., 2021; González-San-Martín et al., 2023; Vázquez-Aguirre et al., 2024). To analyze and improve the performance of the GGA-CGT, problem reduction strategies and diversity mechanisms have been studied and included (González-San-Martín, 2021), parameter control strategies have been implemented (Flores-Torres et al., 2023), and the crossover operator strategies have been analyzed to propose a new crossover operator (Amador-Larrea, 2022).

The performance of the GGA-CGT is mainly related to the Adaptive mutation operator, which adjusts the level of variation to the properties of each solution and promotes the transmission of the best genes, including an intelligent rearrangement heuristic to improve the quality of each gene. However, the strategies included in the operator have not been sufficiently analyzed to explain its performance; furthermore, the GGA-CGT has not been studied with other state-of-the-art mutation operators to understand the impact that changing this operator may have on its final performance.

To the best of the author's knowledge, none of the state-of-the-art grouping mutation operators have been tested on the GGA-CGT before, which serves as motivation for performing the study presented in this paper. In the present work, we implemented five state-of-the-art grouping mutation operators. We analyzed the GGA-CGT performance by replacing the original mutation operator with the state-of-the-art grouping mutation operators: Elimination, Merge & Split, Swap, Insertion, and Item Elimination (Ramos-Figueroa et al., 2021). We measured each one's performance regarding the number of optimal solutions found throughout two benchmarks. The proposed study aims to identify which characteristics of a mutation operator make it perform better than others.

The structure of the paper is as follows. Section 2 presents some of the most relevant state-of-the-art algorithms for 1D-BPP; Section 3 briefly defines the components of the GGA-CGT; in Section 4, we explain each of the five state-of-the-art mutation operators that will be implemented; Section 5 includes the experiments done to analyze the performance of the mentioned mutation operators and the obtained results; finally, Section 6 constitutes the conclusions and the future research directions.

2 Related Work

The specialized literature contains surveys and reviews of several algorithms used to solve the 1D-BPP (Ramos-Figueroa et al., 2019; Munien et al., 2020; Munien & Ezugwu, 2021; Gonzalez-San-Martin et al., 2023). This section presents several algorithms, including heuristic techniques, hybrid algorithms, evolutionary algorithms, and other bio-inspired metaheuristics.

The Hybrid Grouping Genetic Algorithm (HGGA) for 1D-BPP, presented by Falkenauer (1996), utilizes a representation scheme that handles the groups as genes, a fitness function that evaluates the fullness of the bins, variation operators that work with the group-based representation scheme, and a local optimization method inspired by the dominance criterion of Martello & Toth (1990). Fleszar & Hindi (2002) developed Perturbation-MBS', an algorithm that uses a modified MBS heuristic first presented in Gupta & Ho (1999). This algorithm builds new solutions by perturbing the current solution with a variable neighborhood search and lower bounds. Alvim et al. (2004) proposed a hybrid improvement heuristic (HI_BP). This heuristic used the search space reduction techniques of Martello & Toth (1990), lower bounding strategies, a load redistribution method based on dominance, differencing, and unbalancing, and a Tabu search as an improvement process. Levine & Ducatelle (2004) developed another hybrid algorithm; this time, it was an ant colony optimization alongside a strategy for local search that eliminates bins and tries to swap free items with already packed ones.

Singh & Gupta (2007) presented C_BP by combining a hybrid genetic algorithm with an improved version of the Perturbation-MBS' heuristic introduced by Fleszar & Hindi (2002). In another related work, a neighborhood search with the concept of weight annealing was proposed by Loh et al. (2008), where temperature controlled the amount of item distortion in the search. Fleszar & Charalambous (2011) introduced Perturbation-SAW-MBS', a modification to the Perturbation-MBS' method of Fleszar & Hindi (2002) that uses a new principle to control the average weight of the items packed in each bin called sufficient average weight (SAW) by comparing the average weight of a subset of items with the average weight of all currently unpacked items. Layeb & Chenche (2012) presented a new approach based on the GRASP procedure that worked in two phases: (1) a stochastic procedure that can use both First Fit (FF) and Best Fit (BF) heuristics to build initial solutions, and (2) a Tabu search to improve the solutions found in phase one.

In the work of Cruz-Reyes et al. (2012) a hybrid genetic algorithm called HGGA-BP was proposed, which utilized the Falkenauer group representation scheme and efficient heuristics to generate the initial population and to perform crossover and mutation operations, as well as a hybrid strategy to reinsert free items left from the variation operators into the solutions. A memory-integrated artificial bee colony algorithm was developed in Bayraktar et al. (2014) that combines a memory with a neighborhood search to force the bees to stay away, for some time, from the already studied and experienced neighborhoods that had no practical results and, allow them to revisit these neighborhoods with a random search, after a certain amount of time had passed. Dokeroglu & Cosar (2014) used parallel computation techniques with hybrid parallel algorithms, evolutionary grouping metaheuristics, and bin-oriented heuristics to obtain solutions for large-scale 1D-BPP instances.

Quiroz-Castellanos et al. (2015) proposed the Grouping Genetic Algorithm with Controlled Gene Transmission (GGA-CGT), another genetic algorithm that uses the groups representation scheme of Falkenauer (1996) with strategies to keep the best genes in the crossover and mutation operators, as well as controlled selection and replacement techniques to promote diversity with elitism. Later, Buljubašić & Vasquez (2016) presented the consistent neighborhood search (CNSBP). CNSBP uses problem reduction techniques and lower bounds to help construct the initial partial solution, a Tabu search to add or remove elements from bins, and the hill descent procedure to get close to a feasible solution. Ozcan et al. (2016) proposed the 1D-BPP-CUDA, an efficient grouping genetic algorithm using the Graphics Processing Unit (GPU) with Compute Unified Device Architecture (CUDA) that lowers computational times.

The Adaptive Cuckoo Search (ACS), presented in Zendaoui & Layeb (2016), used a decoding mechanism to obtain discrete solutions and apply an algorithm initially designed for continuous problems to the 1D-BPP. Kucukyilmaz & Kiziloz (2018) developed the Island Parallel Grouping Genetic Algorithm (IPGGA), where sequential GGA procedures run in parallel while allowing communication among separate executions to share solutions and improve the results. In the work of El-Ashmawi & Abd Elminaam (2019), a modified version of the Squirrel search algorithm (MSBPP), proposed originally in Jain et al. (2019), was introduced. The main modifications in MSBPP were generating the initial feasible solution by improving the Best Fit heuristic and applying an operator strategy to gain a final feasible solution in terms of bin efficiency by regrouping and packing items into bins. Gherboudj (2019) combined an Adaptive African Buffalo Optimization with the rank order value method to obtain discrete solutions and 1D-BPP heuristics to incorporate problem knowledge.

An Adaptive Fitness-Dependent Optimizer (AFDO) was proposed in Abdul-Minaam et al. (2020). The AFDO is based on the Fitness-Dependent Optimizer developed in Abdullah & Ahmed (2019); however, it includes a modified Fist Fit heuristic for generating the initial population and searches for the final optimal solution by adjusting parameters. The Hybrid Evolutionary Algorithm (HEA), presented in Borgulya (2021), uses two new mutation operators and 10 local searches to improve solutions with the help of a relative pair frequency matrix that contains how often a pair of items is included in the same bin in the best solutions. Ali et al. (2024) presented an Ant Colony Optimization integrated with the grouping representation scheme that Falkenauer (1996) presented and a differential pheromone procedure that rewards efficient packing, encouraging solutions with full bins. A multi-parent crossover operator, called Deep Neural Crossover (DNC), that uses deep reinforcement learning and an encoder-decoder architecture to identify linear and non-linear correlations between genes to select the most promising ones, was implemented in a GA for the 1D-BPP in Shem-Tov & Elyasaf (2024). Cellini et al. (2024) introduced the Quantum Augmented Lagrangian method for Bin Packing (QAL-BP). This method was a Quadratic Unconstrained Binary Optimization (QUBO) formulation that efficiently scales logical qubits and analyzes the estimation of the Lagrangian penalty terms by establishing a connection between QUBO models and Augmented Lagrangian methods.

Ramos-Figueroa et al. (2021) reviewed the state-of-the-art variation operators for grouping genetic algorithms, which included seven mutation operators. Of these seven operators, only five could be implemented for the 1D-BPP. In other previous works, it has been proven that incorporating problem domain knowledge into variation operators is key to developing high-performance grouping genetic algorithms (Ramos-Figueroa & Quiroz-Castellanos, 2024). Examples of this can be found in the specialized literature for different grouping problems, like the Image Segmentation Problem (ISP) (Fernández-Solano, 2023; Zavaleta-García, 2023), Parallel-Machine Scheduling Problem with Unrelated Machines and Makespan Minimization ($R||Cmax$) (Ramos-Figueroa, 2022), and more specifically, an experimental study like the one presented in this article has been done for the crossover operator for the GGA-CGT and yielded promising results (Amador-Larrea, 2022).

3 Groping Genetic Algorithm with Controlled Gene Transmission

Proposed in Quiroz-Castellanos et al. (2015), this is a genetic algorithm (GA) that uses a group-based representation scheme that adapts naturally to solve grouping problems and therefore is named a Grouping Genetic Algorithm (GGA). In this scheme, the genes of a solution represent groups or bins with their respective objects. For the fitness function, the GGA-CGT uses the function proposed in Falkenauer (1996), which evaluates the average squaring of the "bin efficiency". Eq. 1 defines the fitness function, where *m* is the number of bins in the solution, S_i is the sum of the item weights in bin *i*, and *c* is the bin capacity.

$$F_{BPP} = \frac{\sum_{i=1}^{m} \left({S_i}/{c} \right)^2}{m} \quad . \tag{1}$$

The initial population is generated with the FF-$\tilde{n}$ heuristic, proposed for the first time in Quiroz-Castellanos et al. (2015). This heuristic starts by packing the $\tilde{n}$ items that have a weight greater than 50% of the maximum capacity of the bins, in $\tilde{n}$ bins. The rest of the items are randomly permuted and inserted in the solution with the First Fit heuristic, which places the current item in the first bin that can contain it without exceeding its capacity. If no such bin exists, a new one is created.

A controlled selection scheme is used to select individuals to cross and mutate. This strategy uses an elitist approach alongside two inverted rankings to select individuals to promote diversity in the population. For the crossover, solutions are pairwise combined from two sets of parents, *G* and *R*, each with $n_c/2$ individuals. *G* consists of $n_c/2$ randomly selected solutions from the best n_c individuals of the population with uniform probability, and *R* consists of $n_c/2$ randomly selected solutions from the entire population without considering the elite solutions, also with uniform probability. The elite solutions are the population's best $|B|$ solutions; the size of B is a predefined parameter in the GGA-CGT. For the mutation process, the best n_m solutions of the populations are selected and mutated; however, the solutions belonging to the elite group $|B|$ with an age lower than a predefined parameter called *life_span* are cloned before being mutated. This parameter limits the generations for which an elite solution is considered for cloning for the sake of diversity in the population.

Regarding the crossover operator, the GGA-CGT uses gene-level crossover, which combines two parents, p_1 and p_2, and generates two children, c_1 and c_2. The first step is to sort the genes of both parents in descending order of how full each gene (bin) is. After that, in the second step, genes from both parents are compared in parallel, and the fullest bin is selected first to pass onto the child, followed by the gene of the other parent. If both genes are equally full, then a priority criterion is used: for the first child, priority is given to the gene of the first parent, and for the second child, the priority is given to the gene of the second parent. If any parent happens to have more genes than the other, then these genes are inherited directly by the children as their last genes. Lastly, genes with repeated items are eliminated from the solution, and any free items are reinserted using the First Fit Decreasing heuristic. This heuristic uses the same process as First Fit, but the items are sorted in decreasing order based on their weight beforehand.

The mutation operator included in the GGA-CGT is the Adaptive mutation operator. This operator works at the gene level, eliminating the n_b least full bins. The variable n_b gets calculated for each solution and is related to the size of the solution and the number of non-full bins. The equation used for this is shown below:

$$n_b = \left\lceil \iota \cdot \varepsilon \cdot p_\varepsilon \right\rceil , \tag{2}$$

where ι corresponds to the number of non-full bins in the solution, ε is the elimination proportion defined in Eq. 3, $p\varepsilon$ is the elimination probability defined in Eq. 4, and k is a parameter that defines the rate of change of ε and $p\varepsilon$ with respect to ι (k>0).

$$\varepsilon = \frac{2-\left({}^{\iota}\!/_{m}\right)}{\iota^{1/k}} , \tag{3}$$

$$p_\varepsilon = 1 - Uniform\left(0, \frac{1}{\iota^{1/k}}\right) . \tag{4}$$

The free items left from eliminating bins are reinserted into the solution using the Rearrangement by Pairs heuristic. This heuristic consists of two stages: (1) Swaps between pairs of packed and free items are attempted at each bin; (2) the free items left after that are packed into the solution with the FF heuristic.

The controlled replacement strategy in GGA-CGT seeks to preserve diversity and the best solutions by replacing duplicated fitness individuals first and then the worst solutions. For crossover, the first $n_c/2$ children replace the set of parents R, and then the other $n_c/2$ children replace the individuals with duplicated fitness. When no individuals with duplicated fitness are left and there are still children who have not been inserted, they replace the worst individuals. As mentioned, some solutions are cloned when selected for the mutation process. The clones are then reinserted by replacing the solutions with duplicated fitness first. When no such solutions exist and there are still clones to be reinserted, they replace the worst solutions in the population.

4 Grouping Mutation Operators

The mutation operator works by slightly altering a solution, which promotes the exploration of the search space and helps when the algorithm is approaching a local optimum by providing the capacity of redirecting the search elsewhere (Ramos-Figueroa et al., 2021). These operators can be organized into item- and group-oriented operators, considering the procedure they use to mutate a solution. The item-oriented operators change the location of x items in y selected groups, while group-oriented operators apply their operations to y selected groups. Ramos-Figueroa et al. (2021) presented a survey on the state-of-the-art grouping variation operators; from this review, only five operators were selected for implementation in this work due to the specifications of the 1D-BPP. In the following subsections, we will explain each group-oriented operator, Elimination and

Merge & Split, followed by each item-oriented operator, Swap, Insertion, and Item Elimination. Their general procedure, as well as their implementation in GGA-CGT, will be explained below.

4.1 Elimination

The procedure of the Elimination operator consists of selecting *k* groups to eliminate, releasing the items of all these groups, and then reinserting them with a problem-domain heuristic into the solution. The variable *k* is a number generated randomly each time the operator is called, and it can range in the interval [1, number of groups in the solution]. The selection of the groups to eliminate is random for each of them. In Fig. 1, an example of this operator is shown where three groups (marked with a pink color) are selected for elimination, and all the items in the groups end up as free items. These free items are then reinserted into the solution with a problem-domain heuristic.

4.2 Merge & Split

On the other hand, Merge & Split works in two phases: (1) it selects two groups (G_1 and G_2) to join them and convert them into a single group; (2) it selects a group and splits it into two different groups. Given the features of the group-based solutions in the GGA-CGT and the restriction of the bin capacity for the 1D-BPP (which does not allow combining two groups into one), for this implementation, Merge & Split works as the Elimination operator with a fixed number of groups to eliminate equal to 2. This means that this operator will always eliminate two groups. The selection of the groups is also random. An example of this operator is shown in Fig. 1, where two groups are selected for elimination (highlighted in pink) and have their items released and become free items. Finally, these free items are reinserted into the solution with any problem-domain heuristic.

Consider a 1D-BPP instance with a bin capacity equal to 10 and 9 items N = {1,...,9} with weights (3,7,2,5,8,4,6,2,3).
Given a BPP solution:

Items	5,3	2,8	6,4	9,7	1
Fullness	10	9	9	9	3

a) Elimination

Solution	Items	5,3	2,8	6,4	9,7	1
	Fullness	10	9	9	9	3

Free items
2,8,6,4,9,7

Mutation	Items	5,3	1,2	9,8,4	7,6
	Fullness	10	10	10	10

b) Merge & Split

Solution	Items	5,3	2,8	6,4	9,7	1
	Fullness	10	9	9	9	3

Free items
2,8,6,4

Mutation	Items	5,3	9,7	1,8,4	6	2
	Fullness	10	9	10	4	7

Fig. 1. An example of each of the item-oriented mutation operators: a) Elimination and b) Merge & Split for a 1D-BPP instance.

4.3 Swap

The first item-oriented operator, Swap, randomly selects two groups (G_1 and G_2), then randomly picks n_1 items from G_1 and n_2 items from G_2 and exchanges them. The values of n_1 and n_2, as mentioned in Ramos-Figueroa et al. (2021), can be defined with tuning and control techniques. For this implementation, both n_1 and n_2 have a fixed value 1. Fig. 2 depicts an example of this operator where two groups are selected first, and then one item from each group is selected, 8 and 1. Finally, the move is performed only if swapping these items between their respective groups does not exceed the bin capacity. If performing the swap would violate any bin capacity, no movement is made, and the solution remains the same.

4.4 Insertion

In the Insertion operator, a random group *G* is selected, and *n* items are removed from that group. The free items are reinserted in other groups using a problem-domain heuristic. Again, the *n* parameter can be defined using tuning and control techniques. For this implementation, the *n* parameter is generated randomly each time the operator is called and can range in the interval [1, number of items in the group *G*]. In the example shown in Fig. 2, the group selected is highlighted pink, and the item selected for elimination is highlighted in red. This item is then treated as a free item and reinserted into another group with a problem-domain heuristic.

4.4 Item Elimination

The last operator, Item Elimination, generates a probability with uniform distribution for each item in every group. If this probability is less than a threshold *t*, the item is eliminated from the solution. This means that the closer the threshold value is to 1, the more the mutated solutions tend to differ from their original. The mutated solutions tend to be similar when the threshold is closer to 0. The free items are then reinserted with a problem-domain heuristic. Fig. 2 illustrates this operator with an example. First, a probability for each item is generated, and if it is lower than the threshold, in this case 0.3, the item is eliminated. In this example, the items selected for elimination are highlighted in pink with their probability first and red when shown in the group-based scheme. Items eliminated become free items and are reinserted into the solution using a problem-domain heuristic.

Consider a 1D-BPP instance with a bin capacity equal to 10 and 9 items N = {1,...,9} with weights (3,7,2,5,8,4,6,2,3).
Given a BPP solution:

Items	5,3	2,8	6,4	9,7	1
Fullness	10	9	9	9	3

a) Swap

Solution	Items	5,3	2,8	6,4	9,7	1
	Fullness	10	9	9	9	3

Mutation	Items	5,3	2,1	6,4	9,7	8
	Fullness	10	10	9	9	2

b) Insertion

Solution	Items	5,3	2,8	6,4	9,7	1
	Fullness	10	9	9	9	3

Free items
9

Mutation	Items	5,3	2,8	6,4	7	1,9
	Fullness	10	9	9	6	6

c) Item Elimination

Items	5	3	2	8	6	4	9	7	1
Probability	0.1	0.55	0.06	0.55	0.79	0.07	0.12	0.48	0.25

t =0.3

Solution	Items	5,3	2,8	6,4	9,7	1
	Fullness	10	9	9	9	3

Free items
5,2,4,9,1

Mutation	Items	3,9,4	8,5	6,1	7	2
	Fullness	10	10	7	6	7

Fig. 2. An example of each of the item-oriented mutation operators: a) Swap, b) Insertion and c) Item Elimination for a 1D-BPP instance.

In Section 5, we present an analysis of the previous operators and the Adaptive operator (see Section 3) by experimenting with integrating the state-of-the-art operators in the GGA-CGT and studying the algorithm's behavior with each one.

5 Computational Experiments

This section presents the experiments performed to analyze the impact that the different operators (Elimination, Merge & Split, Swap, Insertion, and Item Elimination) can have on the performance of GGA-CGT. The algorithm's performance is assessed in terms of the number of optimal solutions found in test instances, and two benchmarks were used to help with this. First, it was the classic benchmark, which consists of 1615 instances proposed by:

- Scholl et al. (1997): three sets of instances with uniform item sizes. The difficulty of the set of instances increases for each one, starting with Data set 1, containing 720 easy instances, Data set 2, containing 480 medium difficulty instances, and finally, Data set 3, containing 10 hard instances.

- Falkenauer (1996): there are two sets of instances. The first one, called Uniform, has 80 easy instances with uniformly distributed item sizes, and the second set, called Triplets, has 80 more difficult instances with triplets of items that, in any optimal solution, must be packed into the same bin without leaving unused space.

- Schoenfield (2002): a set of 28 hard instances called Hard28. This set is the most complex set of instances in the classic benchmark.

- Schwerin & Wäscher (1997): two sets of 100 low difficulty instances called Was 1 and Was 2.

- Wäscher & Gau (1996): one set of 17 hard instances, known as Gau 1.

.

Table 1 summarizes the information, showing the number of instances (#), the number of items (*n*), the bin capacities (*c*), and the distribution.

Table 1. Main characteristics of the instances used in the classic benchmark

Class	Parameters of the instances			
	#	*n*	*c*	Distribution
Data set 1	720	{50, 100, 200, 500}	{100, 120, 150}	Uniform
Data set 2	480	{50, 100, 200, 500}	1000	Uniform
Data set 3	10	200	100,000	Uniform
Triplets	80	{60, 120, 249, 501}	1000	Ad-hoc
Uniform	80	{120, 250, 500, 1000}	150	Uniform
Hard28	28	{160, 180, 200}	1000	Ad-hoc
Was 1	100	100	1000	Uniform
Was 2	100	120	1000	Uniform
Gau 1	17	[57–239]	10,000	Ad-hoc

The second benchmark is the BPPv$_u$_c benchmark, proposed in Carmona-Arroyo et al. (2021), and it consists of 2,800 instances distributed among four classes (BPP.25, BPP.5, BPP.75, and BPP1). Each class is divided into seven sets; each of these consisting of 100 instances with bin capacities increasing for every set in the following way: $c = 10^2$, $c = 10^3$, $c = 10^4$, $c = 10^5$, $c = 10^6$, $c = 10^7$, and $c = 10^8$. For the BPP.25 class, the number of items *n* varies starting from 110 items to 154 items and with the optimal solutions for this class having a number of bins equal to 15; for the BPP.5 class, the number of items *n* varies starting from 124 items to 167 items and with the optimal solutions for this class having a number of bins equal to 30; for the BPP.75 class, the number of items *n* varies starting from 132 items to 165 items and with the optimal solutions for this class having a number of bins equal to 45; for the BPP1 class, the number of items *n* varies starting from 148 items to 188 items and with the optimal solutions for this class having a number of bins equal to 60. According to Carmona-Arroyo et al. (2021), the difficulty of these instances is caused by the large capacities of the bins alongside the fact that in the optimal solutions all bins end up full. The information is summarized in Table 2, in which the number of instances (#), the number of items (*n*), the bin capacities (*c*), and the distribution are shown.

Table 2. Main characteristics of the instances used in the BPPv$_u$_c benchmark

Class	Parameters of the instances			
	#	n	c	Distribution
BPP.25	700	[110, 154]	10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8	Ad-hoc
BPP.5	700	[124, 167]	10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8	Ad-hoc
BPP.75	700	[132, 165]	10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8	Ad-hoc
BPP1	700	[148, 188]	10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8	Ad-hoc

The GGA-CGT algorithm and the implementation of all the operators tested in this study were developed in the C++ language and were compiled using Borland C++ 5.02 in the Win32 mode. The experiments were performed on a computer with an AMD Ryzen 5 5600 3.50 GHz processor. A single execution of the GGA-CGT was run for each operator for each instance, with an initial seed for the random number generation set to 1. The parameters used are the same in the configuration proposed by Quiroz-Castellanos et al. (2015):

a) Population size $|P| = 100$
b) Maximal number of generations $max\_gen = 500$
c) Number of solutions to be selected for crossover $n_c = 20$
d) Number of solutions to be selected for mutation $n_m = 83$
e) Rate of change for non-cloned solutions $k = 1.3$
f) Rate of change for cloned solutions $k = 4$
g) Number of solutions in the elite set $|B| = 10$
h) Maximal age for a solution in the elite set to be cloned $life\_span = 10$

For all the operators tested, the controlled techniques and reinsertion heuristic (Controlled Selection, Controlled Replacement, and Rearrangement by Pairs) remain the same as in the GGA-CGT, and only the operator itself was modified. The Adaptive operator in the original GGA-CGT selects the bins to eliminate by fullness, meaning that the n_b least full bins are eliminated. An additional version of the Adaptive operator, called Adaptive-R for brevity, was also implemented. Adaptive-R has the same behavior as Adaptive except in the bin selection phase, where the n_b bins are selected randomly instead of by bin fullness. The bin and item selection are random for all five state-of-the-art operators and their different versions. With this, we aim to show the importance of the selection process in the mutation operation.

Tables 3 and 4 show the results of the group-oriented operators, including the Adaptive operator, for the classic and BPPv$_u$_c benchmarks, respectively. The first column represents the names of each class of instances, the second column represents the number of individual instances for each class, and the third through tenth columns represent the number of optimal solutions by each mutation operator. Each row represents a class of instances and the optimal solutions each operator found per class, and the final row represents the total number of instances in the benchmark for the second column and the total number of optimal solutions found for each operator for the rest of the columns. The case is the same regarding Tables 5 and 6, but for the item-oriented operators.

The Adaptive operator found the optimal solution of 1600 instances of the classic benchmark and of 1482 instances of the BPPv$_u$_c benchmark, for 3082 instances in total, and was the operator with the best results. Adaptive-R found the optimal solution of 1495 instances of the classic benchmark and 937 of BPPv$_u$_c, for 2432 instances in total.

Due to the Adaptive operator varying the number of bins to eliminate depending on the solution size, a similar behavior was sought after in the Elimination operator by creating different versions with different numbers of bins to eliminate to analyze the difference in performance for each of them: (1) the original version, which eliminates k random groups (k is a random variable generated each time the operator is called); (2) a version called for brevity, Elimination10, where the operator eliminates a fixed 10 percent of the bins in the solution; the bins are selected at random, (3) a version called for brevity, Elimination20, where the operator eliminates a fixed 20 percent of the bins in the solution; the bins are selected at random, (4) a version called for brevity, Elimination30, where the operator eliminates a fixed 30 percent of the bins in the solution, the bins are selected at random, and (5) a version called for brevity, Elimination40, where the operator eliminates a fixed 40 percent of the bins in the solution, the bins are selected at random. We can observe that Elimination10 is the version of Elimination that performs the best in both the classic benchmark and in BPPv$_u$_c benchmark, with 1449 and 902 optimal solutions found, respectively, for a total of 2351, followed by Elimination20, with 1403 optimal solutions found in the classic benchmark and 605 in the BPPv$_u$_c benchmark for 2008 total optimal solutions. Elimination30, with 542 instances optimally solved, and Elimination40, with 513, outperform

Elimination, with 512, in the BPPv$_u$_c benchmark; however, Elimination, with 1442 optimal solutions, had a better performance in the classic benchmark than Elimination30 and Elimination40, with 1368 and 1332, respectively, and also obtained more optimal solutions in total, with 1954 optimal solutions found for Elimination, while Elimination30 found 1910 and Elimination40 found 1845. It is important to note that the versions that gave the best results are the ones that eliminate the least number of bins, as we will discuss this again later.

As mentioned, the Merge & Split operator works as an Elimination operator with a fixed number of groups to eliminate, equal to two, and the groups are randomly selected. It obtained the optimal solution of 1517 instances in the classic benchmark and 1007 in BPPv$_u$_c, for a total of 2524 instances. It performed the best of all the five state-of-the-art operators and even outperformed the Adaptive operator with random bin selection.

Table 3. Comparison of the group-oriented operators and their different versions with respect to the number of optimal solutions found in each of the nine classes of the classic benchmark

Class	Instances	Adaptive	Adaptive-R	Elimination	Elimination 10	Elimination 20	Elimination 30	Elimination 40	Merge & Split
Data set 1	720	**720**	710	688	688	675	671	663	719
Data set 2	480	**480**	464	464	464	446	432	416	477
Data set 3	10	**10**	9	9	9	8	8	5	**10**
Triplets	80	**80**	16	1	8	9	3	0	9
Uniform	80	**80**	74	62	60	45	38	36	79
Hard28	28	**14**	8	5	5	5	5	5	8
Was 1	100	**100**	**100**	**100**	**100**	**100**	**100**	99	**100**
Was 2	100	**100**	99	**100**	**100**	**100**	99	98	**100**
Gau 1	17	**16**	15	13	15	15	12	10	15
Total	1615	**1600**	1495	1442	1449	1403	1368	1332	1517

Table 4. Comparison of the group-oriented operators and their different versions with respect to the number of optimal solutions found in each of the four classes of the BPPv$_u$_c benchmark

Class	Instances	Adaptive	Adaptive-R	Elimination	Elimination 10	Elimination 20	Elimination 30	Elimination 40	Merge & Split
BPP.25	700	**443**	255	204	329	216	202	200	297
BPP.5	700	**322**	201	106	219	112	110	101	273
BPP.75	700	**239**	143	100	123	102	100	100	198
BPP1	700	**478**	338	102	231	175	130	112	239
Total	2800	**1482**	937	512	902	605	542	513	1007

The Swap operator implemented only makes exchanges between one item from group G_1 and one item from group G_2 (a 1-1 move). The items are randomly selected, and if the exchange is not possible due to exceeding any bin capacity, then the movement is not performed, and the solution will remain the same. According to Tables 5 and 6, this operator found the optimal solution of 1446 instances of the classic benchmark and 700 instances of the BPPv$_u$_c benchmark, totaling 2146 instances. It has the second highest number of instances solved by item-oriented operators.

Regarding the Insertion operator, Tables 5 and 6 show that it found the optimal solution of 1497 instances of the classic benchmark and 930 of the BPPv$_u$_c benchmark, for a total of 2427 instances. It is the item-oriented operator that solved the most instances in both benchmarks.

For the last item-oriented operators, Item Elimination, like in the Elimination operator, five different implementations were made, each with a different value for the threshold: 0.1, 0.3, 0.4, 0.5, and 0.8. Item Elimination with a 0.1 threshold obtained the optimal solution of 1307 instances of the classic benchmark and 532 of the BPPv$_u$_c benchmark; Item Elimination with a 0.3 threshold found the optimal solution of 1238 instances of the classic benchmark and 491 of the BPPv$_u$_c benchmark; Item Elimination with a 0.4 threshold optimally solved 1216 instances of the classic benchmark and 494 of the BPPv$_u$_c benchmark; Item Elimination with a 0.5 threshold found the optimal solution of 1210 instances of the classic benchmark and 491 of the BPPv$_u$_c benchmark; and finally, Item Elimination with a 0.8 threshold obtained the optimal solution of 1194 instances of the classic benchmark and 460 of the BPPv$_u$_c benchmark. Fig. 3 shows that by comparing the total number of instances solved for the different versions of this operator, the lower the threshold, the better the results were.

The best operator in the results is the original operator in GGA-CGT, the Adaptive operator, as shown in Fig. 3, followed by Merge & Split, Adaptive-R, Insertion, Elimination10, and Swap. It can be observed that the best operators are the least disruptive operators; Adaptive only eliminates a small number of bins after its calculations; Merge & Split only eliminates a fixed number of two bins; Insertion only eliminates items in a single bin; Elimination10 only eliminates 10 percent of the total number of bins; and Swap only does 1-1 movements, which only affect the two selected bins. As we increase the percentage of Elimination, the performance decreases, and Elimination with a variable number of *k* groups to eliminate is still too disruptive. With Item Elimination, we can also observe a decline in performance as we increase the threshold; however, even with a low threshold, the operator underperforms compared to the rest.

Table 5. Comparison of the item-oriented operators and their different versions with respect to the number of optimal solutions found in each of the nine classes of the classic benchmark

Class	Instances	Swap	Insertion	Item Elimination 0.1	Item Elimination 0.3	Item Elimination 0.4	Item Elimination 0.5	Item Elimination 0.8
Data set 1	720	711	**718**	672	656	653	651	647
Data set 2	480	452	**465**	386	365	358	353	350
Data set 3	10	**8**	**8**	**8**	0	0	0	0
Triplets	80	0	**5**	1	0	0	0	0
Uniform	80	72	**79**	40	33	31	31	24
Hard28	28	5	**7**	5	5	5	5	5
Was 1	100	89	**100**	93	81	72	73	72
Was 2	100	96	**100**	93	91	90	90	90
Gau 1	17	13	**15**	9	7	7	7	6
Total	1615	1446	**1497**	1307	1238	1216	1210	1194

Table 6. Comparison of the item-oriented operators and their different versions with respect to the number of optimal solutions found in each of the four classes of the BPPv$_u$_c benchmark

Class	Instances	Swap	Insertion	Item Elimination 0.1	Item Elimination 0.3	Item Elimination 0.4	Item Elimination 0.5	Item Elimination 0.8
BPP.25	700	253	**313**	199	195	197	195	175
BPP.5	700	**200**	**200**	102	101	100	100	100
BPP.75	700	139	**199**	100	99	100	100	97
BPP1	700	108	**218**	131	96	97	96	88
Total	2800	700	**930**	532	491	494	491	460

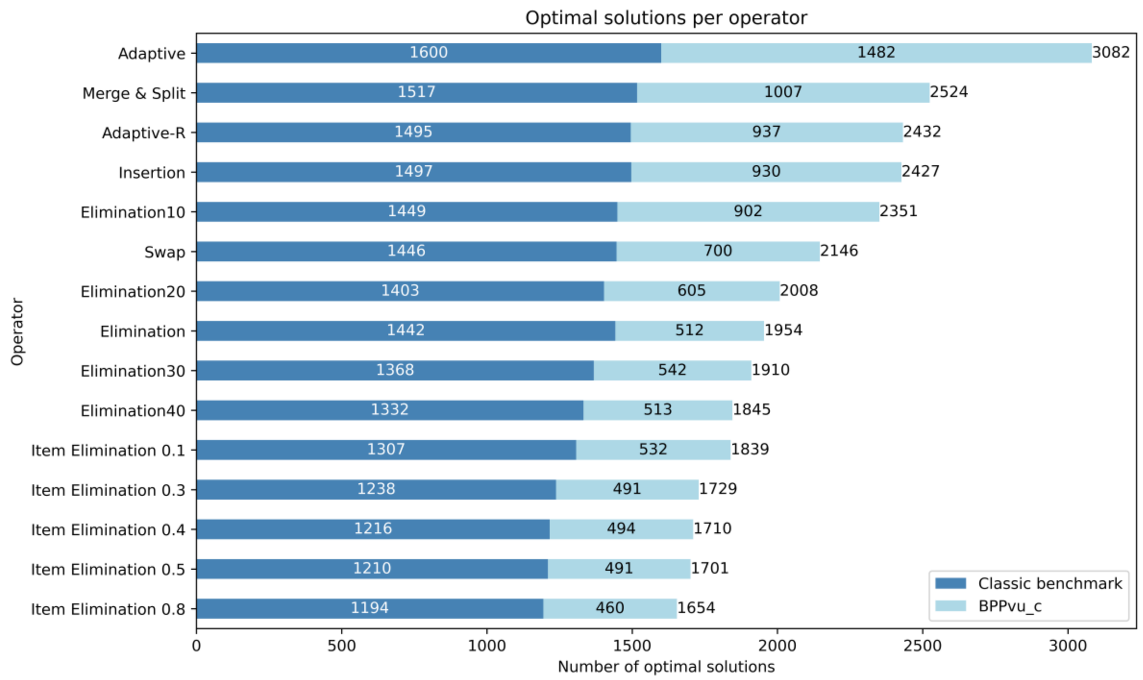

Fig. 3. The number of optimal solutions found in total for all the operators and their different versions implemented.

After seeing the difference in the results of the Adaptive operator, where the bin selection was the n_b least full bins, and Adaptive-R, where the bin selection for the n_b bins is random, more tests were performed for the five best implemented versions: Merge & Split, Insertion, Elimination10, Swap, and Elimination20. In these tests, the bin selection was done regarding the bin fullness, choosing the least full bins instead of randomly selecting bins. Merge & Split would eliminate the two least full bins, Insertion would select the least full bin for later eliminating items from it, Elimination10 would select the least full bins to eliminate after calculating the fixed 10 percent of bins to eliminate, Swap would choose an item from each of the two least full bins, and Elimination20 would select the least full bins to eliminate after calculating the fixed 20 percent of bins to eliminate. Tables 7 and 8 and Figs. 4 and 5 show the results of these experiments. The termination -R indicates the version of the operators that use the random bins selection, and -B indicates the version of the operators that use the least full bins selection.

Table 7. Number of optimal solutions found in each of the nine classes of the classic benchmark for the top five operator versions when modifying least full bins

Class	Instances	Merge & Split	Insertion	Elimination 10	Swap	Elimination 20
Data set 1	720	**717**	710	714	706	689
Data set 2	480	469	434	**472**	435	449
Data set 3	10	**10**	8	9	6	8
Triplets	80	21	1	**27**	0	20
Uniform	80	**73**	72	70	72	54
Hard28	28	8	5	**11**	5	6
Was 1	100	**100**	95	96	91	**100**
Was 2	100	**100**	97	**100**	94	99
Gau 1	17	**16**	13	**16**	12	**16**
Total	1615	1514	1435	**1515**	1421	1441

Table 8. Number of optimal solutions found in each of the four classes of the BPPv$_u$_c benchmark for the top five operator versions when modifying least full bins

Class	Instances	Merge	Insertion	Elimination	Swap	Elimination

		& Split		10		20
BPP.25	700	371	241	240	231	**386**
BPP.5	700	256	200	**299**	197	247
BPP.75	700	201	140	**229**	126	156
BPP1	700	225	108	**416**	101	193
Total	2800	1053	689	**1184**	655	982

While the group-oriented operators improved the algorithm performance with the least full bins selection, the item-oriented operators underperformed with it. Elimination10 optimally solved slightly more instances, 66, in the classic benchmark and showed a greater improvement in the BPPv$_u$_c benchmark, finding the optimal solution of 282 more instances for a total increase of 348 instances while selecting the least full bins for elimination rather than selecting bins at random. Merge & Split found three fewer optimal solutions with the least full bins selection. However, it improved its performance in the BPPv$_u$_c benchmark by obtaining 46 more optimal solutions for a total improvement of 43 instances over the random bin selection. Elimination20 with the least full bins selection found 38 more optimal solutions in the classic benchmark and 377 in the BPPv$_u$_c benchmark for 415 more instances optimally solved than when selecting bins at random. Insertion obtained 62 fewer optimal solutions in the classic benchmark and 241 fewer optimal solutions in the BPPv$_u$_c benchmark with the least full bins selection for a total decrease of 303 fewer optimal solutions than when random bin selection is used. Finally, Swap found 25 fewer optimal solutions in the classic benchmark and 45 fewer in the BPPv$_u$_c benchmark with the least full bins selection, obtaining 70 fewer optimal solutions than when selecting bins at random.

The drop in performance for the item-oriented operators could be due to the least full bins having large items that cannot be reinserted in other groups through the RP heuristic and end up reinserted in the same group or a new one with the FF heuristic. Since these eliminated items were in the last bins due to them not fitting in other bins, they could be packed in the same bin again, and therefore, the algorithm does not explore the search space enough. So far, the results have shown that an operator works best when it is not very disruptive, when selecting the worst bins in the solution, and when not eliminating large items that cannot fit in other bins.

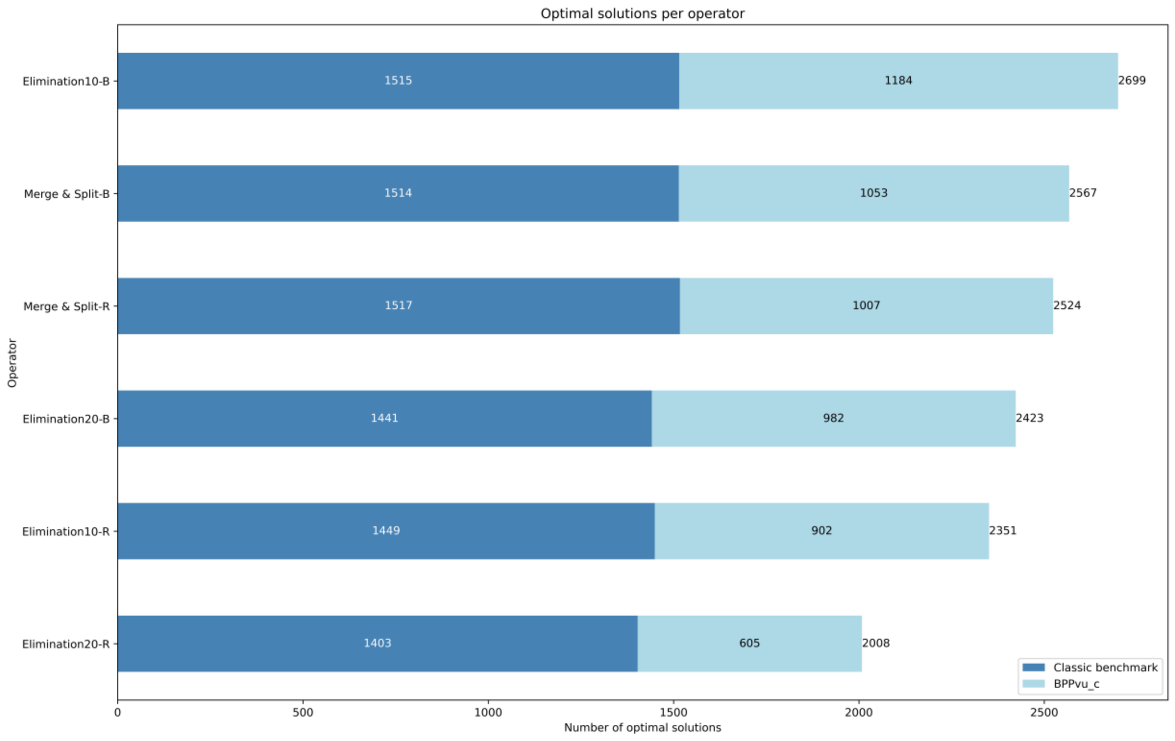

Fig. 4. The number of optimal solutions found in total for the best three group-oriented operators and the Adaptive operator with both random bin selection and least bin fullness selection.

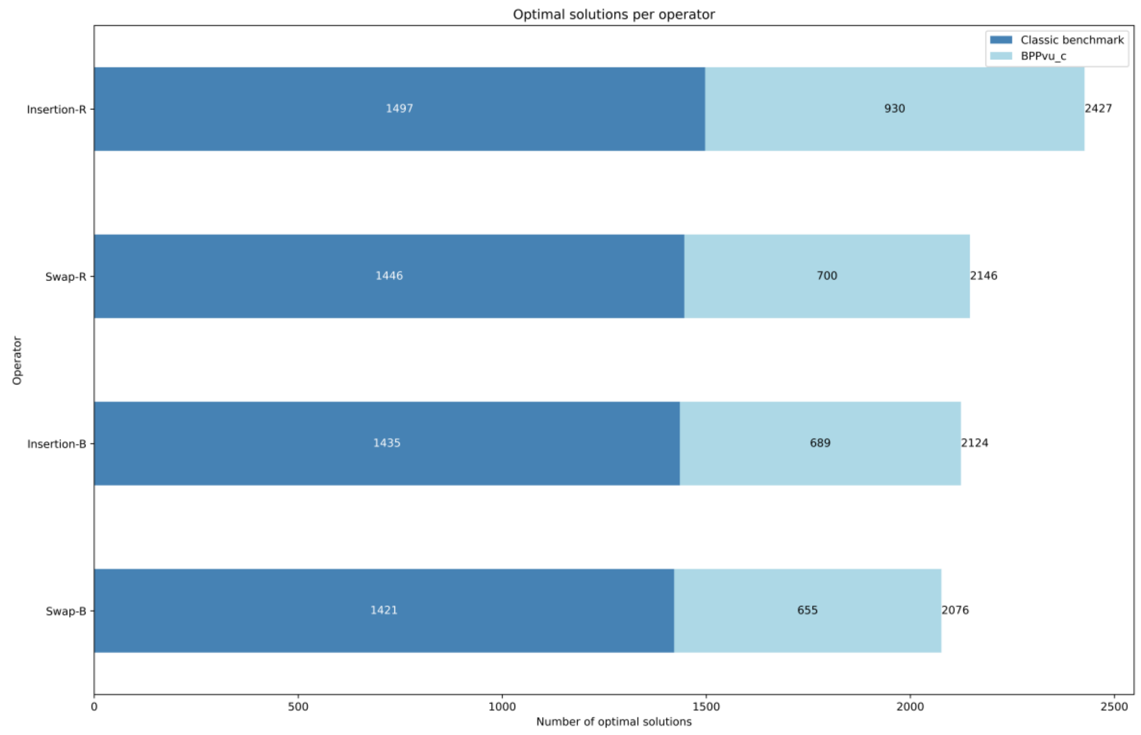

Fig. 5. The number of optimal solutions found in total for the best two item-oriented operators with both random bin selection and least bin fullness selection.

Fig. 6 illustrates the results of this implementation compared to the results of the other Item Elimination versions. A significant improvement can be observed for this version of Item Elimination. The Adaptive IE 0.5 operator found the optimal solution of 1454 instances in the classic benchmark, 147 more than Item Elimination with a 0.1 threshold (previously the best Item Elimination version) and 244 more than Item Elimination with a 0.5 threshold without the Adaptive bin selection or small item selection. In the BPPv$_u$_c benchmark, the Adaptive IE 0.5 operator found the optimal solution of 822 instances, 290 more than Item Elimination 0.1 and 331 more than Item Elimination 0.5.

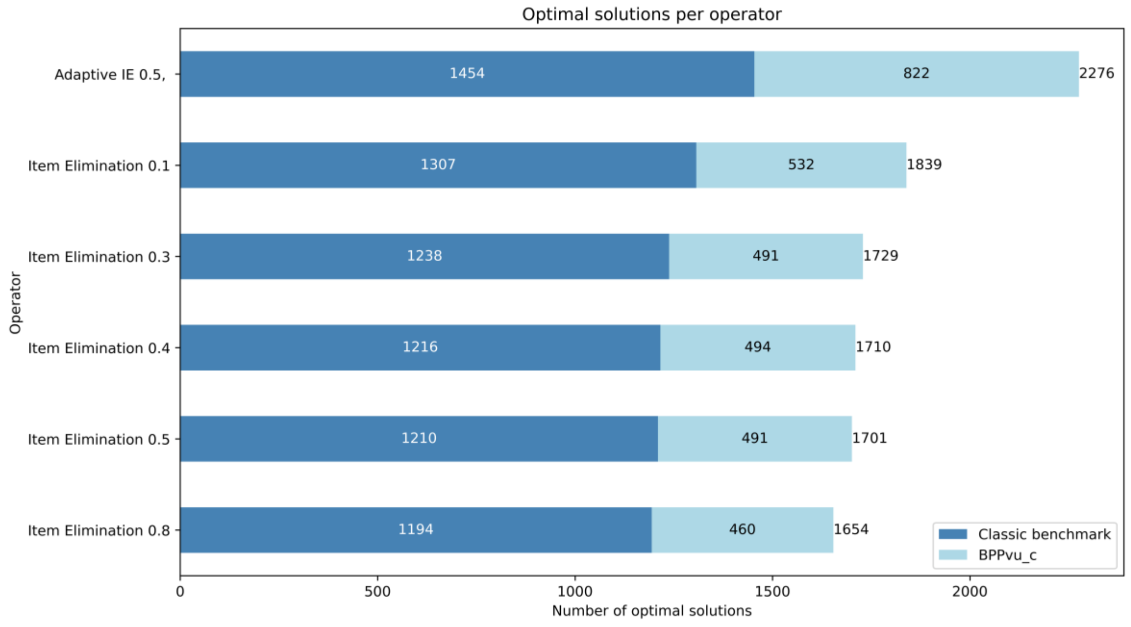

Fig. 6. The total number of optimal solutions found for the Adaptive IE 0.5 operator compared to the other implementations of the Item Elimination operator.

In the previous implementations of Item Elimination, a decrease in the threshold resulted in better operator performance; with this in mind, two more versions of the Adaptive IE operator were implemented. Adaptive IE with threshold values of 0.25 and 0.75. Tables 9 and 10 present the results of the three versions of Adaptive IE for both the classic benchmark and the BPPv$_u$_c benchmark. Adaptive IE 0.25 obtained the worst results in terms of number of optimal solutions found with 1435 in the classic benchmark and 760 in the BPPv$_u$_c benchmark, for a total of 2141; Adaptive IE 0.5 found the optimal solution of 1454 instances in the classic benchmark and 822 in the BPPv$_u$_c benchmark, for a total of 2276; Adaptive IE 0.75 obtained the best results by finding the optimal solution of 1470 instances in the classic benchmark and 863 instances in the BPPv$_u$_c benchmark, for a total of 2333.

Additionally, to assess the impact of this operator on the algorithm's convergence time, an additional test was performed where the fitness of the best solution of each generation is compared between the three different versions of Adaptive IE and the original Adaptive operator in the GGA-CGT. The instance used for this test was BPP.75_10000_1 from the BPPv$_u$_c benchmark, since neither operator tested could find the optimal solution of this instance. It can be observed in Fig. 7 that neither version of Adaptive IE presents earlier convergence compared to the original operator in the GGA-CGT. Adaptive IE 0.25 presents slower convergence and does not reach the same fitness as the best solution. Moreover, while Adaptive IE 0.5 and 0.75 both reach the same fitness value as the Adaptive operator, they do not reach it as fast, with Adaptive IE 0.75 reaching it close to 400 generations and Adaptive IE 0.5 reaching it earlier, close to 250 generations. Meanwhile, the original GGA-CGT reaches the same fitness value close to 100 generations.

Table 9. Number of optimal solutions found in each of the nine classes of the classic benchmark for the three implemented version of the Adaptive IE operator

Class	Instances	Adaptive IE 0.25	Adaptive IE 0.5	Adaptive IE 0.75
Data set 1	720	**710**	706	702
Data set 2	480	449	457	**474**
Data set 3	10	8	8	**9**
Triplets	80	1	5	**6**
Uniform	80	58	60	**61**
Hard28	28	5	**6**	**6**
Was 1	100	97	99	**100**
Was 2	100	93	**99**	**99**
Gau 1	17	**14**	**14**	13
Total	1615	1435	1454	**1470**

Table 10. Number of optimal solutions found in each of the four classes of the BPPv$_u$_c benchmark for the three implemented version of the Adaptive IE operator

Class	Instances	Adaptive IE 0.25	Adaptive IE 0.5	Adaptive IE 0.75
BPP.25	700	220	213	**273**
BPP.5	700	206	**237**	201
BPP.75	700	190	198	**199**
BPP1	700	144	174	**190**
Total	2800	760	822	**863**

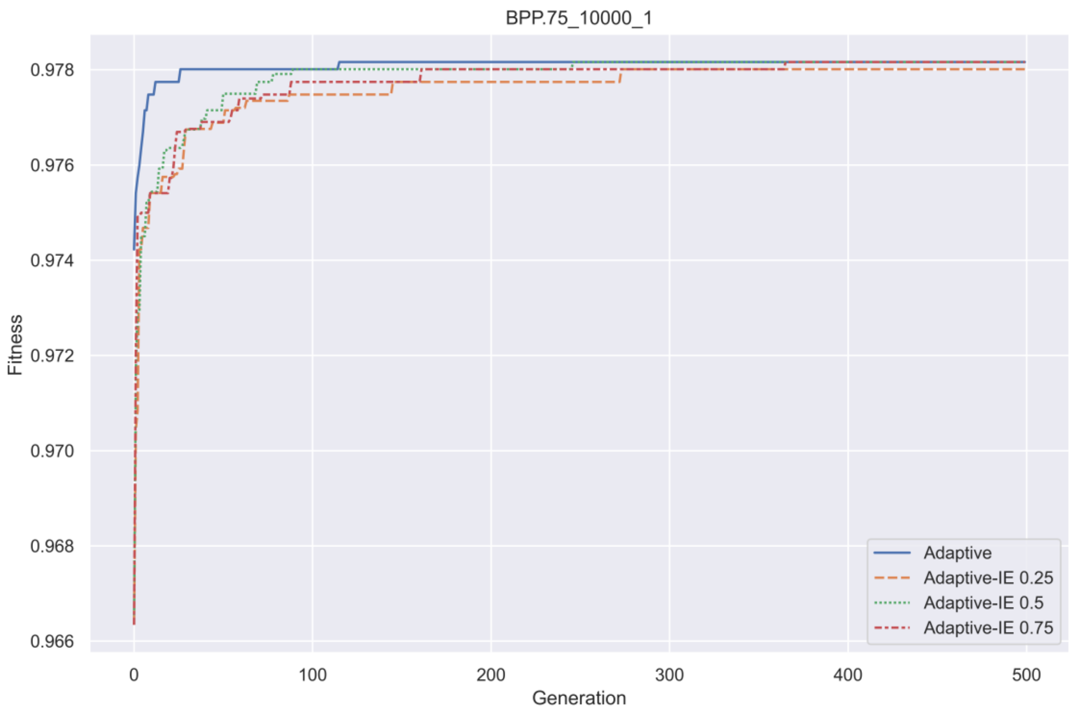

Fig. 7. Convergence graph for comparison of the following operators: Adaptive, Adaptive IE 0.25, 0.5 and 0.75.

Finally, to better evaluate the use of resources, a comparison between the original operator of the GGA-CGT, the Adaptive operator, and Adaptive IE 0.75 is made in Tables 11 and 12, where not only the optimal solutions found per class in each benchmark are compared but also the average time in seconds for each operator for each class and the total average time for each benchmark, as well as the average number of individuals with repeated fitness after mutation for each class and the total average for each benchmark. While the Adaptive operator found more optimal solutions and had a lower computational cost than the Adaptive IE 0.75 operator in both benchmarks, it showed more average individuals with repeated fitness. This difference in computational cost is due to the Adaptive IE operator checking the weight of each item in the selected bins and generating a random probability for those with a weight lower than 50% of the bin capacity. While this helps with diversity regarding the individuals with repeated fitness values after the mutation, it does not result in a competitive number of optimal solutions found compared to the original GGA-CGT operator.

Table 11. Number of optimal solutions found, average time and average number of individuals with repeated fitness value after mutation in each of the nine classes of the classic benchmark for the Adaptive and Adaptive IE 0.75 operators

Class	Instances	Adaptive			Adaptive IE 0.75		
		Opt	Time (s)	Avg Individuals with Repeated Fitness values	Opt	Time (s)	Avg Individuals with Repeated Fitness values
Data set 1	720	**720**	**0.802**	3.555	702	1.140	**0.281**
Data set 2	480	**480**	**0.286**	0.143	474	0.569	**0.019**
Data set 3	10	**10**	**1.934**	0.060	9	4.010	**0.000**
Triplets	80	**80**	**0.330**	0.116	6	4.220	**0.011**
Uniform	80	**80**	**0.226**	**0.119**	61	5.089	0.199
Hard28	28	**14**	**1.399**	0.908	6	1.963	**0.009**
Was 1	100	**100**	**0.005**	0.102	**100**	0.030	**0.070**
Was 2	100	**100**	**0.599**	0.189	99	1.184	**0.029**
Gau 1	17	**16**	**0.219**	0.293	13	0.726	**0.045**
Total	1615	**1600**	**0.644**	0.609	1470	2.103	**0.073**

Table 12. Number of optimal solutions found, average time and average number of individuals with repeated fitness value after mutation in each of the nine classes of the BPPv$_u$_c benchmark for the Adaptive and Adaptive IE 0.75 operators

Class	Instances	Adaptive			Adaptive IE 0.75		
		Opt	Time(s)	Avg Individuals with Repeated Fitness values	Opt	Time(s)	Avg Individuals with Repeated Fitness values
BPP.25	700	**443**	**5.838**	0.476	273	7.970	**0.231**
BPP.5	700	**322**	**4.209**	0.617	201	5.339	**0.298**
BPP.75	700	**239**	**2.606**	3.167	199	1.697	**0.427**
BPP1	700	**478**	1.851	4.921	190	**1.482**	**0.816**
Total	2800	**1482**	**3.626**	2.295	863	4.122	**1.772**

6 Conclusions

In this paper, we presented an experimental study in which we compared and measured the performance of the GGA-CGT for the 1D-BPP with five different grouping mutation operators and tested two bin selection strategies and two item selection strategies. The conclusions are presented as follows.

Our first experiments showed that the least disruptive operators performed better in the GGA-CGT compared to highly disruptive ones. For the top five performing operators in the first test, an additional test was conducted where the bin selection was changed from random to selecting the least full bins for the mutation operations. These changes resulted in improved performance for the group-oriented operators and a decline in performance for the item-oriented operators, likely due to complications with eliminating and reinserting large-sized items. Finally, bin and item selection strategies were tested on an item-oriented operator, where only a calculated number of the least full bins and only small items from those bins were selected for the operations. These strategies resulted in an improvement of 26% in the operator's performance. We conclude that the bin selection strategy significantly impacts the performance of a group-oriented operator. In contrast, the item selection strategy has a greater impact on the item-oriented operators. However, more experiments need to be conducted with more operators to understand further the impact of these strategies on mutation operators. Understanding these characteristics can lead us to further improve all the mutation operators presented in this study, including the adaptive operator, by following a strategy similar to that of the Adaptive IE operator. The GGA-CGT is already one of the best algorithms in the specialized literature to solve the 1D-BPP, and applying this methodology could make it more competitive with other state-of-the-art methods.

More gene selection strategy tests and item selection strategies are planned for future work. More versions of the operators will also be implemented, like adding more options for movements in the Swap operator, testing different values for thresholds for the Item Elimination operator variations, or percentages for the Elimination operator. Finally, we plan to culminate this work in the future by designing a new grouping mutation operator using the knowledge obtained from this study.

References

Abdullah, J. M., & Ahmed, T. (2019). Fitness dependent optimizer: inspired by the bee swarming reproductive process. *IEEe Access*, *7*, 43473-43486.

Abdul-Minaam, D. S., Al-Mutairi, W. M. E. S., Awad, M. A., & El-Ashmawi, W. H. (2020). An adaptive fitness-dependent optimizer for the one-dimensional bin packing problem. *IEEe Access*, *8*, 97959-97974.

Ali, A. I., Keedwell, E., & Helal, A. (2024, July). A Differential Pheromone Grouping Ant Colony Optimization Algorithm for the 1-D Bin Packing Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1463-1469).

Alvim, A. C., Ribeiro, C. C., Glover, F., & Aloise, D. J. (2004). A hybrid improvement heuristic for the one-dimensional bin packing problem. *Journal of heuristics*, *10*, 205-229.

Amador-Larrea, S. (2022). Un estudio de operadores genéticos de cruza para el problema de empacado de objetos en contenedores. Masters Thesis, Universidad Veracruzana.

Amador-Larrea, S. (2022). Un estudio de operadores genéticos de cruza para el problema de empacado de objetos en contenedores.

Bayraktar, T., Aydin, M. E., & Dugenci, M. (2014). A memory-integrated artificial bee algorithm for 1-D bin packing problems.

Borgulya, I. (2021). *A hybrid evolutionary algorithm for the offline bin packing problem. CEJOR 29, 425–445.*

Buljubašić, M., & Vasquez, M. (2016). Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. *Computers & Operations Research*, *76*, 12-21.

Carmona-Arroyo, G., Vázquez-Aguirre, J. B., & Quiroz-Castellanos, M. (2021). One-dimensional bin packing problem: An experimental study of instances difficulty and algorithms performance. *Fuzzy Logic Hybrid Extensions of Neural and Optimization Algorithms: Theory and Applications*, 171-201.

Cellini, L., Macaluso, A., & Lombardi, M. (2024). Qal-bp: an augmented lagrangian quantum approach for bin packing. *Scientific Reports*, *14*(1), 5142.

Cruz-Reyes, L., Quiroz C, M., F Alvim, A. C., Fraire Huacuja, H. J., Gómez S, C., & Torres-Jiménez, J. (2012). Efficient hybrid grouping heuristics for the bin packing problem. *Computación y sistemas*, *16*(3), 349-360.

Dokeroglu, T., & Cosar, A. (2014). Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. *Computers & Industrial Engineering*, *75*, 176-186.

El-Ashmawi, W. H., & Abd Elminaam, D. S. (2019). A modified squirrel search algorithm based on improved best fit heuristic and operator strategy for bin packing problem. *Applied Soft Computing*, *82*, 105565.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, *2*, 5-30.

Fernández-Solano, O. (2023). Un estudio experimental de operadores de mutación para el algoritmo genético de agrupación para la segmentación de imágenes en RGB.

Fleszar, K., & Charalambous, C. (2011). Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. *European Journal of Operational Research*, *210*(2), 176-184.

Fleszar, K., & Hindi, K. S. (2002). New heuristics for one-dimensional bin-packing. *Computers & operations research*, *29*(7), 821-839.

Flores-Torres, L., Amador-Larrea, S., Quiroz-Castellanos, M., & Ramos-Figueroa, O. (2023, November). Deterministic Parameter Control Applied to the Grouping Genetic Algorithm with Controlled Gene Transmission. In *2023 10th International Conference on Soft Computing & Machine Intelligence (ISCMI)* (pp. 42-46). IEEE.

Gherboudj, A. (2019). African buffalo optimization for one dimensional bin packing problem. *International Journal of Swarm Intelligence Research (IJSIR)*, *10*(4), 38-52.

Gonzalez San Martin, J. E. (2021). Un estudio formal de heurísticas para el problema de empacado de objetos de una dimensión. Masters Thesis, Instituto tecnológico de Ciudad Madero.

González-San-Martín, J., Cruz-Reyes, L., Gómez-Santillán, C., Fraire, H., Rangel-Valdez, N., Dorronsoro, B., & Quiroz-Castellanos, M. (2023). Comparative study of heuristics for the one-dimensional bin packing problem. In *Hybrid Intelligent Systems Based on Extensions of Fuzzy Logic, Neural Networks and Metaheuristics* (pp. 293-305). Cham: Springer Nature Switzerland.

Gupta, J. N., & Ho, J. C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production planning & control*, *10*(6), 598-603.

Jain, M., Singh, V., & Rani, A. (2019). A novel nature-inspired algorithm for optimization: Squirrel search algorithm. *Swarm and evolutionary computation*, *44*, 148-175.

Kucukyilmaz, T., & Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. *Computers & Industrial Engineering*, *125*, 157-170.

Layeb, A., & Chenche, S. (2012). A novel grasp algorithm for solving the bin packing problem. *International Journal of Information Engineering and Electronic Business*, *4*(2), 8.

Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research society*, *55*(7), 705-716.

Loh, K. H., Golden, B., & Wasil, E. (2008). Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research*, *35*(7), 2283-2291.

Martello, S., & Toth, P. (1990). Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, *28*(1), 59-70.

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

Munien, C., Mahabeer, S., Dzitiro, E., Singh, S., Zungu, S., & Ezugwu, A.E. (2020). Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study. *IEEE Access, 8*, 227438-227465.

Ozcan, S. O., Dokeroglu, T., Cosar, A., & Yazici, A. (2016). A novel grouping genetic algorithm for the one-dimensional bin packing problem on gpu. In *Computer and Information Sciences: 31st International Symposium, ISCIS 2016, Kraków, Poland, October 27–28, 2016, Proceedings 31* (pp. 52-60). Springer International Publishing.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., & Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Computers & Operations Research*, *55*, 52-64.

Ramos-Figueroa, O. (2022). Efficient Heuristic Strategies for the Parallel-Machine Scheduling Problem with Unrelated Machines and Makespan Minimization. Phd thesis, Universidad Veracruzana, Xalapa, Veracruz.

Ramos-Figueroa, O., & Quiroz-Castellanos, M. (2024). An experimental approach to designing grouping genetic algorithms. *Swarm and Evolutionary Computation*, 86, 101490.

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., & Kharel, R. (2021). Variation operators for grouping genetic algorithms: A review. *Swarm and Evolutionary computation*, *60*, 100796.

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., & Schütze, O. (2019). Metaheuristics to solve grouping problems: a review and a case study. *Swarm Evol Comput. 2020; 53: 100643*.

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., & Cruz-Ramírez, N. (2023). An experimental study of grouping mutation operators for the unrelated parallel-machine scheduling problem. *Mathematical and Computational Applications*, *28*(1), 6.

Schoenfield, J. E. (2002). Fast, exact solution of open bin packing problems without linear programming. Draft. *US Army Space & Missile Defence Command, Huntsville*, *20*, 72.

Scholl, A., Klein, R., & Jürgens, C. (1997). Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, *24*(7), 627-645.

Schwerin, P., & Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International transactions in operational research*, *4*(5-6), 377-389.

Shem-Tov, E., & Elyasaf, A. (2024, July). Deep Neural Crossover: A Multi-Parent Operator That Leverages Gene Correlations. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 1045-1053).

Singh, A., & Gupta, A. K. (2007). Two heuristics for the one-dimensional bin-packing problem. *OR Spectrum*, *29*, 765-781.

Vázquez-Aguirre, J. B., Carmona-Arroyo, G., Quiroz-Castellanos, M., & Cruz-Ramírez, N. (2024). Causal Analysis to Explain the Performance of Algorithms: A Case Study for the Bin Packing Problem. *Mathematical and Computational Applications*, *29*(5), 73.

Wäscher, G., & Gau, T. (1996). Heuristics for the integer one-dimensional cutting stock problem: A computational study. *Operations-Research-Spektrum*, *18*, 131-144.

Zavaleta-García, N. A. (2023). Estudio experimental de operadores genéticos de cruza para el problema de segmentación de imágenes en color.

Zendaoui, Z., & Layeb, A. (2016). Adaptive cuckoo search algorithm for the bin packing problem. In *Modelling and Implementation of Complex Systems: Proceedings of the 4th International Symposium, MISC 2016, Constantine, Algeria, May 7-8, 2016, Constantine, Algeria* (pp. 107-120). Springer International Publishing.