Optimizing Swarm Intelligence: A Comprehensive Analysis of Mutation-Based Enhancements

Muchamad Kurniawan*[1], Gusti E. Yuliastuti[1], Siti Agustini[2], Maftahatul Hakimah[1], and Wahyu Widyanto[1].
[1] Institut Teknologi Adhi Tama Surabaya/ Department of Informatics, Surabaya, Indonesia
[2] Institut Teknologi Sepuluh Nopember/ Electrical Engineering, Surabaya, Indonesia
E-mail: muchamad.kurniawan@itats.ac.id, gustieky@itats.ac.id, 7022211005@student.its.ac.id, hakimah.mafta@itats.ac.id, wahyuuwy01@gmail.com
*Corresponding author

Keywords: mutation, swarm intelligence optimization, exploration optimization, exploitation optimization

Received: December 7, 2023

Swarm Intelligence (SI) represents an optimization approach inspired by the collective behavior observed in swarms during the search for food. Well-established SI methods, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Artificial Bee Colony (ABC), are complemented by newer methodologies like Cat Swarm Optimization (CSO) and Grasshopper Optimization Algorithm (GOA). Typically, exploration techniques in SI are more effective than exploitation techniques. To enhance exploration capabilities, this research employs a modification technique based on mutation, chosen for its strong exploratory attributes and low complexity.
This study introduces 16 modifications by combining four frameworks with four operators. Each modification is paired with the fundamental methods for comprehensive testing. The experimental phase encompasses five benchmark functions of varying dimensions, resulting in 8,000 experiments. Three analytical assessments were conducted based on these results.
The initial analysis reveals that the mutation modification has the most substantial impact on the basic ACO method. The second analysis indicates that mutation modification significantly influences the objective function in scenarios with large dimensions. The concluding analysis highlights the paramount influence of the modification incorporating the random parameter mutation framework, whereas the mutation operator modification shows comparatively less significant results.
A detailed impact assessment shows that Modification 2B achieved the highest number of positive results, succeeding in 69 out of 100 tests, while 2D modifications yielded the smallest sum and average values. The influence of different frameworks and operators was further analyzed, revealing that frameworks have a more pronounced impact on performance than operators. Framework number 2, in particular, demonstrated the most significant effect on improving average impact values.

Povzetek: Analiziran je vpliv mutacijskih izboljšav na algoritme inteligence rojev (SI), kot so ACO, PSO, ABC, CSO in GOA. Predstavlja 16 modifikacij z združevanjem štirih okvirov in štirih operatorjev mutacije s ciljem izboljšanja raziskovalnih sposobnosti algoritmov. Rezultati kažejo, da imajo okviri mutacije večji vpliv na učinkovitost kot operatorji, pri čemer modifikacija z naključnim parametrom mutacije dosega najboljše rezultate.

1 Introduction

Optimization is a pivotal method used to achieve the most optimal results from an objective function. Optimization methods are generally classified into classic and modern approaches, with frameworks categorized as either metaheuristics or heuristics. Metaheuristic optimization algorithms can be further distinguished based on behavioral similarity or artificial sources of intelligence.

Early metaheuristic optimization algorithms predominantly focused on evolution and swarm intelligence, with a particular emphasis on swarm-based algorithms [1]. Evolution-based algorithms, such as Genetic Algorithm (GA) and Differential Evolution (DE), use selection, crossover, and mutation operators. Swarm intelligence algorithms, like Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Cat Swarm Optimization (CSO), and Grasshopper Optimization Algorithm (GOA), rely on internal swarm interactions to facilitate various search solutions. On the other hand, swarm intelligence algorithms are typified by interactions within the internal swarm. These interactions serve diverse purposes, such as facilitating various search solutions or inducing alterations in particle values. Notable examples of swarm intelligence algorithms comprise Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Particle Swarm Improvements in exploration using the mutation operator have been applied to enhance the standard Particle Swarm Optimization (PSO) [8], [9]. The Hybrid PSO with Mutation Operator (HPSOM) method was developed specifically for implementation as a clustering algorithm, demonstrating superior results compared to two other optimization algorithms across six measurements and datasets [4].

Wang et al. proposed PSO Mutation (MPSO) to address multi-objective function problems, specifically with thermoelectric generator (TEG) datasets [10]. In this study, the mutation process is controlled by a parameter with a random value. If the random value exceeds the parameter value, one individual undergoes mutation; otherwise, a position change is made, akin to standard PSO.

Exploration and exploitation are two key techniques within optimization algorithms. Exploration refers to the algorithm's ability to search the entire space for the optimal solution, while exploitation involves refining solutions to converge towards the optimal outcome. Swarm intelligence methods typically excel at exploitation but often struggle with exploration, leading to premature convergence and suboptimal solutions.

To enhance the exploration capabilities of PSO, Jana et al. introduced the repository and mutation technique to create RMPSO. The fitness values and convergence rates of RMPSO were compared with seven previously modified PSO studies [11]. Optimization (PSO), Cat Swarm Optimization (CSO), and Grasshopper Optimization Algorithm (GOA) [2]. It is noteworthy that the landscape of metaheuristic optimization is dynamic, continually witnessing the assimilation of novel algorithms and techniques.

Exploration and exploitation represent two pivotal techniques within optimization algorithms. Exploration pertains to the algorithm's efficacy in locating the optimal solution across the entire search space. Conversely, exploitation involves the capacity of individuals to update their values with the goal of converging towards the optimal solution. Generally, specific operators or parameters govern these two techniques. A higher exploitation value accelerates the algorithm's convergence to a solution, but the obtained solution is susceptible to local optima or spurious solutions. Swarm intelligence methods inherently possess heightened exploitation capabilities relative to exploration. Notably, contemporary studies have incorporated the mutation operator to enhance exploratory capabilities.

To enhance the efficacy of swarm intelligence algorithms, the incorporation of a mutation operator has been proposed as a viable solution [3] [4] [5]. The introduction of a mutation operator serves to augment exploratory capabilities, thereby preventing rapid convergence of the algorithm [6]. An additional advantage associated with the integration of the mutation operator lies in its computational efficiency compared to adjustments involving crossbreeding, modifications to alternative optimization algorithms, or alterations requiring multiple populations [7]. Various studies investigating the incorporation or modification of swarm-based algorithms with mutation operators have consistently demonstrated performance improvements.

Cat Swarm Optimization (CSO), initially proposed by Chu et al. in 2006, is a swarm-based optimization algorithm that has been employed in diverse case studies [11], [12]. While bearing resemblance to PSO, the CSO algorithm features a distinct exploration and exploitation mechanism involving two types of individuals, namely seekers and tracers. A 2018 study reported that the incorporation of a mutation operator into CSO enhances optimal global search capabilities, particularly in high dimensions [14].

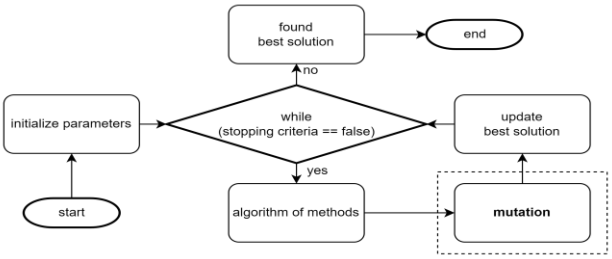


Figure 1: Hybrid framework of swarm-based optimization and mutation processes

The swarm intelligence algorithm by adopting grasshoppers in search of food was first reported by Saremi *et al*. 2017 with the name grasshopper optimization algorithm (GOA). The results of a paper survey on GOA from 2017 to 2020 stated that there were more than 200 implementation or development studies [14]. The limitation of this algorithm is that it is stuck in local optimal and convergent time [16] to be used in various cases. A development of GOA with the addition of the Caucy mutation was reported in 2021 [17]. Research conducted by Ghaleb *et al*. in 2021 using six different mutation operators implemented in GOA, the results of this study were not a significant increase when compared to the standard GOA [18].

Studies of hybrid mutation and ACO discuss the implementation of a random injection operator on ACO by looking at the diversity value of the swarm [18]. Two-point standard mutation and refined mutation are operators used in research [20], and partial ACO mutation is the algorithm's name with the GA standard operator implemented in ACO partially [21]. Cauchy mutation and ABC algorithm can improve standard ABC performance [22]. The uniform mutation is used for searching on the ABC algorithm [23]. These research show that hybrid mutation can improve performance. However, the level of randomness of the mutation technique is still low, so the resulting solution is pre-mature convergence.

To address this limitation, recent studies have incorporated mutation operators into swarm intelligence algorithms to enhance their exploratory capabilities. Mutation operators introduce random variations, preventing rapid convergence and improving the overall performance of the algorithms.

This research aims to improve the exploration capabilities of swarm intelligence algorithms by introducing a new mutation technique using novel operators and frameworks. We propose 16 different combinations of four frameworks and four operators, applied to PSO, GOA, BCO, ACO, and ABC algorithms. The goal is to achieve a more robust level of exploration, leading to more optimal solutions across various swarm-based optimization algorithms. Our contributions include demonstrating the robustness of these mutation methods and their effectiveness in enhancing the performance of swarm intelligence algorithms.

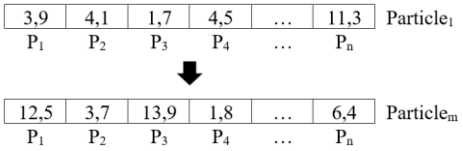


Figure 2: Full injection gaussian random mutation illustration

2 Method

Within this section, we introduce novel mutation techniques. The focus of this research is to categorize mutation into two components: operator and framework. The operator will be represented by an alphabetic code, while the framework will be denoted by a numeric code, simplifying future references. An operator refers to a method employed to modify the value of a particle. In total, there are four distinct types of operators.: (A) full injection gaussian random mutation [24], (B) n-point gaussian random injection mutation, (C) arithmetic random operator mutation, and (D) even odd random injection mutation. Framework mutation is the workflow of the mutation technique. There are four mutation frameworks used: (1) random picker particle mutation, (2) random parameter mutation, (3) sorting mutation, and (4) dynamic sorting mutation. Operators and frameworks cross combined, so there are sixteen mutation techniques: {1A,1B, …,3D,4D}. We assume that the mutation rate value, mr, is 0.3 and there are n particles.

2.1 Hybrid swarm-based and mutation technique

In general, the procedure for initializing a swarm-based algorithm is parameter initialization, searching for each particle, determining the $Gbest$, and updating each particle. This procedure will be repeated until the stopping criteria are met. The main methodology of this research is to modify optimization methods with a mutation process, so a hybrid is needed that can be implemented in all optimization methods. The algorithm used in this study (Figure. 1) is to add mutations after the algorithm process of the optimization method and then update the value for the best solution value or *Gbest* in swarm-based.

2.2 Full injection gaussian random mutation operator

This operator is a mutation operator by utilizing the Gaussian distribution to generate random numbers. The mean and standard deviation values are obtained from all particle values in each dimension. The resulting random number will be used to update the new particle with several dimensions of the objective function as shown in Figure. 2. For example, there are 5 dimensions in an objective function. Each particle that has a mutation will generate a random number of 5 dimensions.

2.3 N-point gaussian random mutation operator

In the n-point gaussian random mutation operator, the randomization process is not carried out at all positions in the particle. The first step is to determine the n number of

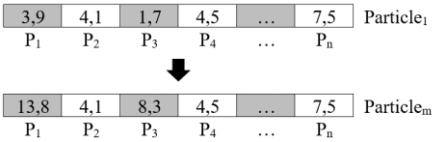


Figure 6: Even Odd Random Injection Mutation Illustration

particle positions carried out to mutation process. The second step is generating other n random value. The resulting random values will replace the particle values at these positions. So that the mutation results are obtained which experience slight changes, because not all the particles are mutated. As an illustration, the value of n is determined to be 3, then there will be 3 positions that are mutated. We need to generate 3 random values as shown in Figure. 3 and then carry out the mutation process at 3 predetermined points. The mutation process is carried out by changing the values of the 3 positions with the values generated by random value generation as shown in Figure. 4.

2.4 Arithmetic random mutation operator

The mutation process in arithmetic random mutation is carried out by involving several arithmetic operators including addition operators, subtraction operators, and multiplication operators. The multiplication operator is used for two conditions, namely the multiplication of positive values and the multiplication of negative values. The first step is to generate a random value (r_1) round value between 1 and 4. The first random value is used to determine the next selected operator process. Then the process continues to generate a second random value (r_2) with decimals. The second random value is used as the value on which the selected operator operates. If the value 1 appears, then the operator used is addition. If value 2 appears, then the operator used is subtraction. If the value 3 appears, then the operator used is the multiplication of positive values. If the value 4 appears, then the operator used is negative multiplication. The simulation of this operator can be seen in Figure. 5, when the value of $r_1 = 3$ or selected operator is multiplication, and the value of $r_2 = 3.2$ as the coefficient value, the updated particle value is obtained from the addition of the particle value with the coefficient value.

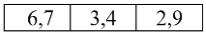


Figure 3: N random generated values

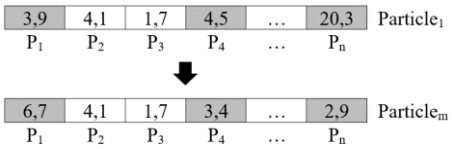


Figure 4: N-Point gaussian random mutation illustration

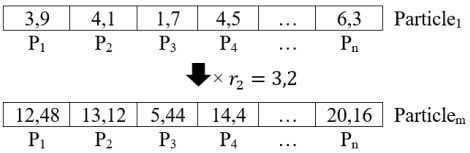


Figure 5: Arithmetic random operator mutation illustration

2.5 Even odd random injection mutation operator

The even odd random injection mutation determines the mutation process based on a binary random value as shown in Fig. 6. If the binary value is 0, then the mutation operator randomizes the value at an even position. Meanwhile, if the value 1 appears, then the mutation operator randomizes the value at the odd gene position [6]. Randomizing the values at these positions is done randomly as shown in Fig. 6. The key distinction of the B, C, and D mutation operators lies in their partial modification of vector values, specifically targeting odd or even indices. These operators involve sampling random values and assigning them to specific dimensions within the decision vector. The range of each dimension within the vector is determined by the minimum and maximum values specified in the range dimension of each objective function.

2.6 Random picker mutation framework

The random picker mutation framework begins with computing the $nMut$ value by multiplying $nParticles$ and mr. The variable $nMut$ is used to determine the number of particles that carried out to mutation process. The particles that carried out to mutation process are determined randomly. The detailed this framework can be seen in Algorithm 1.

Algorithm 1: Random Picker Mutation Framework

```
// mr is mutation rate
// particles is all particle in entire swarm
// nParticles is size swarm
nMut ← floor (nParticles * mr)
for i ← 1 to nMut :
    mutIndex ← random index (particles) ;
    newParticle ← mutation operator (particles[mutIndex]);
    newFitness ← objective function (newParticle) ;
    particles [mutIndex] ← newParticle ;
    fitness [mutIndex] ← newFitness ;
end for
```

2.7 Random parameter mutation framework

The random parameter mutation framework focuses on determining which particles will undergo mutation. We assume that there are n particles. The random values, $rand_i$, where $i = 1, 2, 3, \ldots, n$, are generated and assigned each value into each particle. Each random value, $rand_i$, will be compared to mutation rate, mr. If the random value, $rand_i$ less than mutation rate, mr so the i-th particle will be carried out the mutation process. The new particle and fitness values resulting from the mutation will replace the previous values. The entire steps of algorithms can be seen in Algorithm 2.

Algorithm 2: Framework Random Parameter Mutation

```
// mr is mutation rate
// particles is all particle in entire swarm
// nParticles is size swarm
for i ← 1 to nParticles :
    if rand < mr :
        newParticle ← mutation operator (particles [i]) ;
        newFitness ← objective function (newParticle) ;
        particles [i] ← newParticle ;
        fitness [i] ← newFitness ;
    end if
end for
```

2.8 Sorting mutation framework

The Sorting Mutation Framework (Algorithm 3) works the mutation process by sorting descending particles based on their fitness values. Particles with the worst fitness value will be subject to mutation. The $nMut$ variable controls the number of the worst particle mutations carried $outMut$ variable, which is obtained from the total number of $nParticles$ multiplied by the mutation parameter mr. The mutated particles will replace the old particles.

Algorithm 3: Framework Sorting Mutation

```
// mr is mutation rate
// particles is all particle in entire swarm
// nParticles is size swarm
// fitness is return value from objective function
nMut ← floor (nParticles * mr) ;
worst ← sorting desc index by (fitness) ;
for i ← 1 to nMut :
    newParticle ← mutation operator (particles[worst[i]) ;
    newFitness ← objective function (newParticle) ;
    particles [worst[i]] ← newParticle ;
    fitness [worst[i]] ← newFitness ;
```

end for

2.9 Dynamic sorting mutation framework

The basis of this framework is a sorting mutation framework with dynamic mutation arrangements. The difference is that in this framework, parameters adjust the number of dynamically mutated particles. Setting the number of mutation particles will continue to increase as the number of iterations increases. In Algorithm 4, there is a dynamic $variableMut$, the control parameter for dynamic mutations.

Algorithm 4: Framework Dynamic Sorting Mutation

```
// mr is mutation rate
// particles is all particle in entire swarm
// nPop is size swarm
// fitness is return value from objective function
// maxIter is maximum iteration
dinamicMut ← ceil (( mr/maxIter) * iter) ;
nMut ← floor (nParticles * dinamicMut) ;
worst ← sorting desc index by (fitness) ;
for i ← 1 to nMut :
    newParticle ← mutation operator (particles[worst[i]) ;
    newFitness ← objective function (newParticle) ;
    particles [worst[i]] ← newParticle ;
    fitness [worst[i]] ← newFitness ;
end for
```

2.10 Objective function

The objective function used in this study is divided into two groups. The first group is a function that has a single optimal result or is often called unimodal. The second group is a multimodal function. In the unimodal group the objective functions used are the De Jong, and Rosenbrock functions and the other group functions are Rastrigin, Griewank, and Ackley. The equation used for the De Jong function can be seen in Eq. (1), the Rosenbrock function uses Eq. (2), the Rastrigin, Griwank, Ackley functions use Eq. (3)-(5).

$$f(x) = \sum_{i=1}^{n} x_i^2 \quad (1)$$

$$f(x) = \sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (1 + x_i)^2] \quad (2)$$

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)] \quad (3)$$

$$f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad (4)$$

$$f(x) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(c.x_i)\right) + a + \exp(1) \quad (5)$$

Table 1. Setup parameters of SI

Method	Parameter
PSO	$c_1 = 1; c_2 = 2$
ACO	$alpha = 1;\ beta = 3;\ rho = 0.1$
CSO	$smp = 10;\ mr = 0.2;\ spc = True;$ $cdc = 1;\ srd = 0.01;\ c_1 = 0.5$
GOA	$c_{min} = 0.1;\ c_{max} = 1;$ $lower_{bound} = 1;\ upper_{bound} = 1.5$
ABC	$bee_{explore} = 8; num_{sourcefood} = 2;$

3 Result and discussion

In this study the optimization method or algorithm used is PSO, GOA and ACO. Mutation modification will have 16 modifications from 4 types of frameworks and 4 types of mutation operators that will be added to each method. To make it easier to read, the method given the mutation modification will be affixed with the letter 'M' which indicates modified followed by the code number which indicates the mutation. There are 6 types of benchmark functions used: Rosenbrock, De Jong,

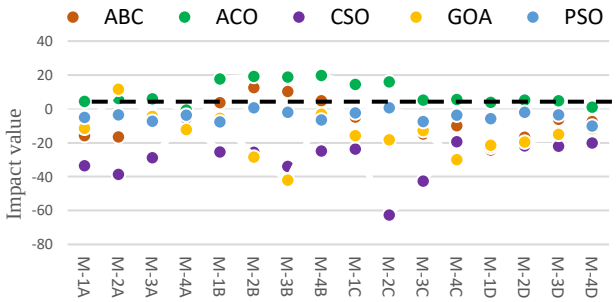


Figure 7: Accumulated total impact value.

Rastrigin, Greiwangk, and Ackley with 4 different dimensions (2, 5, 10, and 30). Because the initial initialization uses random values, to get the best results, the test is carried out ten times for one test scenario. The initialization of parameter values of SI algorithm can be seen in Table 1. These values are fixed so that can measure the effectiveness of research contributions without being influenced by other parameters.

This section will be subdivided into three analyses to ensure a comprehensive evaluation. The first analysis examines the method that exhibits the most significant impact when mutations are introduced. The subsequent analysis investigates the effects of objective function types, specifically unimodal and multimodal functions. Lastly, an analysis is conducted to determine the framework and mutation operators that exert the most substantial influence. The impact calculation is performed using Equation 6. This equation quantifies the difference between the fitness value of the baseline method, denoted as *G(x)*, and the fitness value of the modified method, denoted as *G(x)'*. The difference is divided by the fitness value of the baseline method and then multiplied by 100 to amplify the impact.

$$I = \frac{G(x) - G(x)'}{G(X)} \quad (6)$$

Table 2: Summary of the impact of mutations on basic methods

Method	Best	Worst	Average	Positive Totals
ABC	12,69	-24,15	-5,95	5
ACO	19,70	-0,68	9,24	15
CSO	-5,37	-62,56	-28,13	0
GOA	11,57	-41,96	-14,95	1

3.1 Objective function analysis

To analyze the impact of mutations on the fundamental methods, it is imperative to conduct a comprehensive comparison of mutation effects across all benchmark functions. Figure 7 illustrates the outcomes of all mutation modifications applied to the basic method, with the dotted line denoting the baseline method (having a y-axis value of zero). A positive value signifies that the modified method outperforms the basic method, while a negative value indicates inferior performance. The results of testing the basic method are presented in Figure 7, where a larger positive value denotes a more pronounced impact. Notably, among all the basic methods tested, the ACO method exhibits the most substantial impact, as evidenced by the prevalence of modified methods with values above zero. It is worth mentioning that the addition of mutation modifications exerts varying effects on the ACO method.

In addition to leveraging the difference in mean values, a paired T-Test is employed for a statistical assessment to ascertain whether the modification of the basic method yields significant positive changes. Each method modification result is paired with the method result without modification. With an alpha value of 0.05 and a t-table value of 2.9803, the T-Test results were obtained for all method results, as illustrated in Fig. 8. From these results, the modification that exhibited the most positive effect was observed in the ABC method with the M-1D modification. Among the basic methods, the one that underwent the most successful modifications was the 9th method.

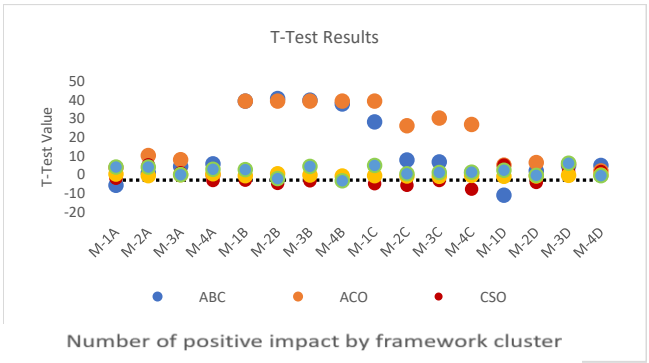
The overall summary of mutation results obtained 4 results as shown in Table 2. The first result is ACO with the M-1C modification gets the best score with a value of 19,70. The second result obtained is the value of the biggest negative impact on CSO. The third result is that only ACO has a positive value from the calculation of the average impact. The final result obtained from all experiments is that the most successful mutation modification is in the ACO method with 15 modifications (out of a total of 16 modifications).

Mutation Impact Analysis of Dimension and Function Problem

To conduct an impact analysis on the introduction of mutations to the dimensions and types of problem functions in this section, the benchmark functions and dimensions will be initially classified. The dimensions of the problem will be categorized into two types: low dimensions (2 and 5 dimensions) and high dimensions (10 and 30 dimensions). Similarly, the types of functions will be distinguished between unimodal functions (Rosenbrock and De Jong) and multimodal functions (Rastrigin, Griewank, and Ackley).

The best fitness results from 10 trials are summarized, including the best value or minimum value, worst value, average, and total, as depicted in Appendix A for unimodal functions and Appendix B for multimodal functions. Following the grouping based on the problem function, the best value results will be further categorized based on the dimensions employed.

Out of the 16 modifications applied to all basic methods and across 10 trials for each test, all basic methods exhibit the best results among the mutation modifications used. The ACO method, with its mutation modification, demonstrates the most significant positive



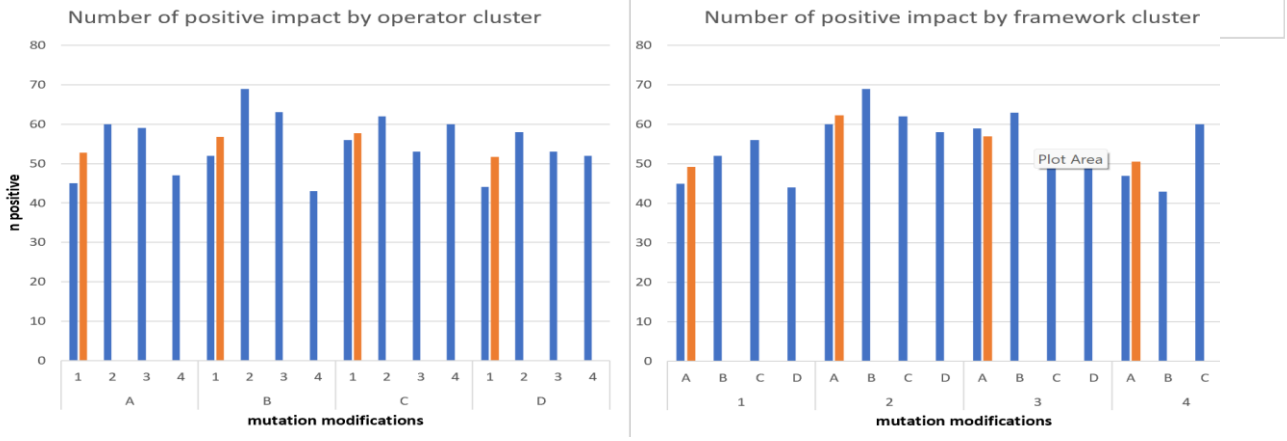


Figure 9: Number of positive impacts by cluster operator and framework mutation modification

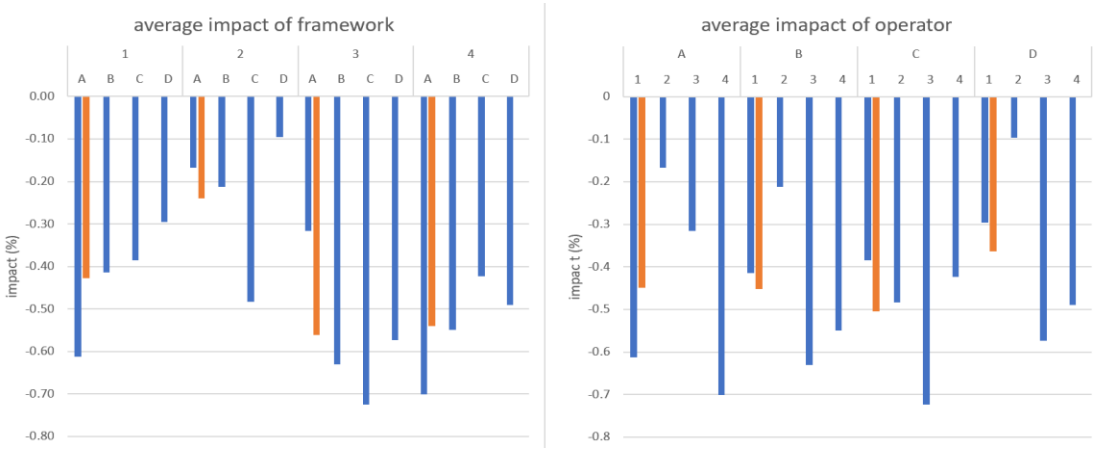


Figure 10: Number of positive impacts by cluster operator and framework mutation modification

impact, securing the best value across all groupings. The GOA method, with its mutation modification, achieves the highest result, with an impact value of 3,484 on high-dimensional unimodal functions. ACO emerges as the method with the most substantial impact, boasting an average modification value of 16 mutations with positive results across all dimensions and problem function types, totaling 0.55.

The ABC and CSO methods yield favorable results in dimensions 10 and 30 (high dimensions) for both unimodal and multimodal functions. On the other hand, PSO and GOA methods exhibit less favorable outcomes with the addition of mutations. PSO only demonstrates improved results on high dimensions with multimodal function mutations. Conversely, mutations applied to GOA show no positive effects and even exacerbate the performance of the basic GOA method, resulting in an average negative impact of mutations.

3.2 Framework and operator modification analysis

The impact of mutation modifications can be assessed through an analysis of the summary of impact values for the 5 basic methods, 4 dimensions, and 5 objective functions—essentially, the average impact value derived from 100 iterations. The summarized value encompasses the total sum, average, best, worst, and total modifications that yield positive values (n positive), as depicted in Table 7. Modification 2B stands out as the modification with the highest number of positive results, achieving success in 69 out of the 100 conducted tests. Conversely, the smallest sum and average values are produced by 2D modifications.

To analyze the influence of different frameworks and operators, the results from Table 7 are organized by framework and mutation operator. Figure 10 presents a graph illustrating the groupings of frameworks and operators based on the sum of positive results. The blue line represents the results from Table 7, while the yellow line represents the average value of the grouped operators (left picture) and frameworks (right picture).

From Figure 10, it becomes apparent that the grouping of frameworks has a more pronounced influence than that of the operators. In the grouping of operators, the average values appear consistent, whereas in the grouping of frameworks, the results exhibit variation. Modification of framework number 2 emerges with the best results, indicating that this modification has the most significant effect. Similarly, an analysis of modifications influencing the average impact value reveals that modifications with Framework 2 yield the best results based on Figure 10. The graphical pattern observed aligns with that depicted in Figure 9, which is based on operator modifications.

The study reveals that mutation modifications have a substantial positive impact on the Ant Colony Optimization (ACO) method, with 15 out of 16 modifications showing improved results. Notably, the ACO with the M-1C modification achieved the highest score of 19.70. Additionally, the Artificial Bee Colony (ABC) method with the M-1D modification exhibited significant positive effects. Among the tested frameworks, Framework number 2 emerged as the most effective, yielding the best results across various dimensions and objective functions. In high-dimensional unimodal functions, the Grasshopper Optimization Algorithm (GOA) with its mutation modification achieved the highest result. Both ABC and Cat Swarm Optimization (CSO) methods performed well in high dimensions (10 and 30) for both unimodal and multimodal functions, while the Particle Swarm Optimization (PSO) method showed improvement only in high dimensions with multimodal functions. Conversely, mutations generally worsened the performance of GOA, resulting in an average negative impact. The analysis also indicates that the grouping of frameworks has a more pronounced influence on performance than the grouping of operators, with Modification 2B standing out for its high number of positive results in 69 out of 100 tests. These findings highlight the enhanced optimization performance achieved through specific mutation modifications, emphasizing the importance of selecting appropriate frameworks and mutation techniques. The results provide valuable insights for optimizing swarm intelligence algorithms, suggesting that ACO and ABC methods are particularly suitable for problems requiring enhanced exploration, while GOA may require further refinement

for targeted applications in high-dimensional unimodal functions.

4 Conclusion

Based on the results obtained from all conducted trials in this study, it is deduced that mutation modification exerts the most substantial impact on optimization methods characterized by low complexity. Among these, the ACO, ABC, and PSO methods exhibit superior values compared to CSO and GOA. The application of T-Test analysis reveals that the ABC method demonstrates the most favorable effect in response to mutations.

A secondary conclusion drawn is that mutation modification exerts the most notable influence on multimodal problem functions and those with high dimensions. This is substantiated by the observation that mutation modification impedes the optimization methods from converging rapidly, thereby leading to more optimal results. In summary, this study affirms that the mutation modification yielding the most significant impact is the 2nd framework modification (Algorithm 2), also known as Random Parameter Mutation.

5 Future work

Expanding upon the insights gained from this study, several promising avenues for future research in optimizing swarm intelligence algorithms emerge. A pivotal direction involves advancing mutation techniques to fine-tune their efficacy within swarm intelligence frameworks. This could entail exploring adaptive mutation strategies that dynamically adjust mutation rates based on algorithmic performance feedback or hybrid approaches that integrate multiple mutation operators tailored to specific problem characteristics. Such endeavors aim to deepen the algorithms' capacity for efficient exploration and exploitation in complex optimization landscapes, thereby enhancing their robustness and adaptability.

Another critical area for exploration lies in extending the study to encompass multi-objective optimization scenarios. Investigating how mutation modifications influence the trade-offs between conflicting objectives in multi-dimensional optimization problems could pave the way for developing versatile multi-objective swarm intelligence algorithms. These algorithms would be capable of handling diverse decision-making challenges across domains such as engineering design, finance portfolio optimization, and logistics planning.

Furthermore, future research could focus on validating and refining optimized swarm intelligence algorithms through rigorous applications in real-world settings. By testing these algorithms against practical datasets and scenarios in sectors like healthcare resource allocation, industrial process optimization, and urban planning, researchers can assess their scalability, robustness, and applicability under real-world constraints and uncertainties.

Advancements could also involve exploring novel algorithmic enhancements beyond mutation. This includes investigating adaptive parameter tuning mechanisms that autonomously adjust algorithmic parameters in response to environmental changes or problem-specific dynamics. Additionally, the integration of advanced initialization techniques and meta-heuristic adaptations could further accelerate convergence rates and improve solution quality across diverse optimization tasks.

Comparative studies across mutation techniques and other state-of-the-art optimization algorithms present another promising avenue. Conducting comprehensive comparative analyses could elucidate the comparative advantages and limitations of swarm intelligence approaches relative to evolutionary strategies, gradient-based methods, and machine learning-driven optimizations in various complex optimization scenarios.

Moreover, theoretical investigations into the underlying mechanisms and mathematical foundations of mutation-based swarm intelligence optimizations could provide deeper insights into algorithmic behaviors and performance landscapes. These theoretical insights would not only enhance algorithm design principles but also contribute to advancing the theoretical foundations of meta-heuristic optimization methodologies.

Lastly, expanding the scope of benchmark functions and problem dimensions used in experimental validations would bolster the generalizability and robustness of findings. This expansion would encompass more intricate optimization landscapes and problem complexities, ensuring the thorough validation of mutation-driven enhancements in swarm intelligence algorithms across diverse and challenging optimization environments.

By pursuing these multifaceted research directions, future studies can significantly advance the field of swarm intelligence optimization. This advancement promises to elevate the practical applicability and effectiveness of swarm intelligence algorithms in tackling real-world optimization challenges across various scientific and industrial domains.

References

[1] Mykel J. Konchenderfer and T. A. Wheeler, Algorithms For Optimization. London: MIT Press, 2019.

[2] A. Naik and S. C. Satapathy, “A comparative study of social group optimization with a few recent optimization algorithms,” Complex & Intelligent Systems, vol. 7, no. 1, pp. 249–295, 2021, doi: 10.1007/s40747-020-00189-6.

[3] R. Hinterding, H. Gielewski, and T. Peachey, “The Nature of Mutation in Genetic Algorithms.,” Proceedings of the Sixth International Conference on Genetic Algorithms, pp. 65–72, 1995, [Online]. Available: http://pdf.aminer.org/000/310/686/the_nature_of_mutation_in_genetic_algorithms.pdf

[4] M. Sharma and J. K. Chhabra, “Sustainable automatic data clustering using hybrid PSO algorithm with

mutation," Sustainable Computing: Informatics and Systems, vol. 23, pp. 144–157, Sep. 2019, doi: 10.1016/j.suscom.2019.07.009.

[5] S. Rani, B. Suri, and R. Goyal, "On the effectiveness of using elitist genetic algorithm in mutation testing," Symmetry (Basel), vol. 11, no. 9, 2019, doi: 10.3390/sym11091145.

[6] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing mutation and crossover ratios for genetic algorithms-a review with a new dynamic approach," Information (Switzerland), vol. 10, no. 12, 2019, doi: 10.3390/info10120390.

[7] C. Audet and W. Hare, "Genetic Algorithms," Springer Series in Operations Research and Financial Engineering, pp. 57–73, 2017, doi: 10.1007/978-3-319-68913-5_4.

[8] T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, "Particle Swarm Optimization: A Comprehensive Survey," IEEE Access, vol. 10, pp. 10031–10061, 2022, doi: 10.1109/ACCESS.2022.3142859.

[9] E. H. Houssein, A. G. Gad, K. Hussain, and P. N. Suganthan, "Major Advances in Particle Swarm Optimization: Theory, Analysis, and Application," Swarm Evol Comput, vol. 63, Jun. 2021, doi: 10.1016/j.swevo.2021.100868.

[10] X. Wang, P. Henshaw, and D. S. K. Ting, "Exergoeconomic analysis for a thermoelectric generator using mutation particle swarm optimization (M-PSO)," Appl Energy, vol. 294, Jul. 2021, doi: 10.1016/j.apenergy.2021.116952.

[11] B. Jana, S. Mitra, and S. Acharyya, "Repository and Mutation based Particle Swarm Optimization (RMPSO): A new PSO variant applied to reconstruction of Gene Regulatory Network," Applied Soft Computing Journal, vol. 74, pp. 330–355, Jan. 2019, doi: 10.1016/j.asoc.2018.09.027.

[12] R. R. Ihsan, S. M. Almufti, B. M. S. Ormani, R. R. Asaad, and R. B. Marqas, "A Survey on Cat Swarm Optimization Algorithm," Asian Journal of Research in Computer Science, pp. 22–32, Jun. 2021, doi: 10.9734/ajrcos/2021/v10i230237.

[13] A. M. Ahmed, T. A. Rashid, and S. A. M. Saeed, "Cat Swarm Optimization Algorithm: A Survey and Performance Evaluation," Computational Intelligence and Neuroscience, vol. 2020. Hindawi Limited, 2020. doi: 10.1155/2020/4854895.

[14] L. Pappula and D. Ghosh, "Cat swarm optimization with normal mutation for fast convergence of multimodal functions," Applied Soft Computing Journal, vol. 66, pp. 473–491, May 2018, doi: 10.1016/j.asoc.2018.02.012.

[15] Y. Meraihi, A. B. Gabis, S. Mirjalili, and A. Ramdane-Cherif, "Grasshopper optimization algorithm: Theory, variants, and applications," IEEE Access, vol. 9, pp. 50001–50024, 2021, doi: 10.1109/ACCESS.2021.3067597.

[16] L. Abualigah and A. Diabat, "A comprehensive survey of the Grasshopper optimization algorithm: results, variants, and applications," Neural Computing and Applications, vol. 32, no. 19. Springer Science and Business Media Deutschland GmbH, pp. 15533–15556, Oct. 01, 2020. doi: 10.1007/s00521-020-04789-8.

[17] S. Zhao, P. Wang, A. A. Heidari, X. Zhao, C. Ma, and H. Chen, "An enhanced Cauchy mutation grasshopper optimization with trigonometric substitution: engineering design and feature selection," Eng Comput, Dec. 2021, doi: 10.1007/s00366-021-01448-x.

[18] S. A. A. Ghaleb, M. Mohamad, E. F. H. Syed Abdullah, and W. A. H. M. Ghanem, "Integrating mutation operator into grasshopper optimization algorithm for global optimization," Soft comput, vol. 25, no. 13, pp. 8281–8324, Jul. 2021, doi: 10.1007/s00500-021-05752-y.

[19] B. N. Silva and K. Han, "Mutation operator integrated ant colony optimization based domestic appliance scheduling for lucrative demand side management," Future Generation Computer Systems, vol. 100, pp. 557–568, 2019, doi: 10.1016/j.future.2019.05.052.

[20] J. H. Tam, Z. C. Ong, Z. Ismail, B. C. Ang, and S. Y. Khoo, "A new hybrid GA−ACO−PSO algorithm for solving various engineering design problems," Int J Comput Math, vol. 96, no. 5, pp. 883–919, 2019, doi: 10.1080/00207160.2018.1463438.

[21] D. M. Chitty, "Partial-ACO as a GA mutation operator applied to TSP instances," GECCO 2021 Companion - Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion, pp. 69–70, 2021, doi: 10.1145/3449726.3459424.

[22] H. Li and W. Li, "Enhanced artificial bee Colony algorithm and its application in multi-threshold image feature retrieval," Multimed Tools Appl, vol. 78, no. 7, pp. 8683–8698, 2019, doi: 10.1007/s11042-018-6066-6.

[23] F. Ye, Z. Zhou, H. Tian, Q. Sun, Y. Li, and T. Jiang, "Intelligent Anti-Jamming Decision Method Based on the Mutation Search Artificial Bee Colony Algorithm for Wireless Systems," 2019 USNC-URSI Radio Science Meeting (Joint with AP-S Symposium), USNC-URSI 2019 - Proceedings, pp. 27–28, 2019, doi: 10.1109/USNC-URSI.2019.8861785.

[24] G. E. Yuliastuti, A. M. Rizki, W. F. Mahmudy, and I. P. Tama, "Optimization of Multi-Product Aggregate Production Planning using Hybrid Simulated Annealing and Adaptive Genetic Algorithm," International Journal of Advanced Computer Science and Applications, vol. 10, no. 11, pp. 484–489, 2019, doi: 10.14569/IJACSA.2019.0101167.

[25] B. Jana, S. Mitra, and S. Acharyya, "Repository and Mutation based Particle Swarm Optimization (RMPSO): A new PSO variant applied to reconstruction of Gene Regulatory Network," Applied Soft Computing Journal, vol. 74, pp. 330–355, Jan. 2019, doi: 10.1016/j.asoc.2018.09.027

APPENDIX A. Result of unimodal objective function

		Rosenbrock				De Jong			
		2-D	5-D	10-D	30-D	2-D	5-D	10-D	30-D
ABC	Mean	-2,88	-0,86	0,48	0,33	-1,89	-1,16	-0,65	0,39
	Sum	-46,11	-13,80	7,62	5,26	-30,19	-18,53	-10,44	6,29
	Worst	-13,43	-5,23	0,06	-0,47	-8,59	-12,56	-4,71	-0,05
	Best	0,61	0,84	0,97	0,99	0,77	1,00	0,98	1,00
ACO	Mean	0,20	**0,85**	0,53	0,55	0,68	0,37	0,60	0,52
	Sum	3,14	13,65	8,53	8,79	10,83	5,93	9,66	8,24
	Worst	-1,96	0,55	-0,08	-0,03	-0,47	-0,13	0,09	0,04
	Best	0,98	0,98	0,97	0,99	1,00	1,00	1,00	1,00
CSO	Mean	-21,22	-0,96	-0,06	-0,11	0,03	-4,06	-0,02	0,23
	Sum	-339,45	-15,32	-0,89	-1,75	0,43	-64,96	-0,26	3,66
	Worst	-52,09	-5,66	-0,87	-0,46	-2,77	-6,98	-0,64	0,05
	Best	-0,23	0,27	0,85	0,27	0,88	-0,63	0,32	0,34
GOA	Mean	0,29	-0,01	-0,26	-8,50	-1,17	-0,57	0,01	-0,01
	Sum	4,66	-0,22	-4,23	-135,93	-18,78	-9,06	0,15	-0,23
	Worst	0,00	-0,23	-0,52	-37,61	-4,42	-2,09	-0,67	-0,13
	Best	0,63	0,16	0,02	13,30	0,93	0,66	0,48	0,15
PSO	Mean	0,59	-0,89	0,03	0,10	0,09	-0,39	0,17	-0,02
	Sum	9,43	-14,24	0,41	1,66	1,49	-6,17	2,65	-0,36
	Worst	-0,14	-2,35	-0,80	-0,31	-2,19	-1,88	-0,60	-0,15
	Best	1,00	0,05	0,84	0,49	0,99	0,46	0,63	0,32

APPENDIX B. Result of multimodal objective function

		Rastrigin				Griewangk				Ackley			
		2-D	5-D	10-D	30-D	2-D	5-D	10-D	30-D	2-D	5-D	10-D	30-D
ABC	Mean	0,03	0,01	0,28	0,19	-1,26	-0,05	0,16	0,28	-0,23	0,43	0,19	0,25
	Sum	0,54	0,19	4,45	3,04	-20,08	-0,79	2,64	4,56	-3,71	6,83	3,08	3,95
	Worst	-1,09	-0,81	-0,20	-0,29	-3,61	-1,43	-0,32	-0,15	-1,39	-0,05	-0,05	-0,01
	Best	0,99	1,00	0,94	0,92	0,75	0,78	0,96	1,00	0,93	0,94	0,86	0,99
ACO	Mean	0,16	0,38	0,37	0,33	0,16	0,38	0,37	0,33	0,61	0,37	0,40	0,36
	Sum	2,57	6,06	5,84	5,31	2,57	6,06	5,84	5,31	9,68	5,93	6,39	5,76
	Worst	-2,05	-0,31	0,00	0,00	-2,05	-0,31	0,00	0,00	-0,01	-0,13	0,00	0,00
	Best	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
CSO	Mean	-0,23	-0,37	0,11	-0,01	0,32	-0,19	0,06	0,08	-1,31	-0,48	0,03	0,01
	Sum	-3,75	-5,84	1,76	-0,12	5,10	-3,05	1,02	1,20	-20,99	-7,61	0,49	0,21
	Worst	-5,26	-1,06	-0,08	-0,10	-0,51	-1,33	-0,39	-0,15	-6,31	-1,10	-0,08	-0,02
	Best	0,88	0,50	0,44	0,13	0,96	0,55	0,38	0,33	0,61	0,65	0,13	0,06
GOA	Mean	-0,26	0,06	0,06	-0,11	-2,29	0,10	0,05	-0,10	-2,29	0,10	0,05	-0,10
	Sum	-4,19	0,93	0,93	-1,70	-36,63	1,57	0,87	-1,61	-36,63	1,57	0,87	-1,61
	Worst	-2,41	-0,32	-0,07	-0,21	-3,30	-0,33	-0,17	-0,37	-3,30	-0,33	-0,17	-0,37
	Best	0,99	0,47	0,34	0,06	-0,62	0,47	0,28	0,17	-0,62	0,47	0,28	0,17
PSO	Mean	0,25	0,21	0,16	0,06	-1,78	0,03	-0,39	-0,01	-2,31	-0,09	0,07	-0,03
	Sum	4,07	3,39	2,63	0,92	-28,52	0,53	-6,29	-0,20	-37,03	-1,50	1,06	-0,56
	Worst	-0,38	-0,04	-0,01	-0,02	-3,96	-0,47	-1,14	-0,26	-4,33	-0,32	-0,02	-0,07
	Best	0,86	0,55	0,28	0,15	-0,23	0,54	0,36	0,29	0,64	0,20	0,16	0,01