Sigmoid-Function-Based Adaptive Pelican Optimization Algorithm for Global Optimization

A. F. Seini Yussif[1*], S. Adjei[2], B. E. Wilson[3]
[1*]Department of Electrical and Electronics Engineering, University for Development Studies, Tamale, Ghana
[2]Department of Electrical and Electronics Engineering, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana
[3]Department of Computer and Electrical Engineering, University of Energy and Natural Resources, Sunyani, Ghana
[*]corresponding author's email: seini.coe@gmail.com

Abstract – *This paper introduces the Sigmoid-function-based Adaptive Pelican Optimization Algorithm (MPOA), an enhanced version of the traditional Pelican Optimization Algorithm (POA) aimed at improving the POA's performance. Inspired by the hunting behavior of pelicans, the POA features two main strategies: the Exploration phase and the Exploitation phase. The Exploration phase involves searching new areas within the solution space, while the Exploitation phase focuses on refining the optimal solution space to achieve convergence. However, the Exploitation phase is inefficient, leading to slower convergence rates when striving for a global optimum. The MPOA incorporates an adaptive inertia weight mechanism that leverages the sigmoid function to balance exploration and exploitation throughout the optimization process. This adaptive approach ensures an efficient transition between searching for new solution areas and refining existing ones, thereby enhancing the overall optimization process. The algorithm was tested using a set of widely recognized standard benchmark functions to assess its performance. The results demonstrated that the MPOA significantly improved both convergence speed and solution quality compared to the original POA. Additionally, the MPOA outperformed other traditional optimization algorithms, such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA), in terms of achieving better optimization results. It specifically outperformed the others on 22 out of the 23 functions representing a 95.65% success rate. These findings suggest that the proposed MPOA provides an efficient optimization approach, leading to faster convergence and higher-quality solutions.*

Keywords*: Algorithm, Adaptive, Metaheuristic, Nature-Inspired, Pelican Optimization Algorithm*

Article History
Received 24 July 2024
Received in revised form 20 August 2024
Accepted 28 August 2024

I. Introduction

Optimization algorithms are vital tools in engineering and applied sciences for addressing a wide range of complex optimization problems [1]. These algorithms strive to find the optimal solution from an extensive set of possibilities [2]. Metaheuristic optimization has been utilized in many optimization problems. Metaheuristic approaches, which serve as strategies for global optimization, emulate natural processes or social behaviors [3]-[4].

Numerous metaheuristic methods have been inspired by nature. For instance, the binary Grey-Wolf Optimization (bGWO) is inspired by the fascinating behaviors of grey wolves [5], particularly their social hierarchy and hunting techniques, the Butterfly Optimization Algorithm (BOA) draws inspiration from the remarkable food foraging behavior of butterflies [6], Ant-Lion Optimization (ALO) draws inspiration from the ingenious hunting strategy of antlion larvae [7], where these larvae create sand pits to trap ants, demonstrating a sophisticated method of predation. Artificial-Bee Colony (ABC) is based on the foraging behavior of honey bees, which efficiently search for food sources using a complex communication system involving the waggle dance [8]. Satin-Bowerbird Optimization (SBO) mimics the unique mating behavior of male Satin Bowerbirds, known for building elaborate structures, or bowers, to attract females [9]. The Crow-Search Algorithm (CSA) is inspired by the intelligent and social characteristics of crows, particularly their remarkable memory and ability to use tools [10]. Pelican Optimization Algorithm (POA) is inspired by the foraging and hunting strategies of pelicans, especially their cooperative behavior and unique methods of catching fish,

such as synchronized diving [5]. Finally, Particle Swarm Optimization (PSO) is derived from the social behavior observed in bird flocking and fish schooling, where individuals follow simple rules based on the behavior of their neighbors to achieve collective movement [2].

Among these various optimization algorithms, the Pelican Optimization Algorithm (POA) stands out for its simplicity and effectiveness in solving optimization problems [11]. Inspired by the collaboration and competitive hunting behavior of pelicans to catch fish, POA simulates the cooperative and competitive strategies pelicans use to catch prey, translating this behavior into a metaheuristic optimization framework [12]. To optimize their catch, Pelicans use special hunting techniques that involve individual and group strategies. For fish hunting, Pelicans form semicircular formations which compels the fish to move towards the shore making them easier to catch. This coordinated effort improves the group's success rate, while individual pelicans compete for the best catch. This combination of cooperation and competition is key to the success of POA, which uses natural behaviors to find optimal solutions in complex search spaces.

In the context of POA, several studies have highlighted its potential and limitations [16]. The original POA inspired by the collaborative and competitive hunting strategies of pelicans is effective in the exploration phase, where pelicans (candidate solutions) spread out and search the solution space [13]. However, the exploitation phase, which focuses on refining and converging to the optimal solution, is less efficient, leading to slower convergence rates in achieving high-quality solutions [14]. This aspect shows the need for researchers to enhance the algorithm's performance by balancing the exploration and exploitation phases effectively.

Similar drawbacks have been addressed in other algorithms using adaptive weights mechanisms. For instance, adaptive inertia weights in PSO and adaptive mutation rates in GAs have shown promising results [15]. However, there has been limited focus on enhancing the POA's exploitation phase through adaptive mechanisms. This study aims to fill this gap by introducing a sigmoid-function-based adaptive inertia weight in the POA. This creates the Sigmoid-function-based Adaptive Pelican Optimization Algorithm (MPOA), which uses a dynamic approach to address these challenges and improve the overall performance [16].

The primary aim of this research is to assess the impact of the Sigmoid-function-based Adaptive strategy on the POA's performance using well-known twenty-three standard benchmark test functions [12]. The study seeks to prove that incorporating an adaptive sigmoid-function-based inertia weight can enhance the POA's convergence speed and solution quality in solving complex optimization problems.

The structure of the rest of the paper is as follows: Section II explains the original Pelican Optimization Algorithm (POA) and its key components. Section III introduces the new Sigmoid-function-based Adaptive Pelican Optimization Algorithm (MPOA) and the reasoning behind its development. Section IV outlines the experimental setup and the testing process for the MPOA. Section V presents the results and discusses the outcomes. Finally, Section VI concludes the paper and suggests future research directions.

II. The Original Pelican Optimization Algorithm (POA)

The Pelican Optimization Algorithm (POA) is a metaheuristic optimizer inspired by pelican behavior. It is based on swarm intelligence, where the algorithm mimics the collaborative and competitive foraging behaviors of pelicans. In this system, each agent within the swarm shares communal knowledge to enhance their search efficiency and effectiveness. By emulating pelicans' natural strategies, POA optimizes the search process, making it a powerful tool for solving complex optimization problems. This approach leverages the strength of collective intelligence and adaptive behavior to achieve superior performance in various optimization tasks. Similar to Particle Swarm Optimization (PSO) [15], which models birds flying together in search of food, POA utilizes the group dynamics of pelicans hunting for prey to solve complex optimization problems effectively [12].

The Pelican Optimization Algorithm (POA) utilizes pelicans as primary elements in its population, with each pelican representing a potential solution. These pelicans propose values for the optimization variables based on their positions within the search space [17]. Initially, the pelicans are assigned random values within the problem's defined lower and upper bounds, as outlined in (1). This random assignment ensures a diverse set of solutions from the start, allowing the algorithm to effectively explore the search space. As the algorithm progresses, the pelicans continuously adjust their values in pursuit of the optimal solution. They do this by evaluating their current positions and updating them according to the optimization criteria. This iterative process enables the pelicans to gradually converge toward the best possible solution by refining their positions based on the feedback from the optimization process. Through this method, POA balances exploration and exploitation, leveraging the pelicans' movements to efficiently navigate the search space and identify optimal solutions.

$$X_{ij} = l_j + \left(u_j - l_j\right).rand\,, \quad i = 1,2,\dots,N \quad j = 1,2,\dots,m \tag{1}$$

where; x_{ij} represents the value of the jth variable specified by the ith candidate solution. N denotes the population size, while m is the number of problem variables. l_j and u_j represent the lower and upper bounds of the search

interval respectively, and *rand* is a randomly determined value within the interval [0, 1].

The POA algorithm emulates pelicans' hunting behavior to update candidate solutions. It replicates their strategies through a two-stage process, where the pelicans' attack and prey capture methods are used to refine and optimize potential solutions, improving the algorithm's effectiveness and efficiency. These two-stage processes are the Exploration Phase and the Exploitation Phase for searching for new possible solutions and refining already gotten solutions respectively for efficient execution. The two stages are further elaborated as follows:

Exploration Phase:

During the first phase, pelicans advance towards their prey once they have detected its position. The unpredictable nature of the prey's location significantly enhances the exploration capabilities of the Pelican Optimization Algorithm (POA) [12]. By emulating the pelican's strategy in approaching its prey, an exploration update operator is formulated. This operator, as detailed in (2), leverages the pelican's movement strategy to improve the algorithm's ability to explore the solution space more effectively. This ensures a more thorough search, potentially leading to better optimization results.

The randomness in the prey's positioning introduces a crucial element of variability, which helps in avoiding local optima. This variability is essential for achieving a comprehensive exploration of the solution space, thereby enhancing the algorithm's overall performance. By incorporating the pelican's natural hunting behavior, the algorithm gains a robust mechanism for exploration, which is key to its effectiveness in finding optimal solutions.

$$x_{ij}^{P1} = \begin{cases} x_{ij} + rand.\left(p_j - I.x_{ij}\right), & F_p < F_i \\ x_{ij} + rand.\left(x_{ij} - p_j\right), & else \end{cases} \quad (2)$$

where; x_{ij}^{P1} represents the updated status of the *i*th pelican in the *j*th dimension during the exploration phase. I is a random integer of either 1 or 2, p_j is the location of the prey in the *j*th dimension, and F_p is its objective function value. The exploration phase keeps a record of the best pelican using (3) as an updated technique.

$$X_i = \begin{cases} X_i^{P1}, & F_i^{P1} < F_i \\ X_i, & else \end{cases} \quad (3)$$

where; X_i^{P1} represents the new status of the pelican in the *i*th dimension, and F_i^{P1} represents its objective function value based on the exploration phase. X_i is the present status of the pelican in the *i*th dimension, while F_i denotes its objective function value.

Exploitation Phase:

Pelicans enhance their fishing efficiency by spreading their wings on the water's surface during the exploitation phase, lifting fish into their throat pouches. This technique boosts the algorithm's exploitation capability, leading to superior solutions within the hunting zone [12]. The mathematical modeling of the exploitation phase update operator, as shown in (4), is inspired by this pelican behavior of spreading wings on the water surface to extract fish. This analogy illustrates how the algorithm mimics pelicans' method of increasing their catch rate, ultimately improving convergence to optimal solutions. The pelicans' wing-spreading action serves as a biological metaphor for the algorithm's strategy, emphasizing the importance of effective exploitation in achieving better results.

By drawing from the pelicans' natural hunting techniques, the algorithm can more efficiently navigate the solution space, ensuring a higher likelihood of identifying the best possible outcomes.

$$x_{ij}^{P2} = x_{ij} + R.\left(1 - \frac{t}{T}\right).(2.rand - 1).x_{ij} \quad (4)$$

where; x_{ij}^{P2} represents the new status of the *i*th pelican in the *j*th dimension, R is a constant with a value of 0.2, and t and T represent the iteration count and the maximum number of iterations. The exploitation process has the same updated mechanism as the exploration phase as shown in (5).

$$X_i = \begin{cases} X_i^{P2}, & F_i^{P2} < F_i \\ X_i, & else \end{cases} \quad (5)$$

where; X_i^{P2} is the new status of the pelican in the *i*th dimension, and F_i^{P2} is its objective function value based on the exploitation phase.

The implementation process of the POA is systematically presented in the flowchart in Fig. 1.

III. Proposed Modification with Sigmoid-function-based Adaptive (SA) Inertia Weight (ω)

Like most optimization algorithms, no single algorithm is capable of effectively solving all optimization problems. Each algorithm possesses distinct weaknesses or limitations that necessitate various modifications or enhancements to boost its performance in solving optimization challenges. POA is among these optimization algorithms and is recognized for its strong exploration capabilities. Nevertheless, the POA's exploitation phase struggles with a slow search ability when it comes to achieving global optimal solutions [12].

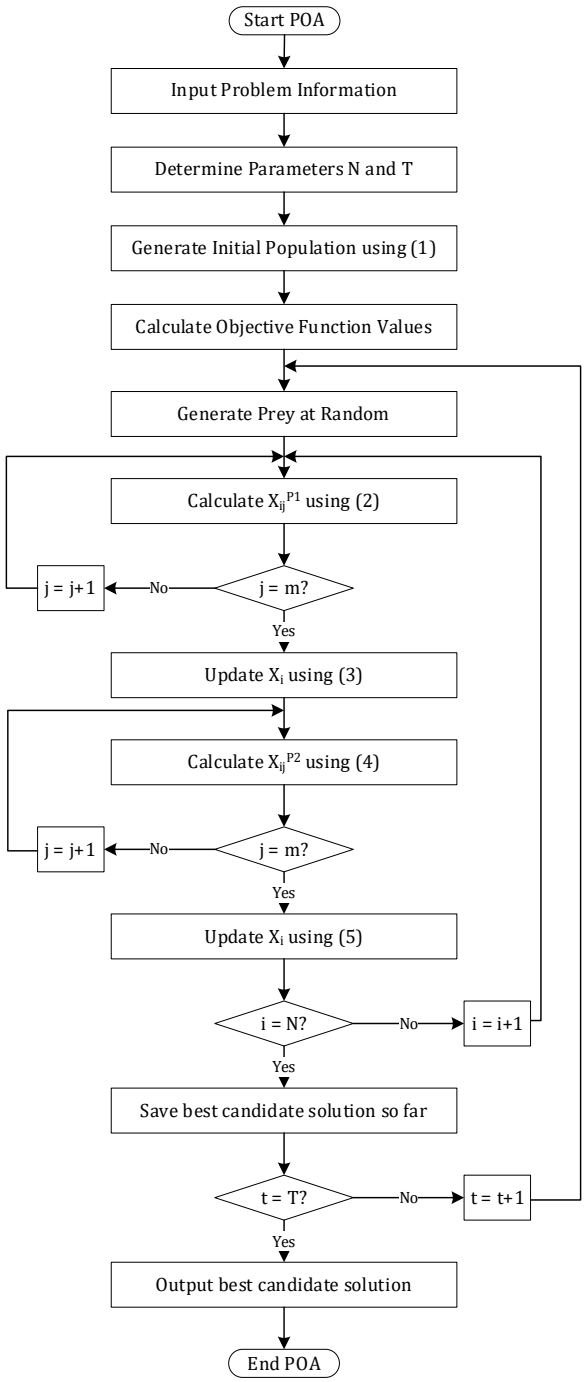


Fig. 1. Implementation Flowchart of POA

The update function of POA is highly dependent on several factors: the iteration count, the initial value assigned to the pelican, and the maximum number of iterations. This dependency can lead to inefficiencies in the exploitation process for certain optimization applications [5]. As a result, while POA demonstrates notable potential in exploring the solution space, its ability to refine and exploit these solutions to achieve the best possible outcome is hindered. Moreover, the limited nature of the POA to adapt to changing conditions makes it inefficient to employ for different purposes.

Addressing these specific weaknesses through targeted modifications could significantly enhance the overall efficiency and effectiveness of the POA, enabling it to perform better across a wider range of optimization problems. Such enhancements not only aim to improve the convergence speed but also seek to enhance the overall quality of solutions obtained. Therefore, continuous efforts to refine and improve the POA are essential for maximizing its utility in diverse optimization problems. In particular, incorporating advanced adaptive mechanisms can provide the necessary flexibility and robustness required for complex optimization tasks.

In this work, a sigmoid-function-based adaptive inertia weight (ω) [15] is introduced in the exploitation update phase to augment the poor exploitation capability of the POA as shown in Eqn. (6). This innovative approach leverages the mathematical properties of the sigmoid function to dynamically adjust the inertia weight, facilitating a more balanced and effective search strategy throughout the optimization process.

$$x_{ij}^{P2} = \omega x_{ij} + R.\left(1 - \frac{t}{T}\right).(2.rand - 1).x_{ij} \qquad (6)$$

where, ω represents the value of the inertia weight calculated using the sigmoid function expressed in Eqn. (7) below.

$$\omega_k = \omega_{max} - \frac{(\omega_{max} - \omega_{min})}{\left(1 + e^{3.4 - 0.7 \times Iter)}\right)} \qquad (7)$$

where: ω_k is the inertia weight at iteration k, ω_{min} and ω_{max} are the minimum and maximum range of the inertia weights, respectively. Iter represents the iterations counter.

The implementation of the proposed sigmoid-function-based adaptive pelican optimization algorithm in this work is by modifying (4) at the exploitation phase of the original POA into (6), and then retaining the rest of the implementation as shown in Fig. 2.

Testing Proposed MPOA Algorithm

To establish the improvement of the proposed modification on the POA, it has been tested on the same standard benchmark test functions used in reference [12], and the simulation results compared to those of the original POA, PSO, and GA.

Using MATLAB R2018a software, the proposed MPOA was coded and simulation tests on the same twenty-three (23) benchmark functions used in the original POA [11] with parameters as in Table I. Refer to Appendix A in [11] for the details of the benchmark functions. Thirty (30) simulations were repeated on each benchmark function with a thousand iterations each. The performances were compared with those of the original

POA, PSO, and GA reported in the literature [11] based on four statistical indicators; the mean, the median, the standard deviation, and the best optimal solution. The outcome is reported in the next section.

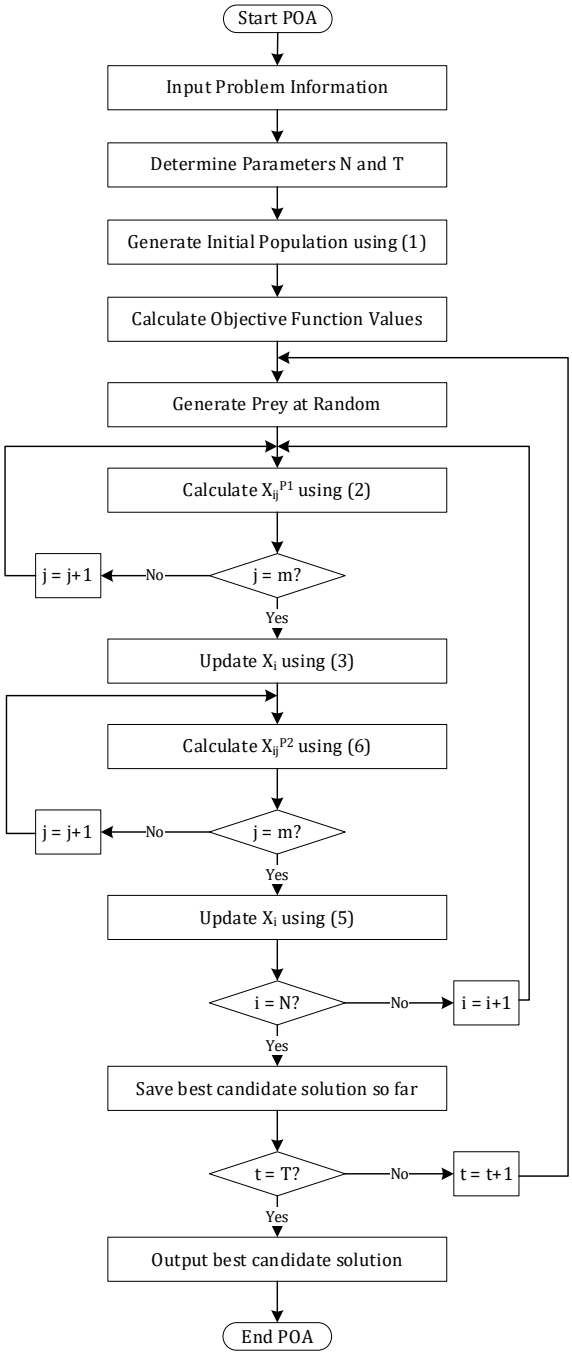


Fig. 2. Implementation Flowchart of MPOA

TABLE I
PARAMETERS SETTINGS FOR SIMULATION

Parameter	Value
Population Size	30
Maximum Iteration	1000
ω_{min}	0.1
ω_{max}	1.0

IV. Results and Discussion

This section details the simulation results comparing the proposed modified Pelican Optimization Algorithm (MPOA) with the original Pelican Optimization Algorithm (POA), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA). The performance metrics are summarized in Table II, where the best outcomes among the four algorithms are highlighted in bold.

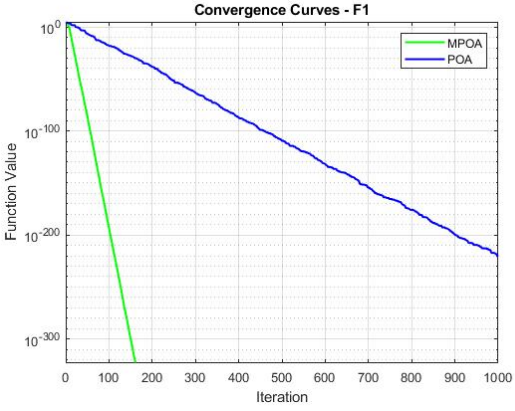
From the simulation results in Table II, it is evident that the MPOA demonstrates superior performance on nearly all of the 23 benchmark functions. However, there is an exception with function F7, where the original POA outperforms the other algorithms regarding mean value, median value, and standard deviation. Despite this, the MPOA still managed to produce the best single solution for the F7 function.

These findings suggest that the MPOA generally enhances performance across a wide range of benchmark functions compared to the original POA, PSO, and GA. The improved performance of the MPOA is particularly notable in most cases, although it is important to acknowledge that there are instances, such as with the F7 function, where the original POA may still hold an advantage in terms of consistency in mean, median, and standard deviation values. Nonetheless, the MPOA's ability to generate the best individual solution for F7 indicates its potential for optimization tasks.

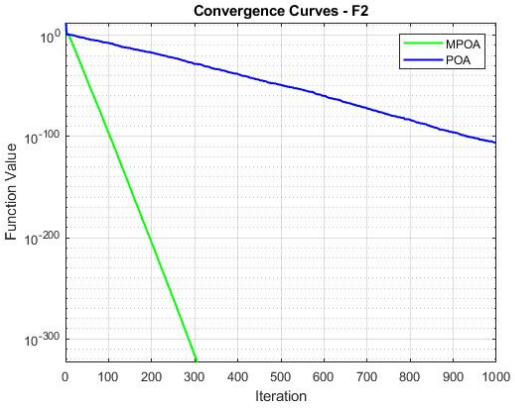
Specifically, the Modified Pelican Optimization Algorithm (MPOA) surpassed the original Pelican Optimization Algorithm (POA), Particle Swarm Optimization (PSO), and Genetic Algorithm (GA) on benchmark functions F1, F2, F3, F4, F5, F8, F12, F13, and F17, totaling 9 out of the 23 functions tested. Additionally, the MPOA delivered competitive results on F6, F9, F10, F11, F14, F15, F16, F18, F19, F20, F21, F22, and F23 when compared to the original POA, but it still outperformed PSO and GA on these functions. This accounts for 13 out of the 23 functions considered.

Overall, the MPOA showed excellent performance on 22 of the 23 benchmark functions, which translates to a 95.65% success rate relative to the other algorithms. These results indicate that the MPOA has significantly improved performance in optimization tasks when compared to the original POA, PSO, and GA.

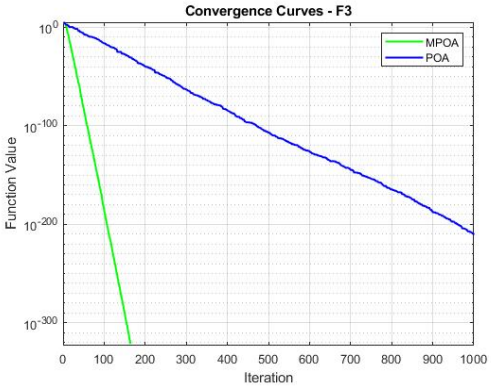
To further illustrate this, the convergence behaviors of the proposed MPOA and the original POA are compared across the 23 benchmark functions in the following Fig. 3 for F1 to F23. The figures highlight how the MPOA not only reaches optimal solutions more efficiently but also maintains a high level of consistency across different types of optimization problems. This comprehensive analysis confirms that the MPOA is a robust and effective algorithm, showcasing its enhanced capability in handling diverse and complex optimization tasks.



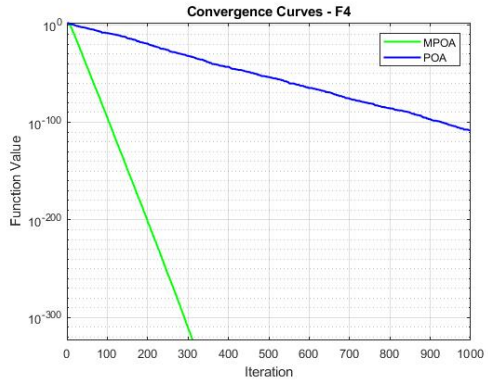
(a) Convergence behaviors for F1



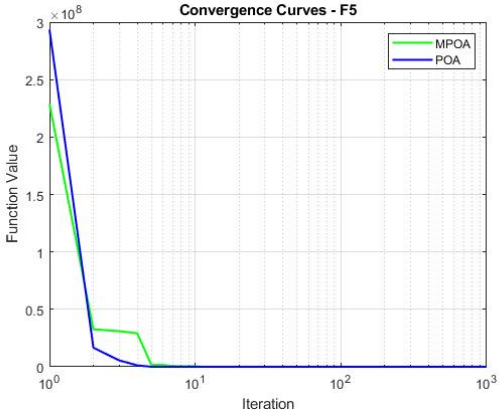
(b) Convergence behaviors for F2



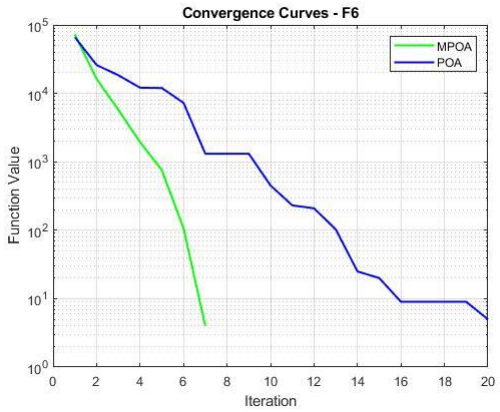
(c) Convergence behaviors for F3



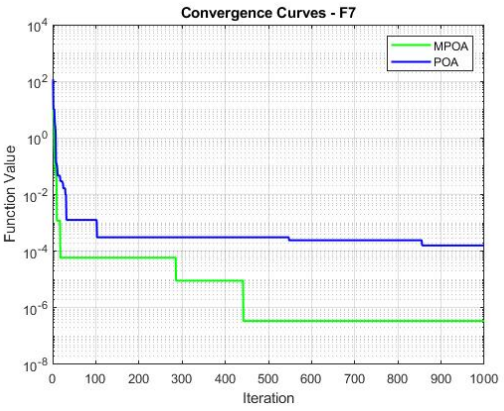
(d) Convergence behaviors for F4



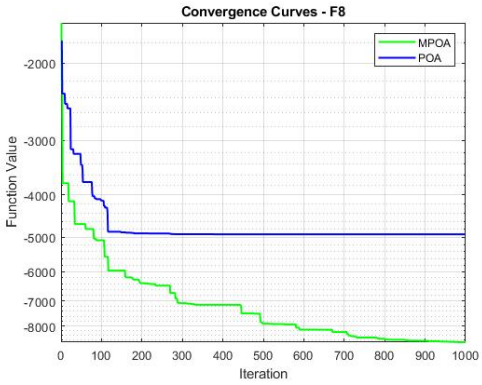
(e) Convergence behaviors for F5



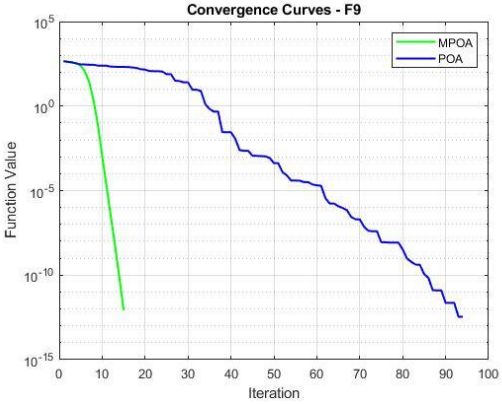
(f) Convergence behaviors for F6



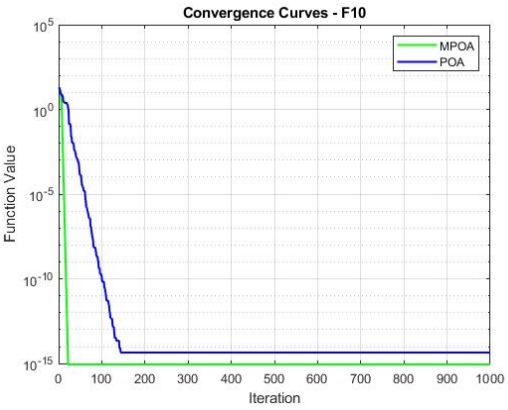
(g) Convergence behaviors for F7



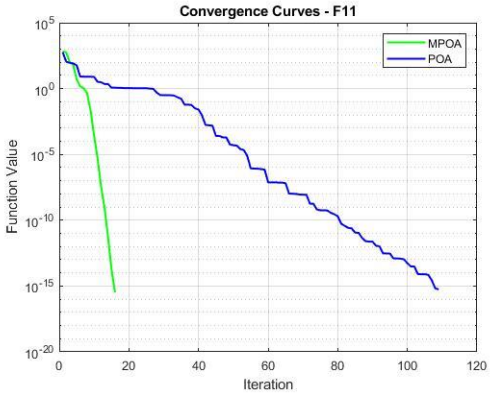
(h) Convergence behaviors for F8



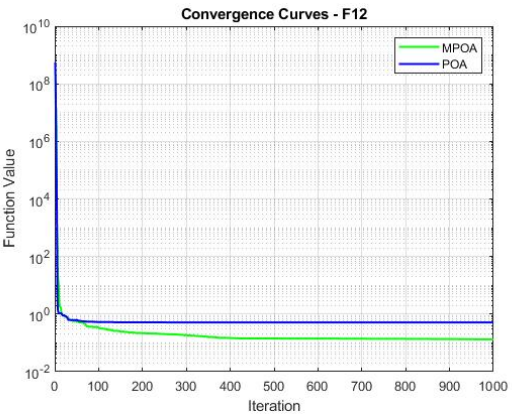
(i) Convergence behaviors for F9



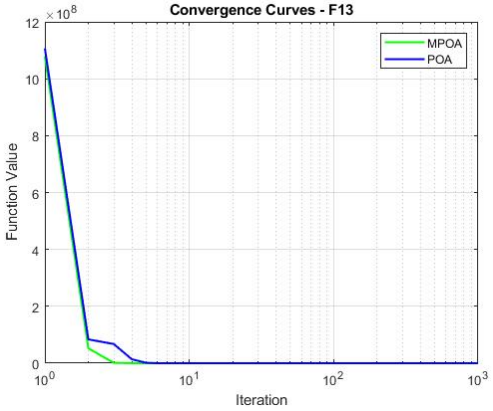
(j) Convergence behaviors for F10



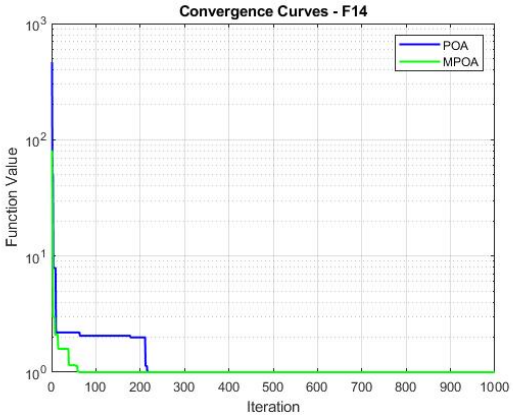
(k) Convergence behaviors for F11



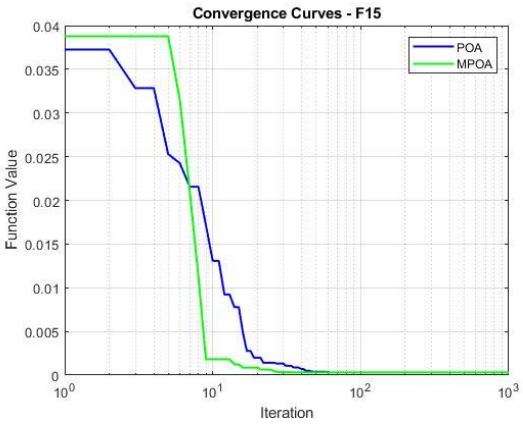
(l) Convergence behaviors for F12



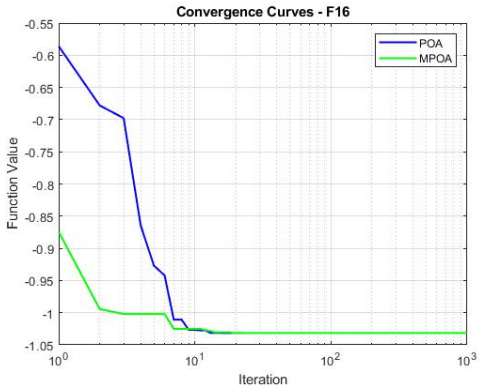
(m) Convergence behaviors for F13



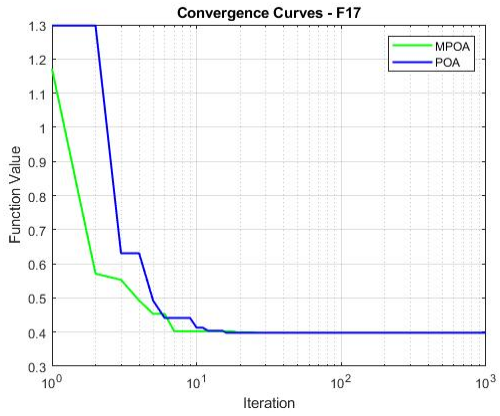
(n) Convergence behaviors for F14



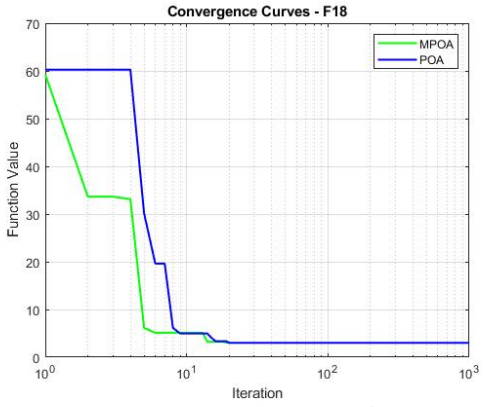
(o) Convergence behaviors for F15



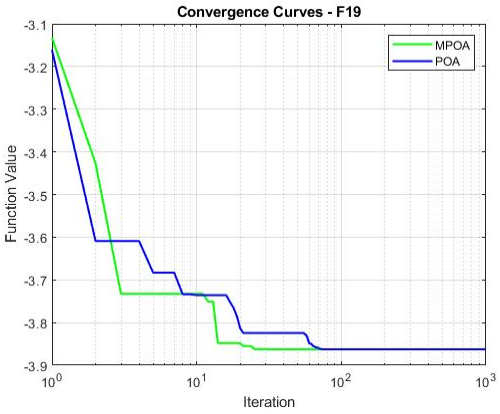
(p) Convergence behaviors for F16



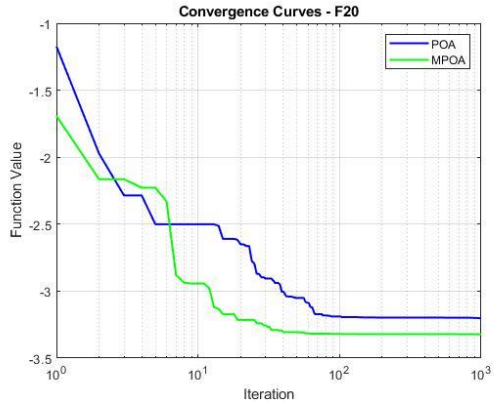
(q) Convergence behaviors for F17



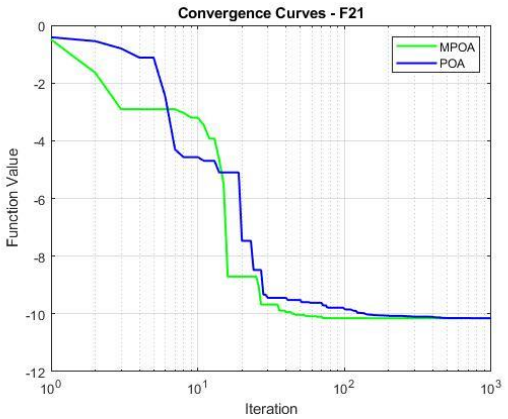
(r) Convergence behaviors for F18



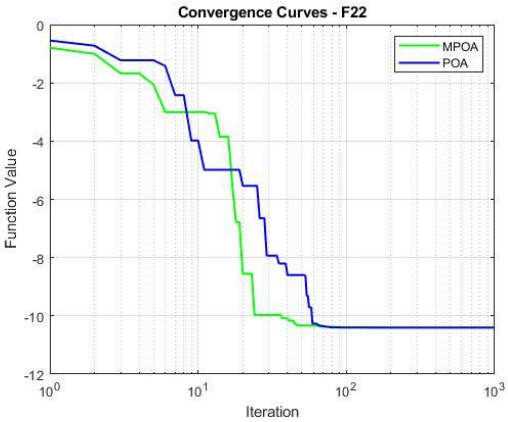
(s) Convergence behaviors for F19



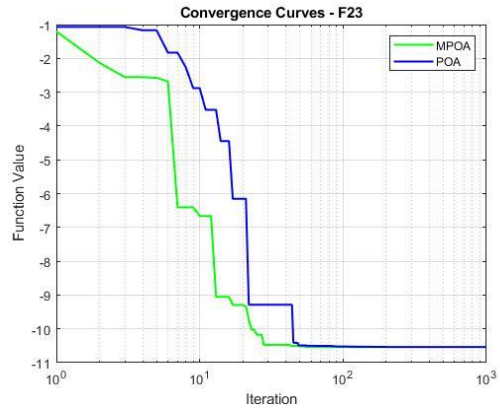
(t) Convergence behaviors for F20



(u) Convergence behaviors for F21



(v) Convergence behaviors for F22



(w) Convergence behaviors for F23

Fig. 3. Comparison of convergence behaviours between MPOA and POA for 23 benchmark functions

The convergence characteristics depicted in Fig. 3 for F1 to F23 shows the performance of MPOA and POA. The green line represents the convergence behavior of the MPOA across different test functions, while the blue line depicts the original POA's performance. The MPOA exhibits superior convergence compared to the POA, underscoring the enhancements achieved through the proposed modifications. This improved performance is consistently observed across all test functions, affirming the effectiveness of the modifications made to the original POA. Thus, the MPOA's enhanced convergence characteristics highlight the significant advancements introduced by the proposed changes.

The MPOA showed significantly faster and more efficient convergence to optimal solutions than the original POA on test functions F1, F2, F3, F4, F6, F7, F8, F9, and F11, with a markedly greater difference. This rapid convergence is crucial for practical applications requiring high computational efficiency, such as automated control systems. For functions F10, F12, and F20, the MPOA achieved substantially better convergence performance compared to the POA, resulting in superior solutions, although the differences were not as pronounced as in the earlier cases. In the remaining scenarios, the MPOA and POA exhibited very competitive convergence characteristics, with the MPOA narrowly outperforming the POA, leading to better final convergence solutions. These results highlight the MPOA's enhanced efficiency and effectiveness, making it a significant improvement over the original POA for applications demanding rapid and reliable optimization.

The consistent achievement of better or competitive convergence by the MPOA underscores its potential for practical implementations where optimization speed and solution quality are critical. Overall, the MPOA's ability to quickly converge to superior solutions emphasizes its value in real-world applications requiring high computational time efficiency.

The findings from the simulations and comprehensive statistical analysis strongly validate that incorporating the sigmoid-function-based adaptive inertia weight has substantially enhanced the performance of the POA. This innovative adaptive mechanism empowers the algorithm to dynamically and intelligently modify its search strategy in real-time, leading to a remarkable improvement in both the speed of convergence and the quality of the solutions obtained. By allowing for this dynamic adjustment, the algorithm becomes more efficient and effective, optimizing its search process and yielding superior results in a variety of scenarios. This enhancement demonstrates a profound advancement in the POA's capabilities, making it a robust and powerful tool for complex optimization tasks. The substantial improvements observed underscore the critical role of adaptive inertia weight in advancing state-of-the-art optimization algorithms, highlighting its potential for widespread application in fields that require high precision and efficiency.

V. Conclusion

This paper presents the Sigmoid-function-based Adaptive Pelican Optimization Algorithm (MPOA), an improved version of the traditional Pelican Optimization Algorithm (POA) designed to enhance its performance. The POA includes two main strategies: the Exploration phase and the Exploitation phase. The Exploration phase involves searching new areas within the solution space, while the Exploitation phase focuses on refining the optimal solution space to achieve convergence. However, the Exploitation phase tends to be less efficient, resulting in slower convergence rates when seeking a global optimum. The MPOA incorporates an adaptive inertia weight mechanism that uses the sigmoid function to dynamically balance exploration and exploitation throughout the optimization process to address this issue. This adaptive approach ensures a more efficient transition between exploring new solution areas and refining existing ones, thereby enhancing the overall optimization process. The algorithm was tested using a set of widely recognized standard benchmark functions to evaluate its performance. The results on the 23 functions showed that the MPOA significantly improved both convergence speed and solution quality compared to the original POA by producing good performance on 22 out of the 23 representing 95.65%. Additionally, the MPOA outperformed other traditional optimization algorithms, such as Particle Swarm Optimization (PSO) and Genetic Algorithms (GA), in terms of achieving superior optimization results.

This demonstrates the MPOA’s capability to handle a wide range of optimization challenges effectively. These findings indicate that the adaptive mechanism introduced in the MPOA provides a more balanced and efficient approach to optimization, leading to faster convergence and higher-quality solutions. Overall, the development of the MPOA represents a significant advancement in optimization algorithms, offering an effective tool for tackling complex optimization problems with enhanced efficiency. The successful application of the MPOA to standard benchmark functions highlights its potential for broader applications in various optimization scenarios such as the optimal power flow problem.

TABLE II
RESULTS COMPARISON OF BENCHMARK FUNCTIONS

FUNCTION	STATISTICAL INDICATORS	GA	PSO	POA	MPOA
F1 Sphere	MEAN	11.6208	4.1728E-04	2.87E-258	0.000E+00
	MEDIAN	11.04546	9.920E-7	8.2E-248	0.000E+00
	STD	2.6142E-11	3.6142E-21	4.51E-514	0.000E+00
	BEST	5.593489	2.00E-10	7.61E-264	0.000E+00
F2 Schwefel 2.22	MEAN	4.6942	0.3114	1.43E-128	0.000E+00
	MEDIAN	2.463873	0.130114	7.1E-123	0.000E+00
	STD	5.4318E-14	4.4667E-16	2.90E-129	0.000E+00
	BEST	1.591137	0.001741	2.61E-131	0.000E+00
F3 Schwefel 1.2	MEAN	1361.2743	588.3012	1.88E-256	0.000E+00
	MEDIAN	1510.715	54.15445	8.2E-244	0.000E+00
	STD	6.6096E-12	9.7117E-12	5.16E-614	0.000E+00
	BEST	1014.689	1.614937	7.36E-262	0.000E+00
F4 Schwefel 2.21	MEAN	2.0396	4.3693	2.36E-133	0.000E+00
	MEDIAN	2.09854	3.260672	2.8E-123	0.000E+00
	STD	4.3321E-14	4.2019E-15	8.37E-134	0.000E+00
	BEST	1.389849	1.60441	6.08E-138	0.000E+00
F5 Rosenbrock	MEAN	308.4196	50.5412	2.71253E+01	1.5585E-03
	MEDIAN	279.5174	28.69298	2.8707E+01	6.1201E-05
	STD	3.0412E-12	1.8529E-13	1.91E-15	5.4197E-03
	BEST	160.5013	3.647051	2.62052E+01	1.3053E-05
F6 Quadratic	MEAN	15.6231	20.2691	0.000E+00	0.000E+00
	MEDIAN	13.50	19.00	0.000E+00	0.000E+00
	STD	7.3160E-14	2.6314	0.000E+00	0.000E+00
	BEST	6.00	5.00	0.000E+00	0.000E+00
F7 Quartic	MEAN	8.6517E-2	0.3218	9.370E-06	9.1874E-05
	MEDIAN	0.005365	0.107872	4.860E-05	7.1340E-05
	STD	8.9206E-17	3.4333E-16	8.030E-20	8.0161E-05
	BEST	0.002111	0.029593	7.050E-07	0.000E+00
F8 Michalewicz	MEAN	−8210.3415	−6899.9556	−9.3367304E+03	-1.1941E+04
	MEDIAN	−8117.66	−7098.95	−8.50555E+03	-1.256948E+04
	STD	833.5126	625.4286	2.640E-12	2.81062E+03
	BEST	−9717.68	−8501.44	−9.85021E+03	-1.25694E+04
F9 Rastrigin	MEAN	62.1441	57.0503	0.000E+00	0.000E+00
	MEDIAN	61.67858	55.22468	0.000E+00	0.000E+00
	STD	2.1637E-13	6.0013E-14	0.000E+00	0.000E+00
	BEST	36.86623	27.85883	0.000E+00	0.000E+00
F10 Ackley	MEAN	3.8134	2.6304	8.88E-16	8.88E-16
	MEDIAN	3.120322	2.170083	8.88E-16	8.88E-16
	STD	6.8972E-15	6.9631E-15	0.000E+00	0.000E+00
	BEST	2.757203	1.155151	8.88E-16	8.88E-16
F11 Griewank	MEAN	1.1973	0.0364	0.000E+00	0.000E+00
	MEDIAN	1.227231	0.029473	0.000E+00	0.000E+00
	STD	4.8521E-15	2.6398E-17	0.000E+00	0.000E+00
	BEST	1.140471	7.29E-09	0.000E+00	0.000E+00
F12 Penalized	MEAN	0.0469	0.4792	5.83E-02	1.4389E-02
	MEDIAN	0.04179	0.1556	1.464E-01	1.9981E-02
	STD	1.7456E-14	9.3071E-15	2.73E-16	25401E-01
	BEST	0.018364	0.000145	4.52E-02	1.2133E-05
F13 Penalized 2	MEAN	1.2106	0.5156	1.42866E+00	4.1196E-08
	MEDIAN	1.218053	0.043997	2.976773E+00	1.2697E-08
	STD	3.5630E-15	4.1427E-16	2.83E-15	5.856E-08
	BEST	0.49809	9.99E-07	1.428663E+00	2.0793E-08
F14 Vincent	MEAN	0.9969	2.3909	9.980E-01	9.980E-01
	MEDIAN	0.998018	0.998004	9.980E-01	9.980E-01
	STD	6.3124E-14	8.0126E-15	0.000E+00	2.2316E-01
	BEST	0.998004	0.998004	9.980E-01	9.980E-01
F15 Kowalik	MEAN	0.0042	0.0528	3.000E-04	3.000E-04
	MEDIAN	0.002074	0.000307	3.000E-04	3.000E-04
	STD	1.6317E-17	2.6159E-18	1.21E-19	2.24E-20
	BEST	0.000775	0.000307	3.000E-04	3.000E-04
F16 Michalewicz	MEAN	-1.0307	-1.0312	-1.0316E+00	-1.0316E+00
	MEDIAN	-1.0309	-1.0311	-1.03163E+00	-1.03163E+00
	STD	9.1449E-15	3.2496E-15	1.93E-18	2.3068E-21
	BEST	-1.0316	-1.0316	-1.03163E+00	-1.0316E+00
F17 Schaffer 2	MEAN	0.4401	0.7951	3.978E-01	3.9799E-01
	MEDIAN	0.4016	0.6521	3.978E-01	3.9789E-01
	STD	1.4109E-16	3.9801E-5	0.000E+00	8.897E-02

F18 Langerman	BEST	0.3978	0.3978	3.978E-01	3.979E-01
	MEAN	4.3601	3.0010	3.000E+00	3.0E+00
	MEDIAN	3.7581	3.0005	3.000E+00	3.0E+00
	STD	2.6108E-15	1.1041E-14	1.090E-16	6.7082E-21
F19 Deb's	BEST	3.0002	3.0E+00	3.000E+00	3.0E+00
	MEAN	-3.8519	-3.8627	-3.86278E+00	-3.6696E+00
	MEDIAN	-3.8413	-3.8560	-3.8627E+00	-3.8628E+00
	STD	3.6015E-14	7.0114E-14	6.45E-01	8.6374E-01
F20 Hartmann (6-Dim)	BEST	-3.86278	-3.8627	-3.8627E+00	-3.862E+00
	MEAN	-2.8301	-3.2626	-3.3220E+00	-3.322E+00
	MEDIAN	-2.96828	-3.2160	-3.322E+00	-3.322E+00
	STD	3.7124E-15	3.4567E-15	1.970E-16	2.570E-21
F21 Hartmann (4-Dim)	BEST	-3.31342	-3.322	-3.322E+00	-3.322E+00
	MEAN	-4.2593	-5.4236	-1.0153E+01	-1.0153E+01
	MEDIAN	-4.16238	-5.10077	-1.01532E+01	-1.01532E+01
	STD	2.3631E-08	6.3014E-09	1.930E-16	5.13E-23
F22 Shekel's Foxholes	BEST	-7.82781	-8.0267	-1.01532E+01	-1.01532E+01
	MEAN	-5.1183	-7.6351	-1.04029E+01	-1.04029E+01
	MEDIAN	-5.0296	-10.4020	-1.04029E+01	-1.04029E+01
	STD	6.1697E-14	5.0610E-14	3.57E-01	2.420E-03
F23 Shekel's Family	BEST	-9.1106	-10.4024	-1.04029E+01	-1.04029E+01
	MEAN	-6.5675	-6.1653	-1.0536E+01	-1.0536E+01
	MEDIAN	-6.5629	-4.50554	-1.05364E+01	-1.05364E+01
	STD	5.6014E-14	5.3917E-15	3.970E-16	1.950E-21
	BEST	-10.2227	-10.5364	-1.05364E+01	-1.05364E+01

Conflict of Interest

The authors declare no conflict of interest in the publication of this research article.

Author Contributions

Author 1: Research conceptualization, interpretation of results, supervision, draft review, and editing; Author 2: Conducted simulation test, and analysis; Author 3: Writing – original draft preparation.

References

[1] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," Comput Ind Eng, vol. 137, Nov. 2019, doi: 10.1016/j.cie.2019.106040.

[2] B. Morales-Castañeda, D. Zaldívar, E. Cuevas, F. Fausto, and A. Rodríguez, "A better balance in metaheuristic algorithms: Does it exist?," Swarm Evol Comput, vol. 54, May 2020, doi: 10.1016/j.swevo.2020.100671.

[3] J. O. Agushaka, A. E. Ezugwu, L. Abualigah, S. K. Alharbi, and H. A. E. W. Khalifa, "Efficient Initialization Methods for Population-Based Metaheuristic Algorithms: A Comparative Study," Archives of Computational Methods in Engineering, vol. 30, no. 3. 2023. doi: 10.1007/s11831-022-09850-4.

[4] L. Abualigah, A. Diabat, S. Mirjalili, M. Abd Elaziz, and A. H. Gandomi, "The Arithmetic Optimization Algorithm," Comput Methods Appl Mech Eng, vol. 376, Apr. 2021, doi: 10.1016/j.cma.2020.113609.

[5] P. Hu, J. S. Pan, and S. C. Chu, "Improved Binary Grey Wolf Optimizer and Its application for feature selection," Knowledge-Based Syst., vol. 195, p. 105746, 2020, doi: 10.1016/j.knosys.2020.105746.

[6] M. Tubishat, M. Alswaitti, S. Mirjalili, M. A. Al-Garadi, M. T. Alrashdan, and T. A. Rana, "Dynamic butterfly optimization algorithm for feature selection," IEEE Access, vol. 8, no. September 2020, pp. 194303–194314, 2020, doi: 10.1109/ACCESS.2020.3033757.

[7] A. S. Assiri, A. G. Hussien, and M. Amin, "Ant lion optimization: Variants, hybrids, and applications," IEEE Access, vol. 8, pp. 77746–77764, 2020, doi: 10.1109/ACCESS.2020.2990338.

[8] A. F. S. Yussif, T. Seini, B. Ayasu, and E. A. Nyantakyi, "Enhancing Reactive Power Compensation in Distribution Systems through Optimal Integration of D-STATCOM using the Pelican Optimization Algorithm," International Journal of Electrical Engineering and Applied Science, vol 7 no. 1, April 2024.

[9] D. Karaboga and C. Ozturk, "A novel clustering approach: Artificial Bee Colony (ABC) algorithm," Appl. Soft Comput. J., vol. 11, no. 1, pp. 652–657, 2011, doi: 10.1016/j.asoc.2009.12.025.

[10] S. Hamid, S. Moosavi, and V. K. Bardsiri, "Engineering Applications of Artificial Intelligence Satin bowerbird optimizer : A new optimization algorithm to optimize ANFIS for software development e ff ort estimation," Eng. Appl. Artif. Intell., vol. 60, no. January, pp. 1–15, 2017, doi: 10.1016/j.engappai.2017.01.006.

[11] P. Trojovský and M. Dehghani, "Pelican Optimization Algorithm: A Novel Nature-Inspired Algorithm for Engineering Applications," Sensors, vol. 22, no. 3, Feb. 2022, doi: 10.3390/s22030855.

[12] P. D. Kusuma and A. L. Prasasti, "Guided Pelican Algorithm," International Journal of Intelligent Engineering and Systems, vol. 15, no. 6, pp. 179–190, Dec. 2022, doi: 10.22266/ijies2022.1231.18.

[13] S. D. SeyedGarmroudi, G. Kayakutlu, M. O. Kayalica, and Ü. Çolak, "Improved Pelican optimization algorithm for solving load dispatch problems," Energy, vol. 289, Feb. 2024, doi: 10.1016/j.energy.2023.129811.

[14] Z. Chen, Y. Wang, T. H. T. Chan, X. Li, and S. Zhao, "A Particle Swarm Optimization Algorithm with Sigmoid Increasing Inertia Weight for Structural Damage Identification," Applied Sciences (Switzerland), vol. 12, no. 7, Apr. 2022, doi: 10.3390/app12073429.

[15] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization," Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011, pp. 633–640, 2011, doi: 10.1109/NABIC.2011.6089659.

[16] H. M. Song et al., "Improved pelican optimization algorithm with chaotic interference factor and elementary mathematical function," Soft comput, vol. 27, no. 15, pp. 10607–10646, Aug. 2023, doi: 10.1007/S00500-023-08205-W/METRICS.