Comparative Study of Different Inertia Weight Strategies in Particle Swarm Optimization Based on Actual Computational Time Cost

N.E. UDENWAGU, A.A. ONI, A.A. EZENWOKE
Covenant University, Ota
NIGERIA

Abstract: - The particle swarm optimization (PSO) algorithm is attracting a lot of research attention due to its superior performance over other swarm-based algorithms. However, one of the major challenges facing PSO is the tendency to fall into local optima, which is known as premature convergence. The inertia weight variable was introduced into PSO to solve this problem by balancing the relationship between exploration and exploitation stages in swarm activity within a given search space. Many studies have proposed different inertia weight strategies to improve on convergence performance of PSO including, Constant Inertia Weight (CIW), Linearly Decreasing Inertia Weight (LDIW), Exponential Inertia Weight (EIW), Chaotic Inertia Weight (CHIW), Nonlinear Decreasing Inertia Weight (NDIW), Adaptive Inertia Weight (AIW), Random Inertia Weight (RIW) and Time Varying Inertia Weight (TVIW). However, these strategies have also introduced varying levels of computational complexities into the PSO algorithm. This study compares eight different inertia weight strategies based on their computational time cost, in order to propose the most efficient strategy. The experiments were carried out using PSO implementation in a Cloudsim simulation environment based on actual computational runtime of each inertia weight strategy. In summary, the chaotic inertia weight strategy has the lowest average runtime of 3610552.27 microseconds, followed by TVIW = 3611035.51 LDIW = 3611035.95, CIW = 3611044.09, AIW = 3611539.87, NDIW = 3612029.75, RIW = 3612520.84, and EIW = 3612524.36.

Keywords: – Inertia weight, Particle swarm optimization, Premature convergence, Time complexity

Received: March 29, 2024. Revised: November 23, 2024. Accepted: December 27, 2024. Published: March 5, 2025.

1. Introduction

Particle Swarm Optimization (PSO) is a swarm-based search algorithm first introduced by Ebernart and Kennedy in 1995 [1]. Compared to other nature inspired optimization algorithms, PSO has proven to be more efficient and easier to implement due to its smaller number of parameters [2]. Its performance is largely dependent on selection and tuning of its parameters such as the inertia weight.

In a population-based swarm intelligence algorithms like PSO, exploration and exploitation are the two important search stages. The balance between these two stages will yield the best results and achieve maximum efficiency. Inertia weight (ω) in PSO is a critical parameter that controls this balance in the search space. It determines how much of the particle's previous velocity will be retained in the next iteration. Choosing an appropriate inertia weight is crucial for the performance of PSO to prevent it from converging prematurely. When the inertia weight is large, the algorithm tends towards a global search while a smaller inertia weight tends more to a local search [2]. Inertia weight was first introduced by Shi and Ebernart [3] to improve on population diversity and balance in PSO. Different inertia weight strategies have been proposed in existing literature to solve this problem of premature convergence including constant inertia weight (CIW) [4], exponential inertia weight (EIW) [5], linearly decreasing inertia weight (LDIW) [6], nonlinear decreasing inertia weight (NDIW) [7], random inertia weight (RIW) [8], adaptive inertia weight (AIW) [9], time varying inertia weight (TVIW) [10] and chaotic inertia weight (CHIW) [11]. These different strategies have introduced varying levels of improvement along with their computational complexities into PSO as investigated in this study.

The computational time cost of an algorithm also referred to as time complexity, describes how the runtime of an algorithm increases with the size of its input [12]. It provides a high-level measure of the efficiency of an algorithm and is typically expressed using Big-O notation. The time cost of an algorithm depends on factors such as, input size (n), the number of basic operations, best, average and worst case scenarios [13]. In this study, the time complexity of each inertia weight strategy was observed and compared based on the actual computational time cost of executing the PSO algorithm. The rest of this study is organized as follows: Section 2 is a brief review of the characteristics of PSO, section 3 explains the different inertia weight strategies and their time complexities as presented in literature. Section 4 presents the experiments to compare different strategies, while section 5 discusses the experimental results. Finally, the last section presents the conclusion.

2. Basic Concepts of Particle Swarm Optimization

The first version of PSO is generally referred to as standard PSO (SPSO) [14]. Standard PSO contains a swarm of particles moving in a D-dimensional search space in order to locate an optimal solution. Each particle i, has a current velocity vector $Vi = [vi1, vi2, ..., viD]$ and a current position vector $Xi = [xi1, xi2, ..., xiD]$, where D is the number of dimensions. In the SPSO, the Vi and Xi are first randomly initialized. Then in consecutive iterations, $Pbest_i$ which is the best position found

by particle i, is represented as $Pbest_i = [Pbest_{i1}, Pest_{i2}, \ldots, Pbest_{iD}]$ and the best position found by the whole swarm $Gbest = [Gbest_1, Gbest_2, \ldots, Gbest_D]$. Both the local position and the global position are therefore used as a guide in updating the velocity and position of each particle as shown in equations 1 and 2 [15].

$$V_{id}^{(t+1)} = \omega V_{id}^{(t)} + c_1 r_1 \left(Pbest_{id}^{(t)} - X_{id}^{(t)}\right) + c_2 r_2 \left(Gbest_d^{(t)} - X_{id}^{(t)}\right) \quad (1)$$

$$X_{id}^{(t+1)} = X_{id}^{(t)} + V_{id}^{(t+1)} \quad (2)$$

Where ω is the inertia weight, which determines to what extent the previous velocity of a particle is preserved, c_1 is the cognitive acceleration coefficient, c_2 is the social acceleration coefficient, both of which are positive constants, r_1 and r_2 are the two uniform random values generated within the range [0,1]. The pseudo-code for solving a minimization optimization problem, using standard PSO is shown in Algorithm 1.

Algorithm 1: Standard PSO pseudo code for minimization optimization [15]

1: Initialization
2: Define the swarm size S and the number of dimensions D
3: for each particle i Є [1…S]
4: Randomly generate Xi and Vi, and evaluate the fitness of Xi denoting it as f(Xi)
5: Set Pbesti = Xi and f(Pbesti) = f(Xi)
6: end for
7: Set Gbest = Pbest1 and f(Gbest) = f(Pbest1)
8: for each particle i Є [1…S]
9: if f(Pbesti) < f(Gbest) then
10: F(Gbest) = f(Pbesti)
11: end if
12: end for
13: while t < maximum number of iterations
14: for each particle i Є [1...S]
15: Evaluate its velocity Vid (t+1) using Equation (1)
16: Update the position Xid(t+1) of the particle using Equation (2)
17: if f(Xi(t+1)) < f(Pbesti) then
18: Pbesti = Xi (t+1)
19: f(Pbesti) = f(Xi(t + 1))
20: end if
21: if f(Pbesti) < f(Gbest) then
22: Gbest = Pbesti
23: f(Gbest) = f(Pbesti)
24: end if
25: end for
26: t = t + 1
27: end while
28: return Gbest

2.1 Advantages of Particle Swarm Optimization Algorithm

There are several meta-heuristic algorithms that have been used to solve many types of optimization problems. However, some of these algorithms may have some weaknesses such as too many parameters, requiring high programming skills, extremely high computational cost, transmuting to binary forms and so on [16], [17]. The advantages of PSO can be summarized into three basic categories. (a) Relative implementation simplicity. (b) Fewer controlling parameters, (inertia weight, cognitive ratio, and social ratio). (c) PSO also is easily hybridized with other optimization algorithms. PSO equally has a good control of its exploration and exploitation phases. In exploration, particles carry out extensive search of the space, while in exploitation, particles focus on promising regions of the explored space [18].

Having considered the excellent performance of PSO, it is important to look at some of its weaknesses which can be improved by new modifications to its variants especially the inertia weight. The first weakness in PSO is premature convergence as outlined in literature. Premature convergence is caused by lack of diversity in population, especially in complex multimodal functions [15]. In PSO, the convergence criterion measures the closeness of a particle in the swarm to reaching the optimal solution in the problem space. Therefore if the particles converge away from the actual optimum, this is termed as premature. Premature convergence is also the downside of its fast convergence speed. Another weakness of PSO is the difficulty of controlling the parameters [19]. Though it has three parameters only (c1, c2, w), finding appropriate setting for these parameters at each iteration has proven difficult. Several methods have been proposed in literature for controlling the parameters but none guarantees an optimal setting. Another weakness of PSO is improper velocity adjustment which occurs when inappropriate values of the parameter are chosen [20]. This makes the particles fly in wrong directions, thereby causing stagnation around the optimum solution.

2.2 Time Complexity Analysis of Particle Swarm Optimization

The main factors influencing the time complexity of the PSO algorithm include, the number of particles (swarm size N), the number of iterations (termination factor I), the dimensionality of the problem being solved D, and the cost function complexity C [12], [13]. The swarm size determines the number of candidate solutions that are evaluated per iteration. A higher number of particles typically increased the chances of finding a global optimum but also increases computational cost. Each particle has its own velocity and position in the

solution space. The number iterations directly affect the time complexity, as each iteration evaluates the entire swarm. It defines how long the PSO will run and how many times the particle positions will be updated. This parameter can be set based on the convergence criteria (such as fixed number of iteration or when a solution is found). Larger iteration counts improve the chance of convergence but lead to higher computation time. PSO can stop early if convergence occurs, but in worst-case scenarios, all iterations are needed, contributing to the time complexity.

The dimensionality is the number of variables or parameters in the problem space, which influences how much work is done when calculating new positions and velocities [21]. Each particle exists in a D-dimensional space where D represents the number of variables in the optimization problem. Each particle must update its position and velocity in all dimensions. High-dimensional problems increase the complexity since each particle needs to adjust its position in each dimension at every iteration. For example, if you have 10-dimensional optimization problem, every particle needs to update its position in each of the 10 dimensions, in each iteration. The cost function is the complexity of evaluating the fitness function for a single particle, which can be significant especially in the case of multi-objective optimization and therefore is problem-specific. This is usually the most computationally expensive part of PSO [22]. For each particle in every iteration, the algorithm has to calculate the fitness of the current position. The complexity of the fitness evaluation function depends entirely on the specific problem being solved. If evaluating the fitness function is computationally expensive, for example non-linear functions or functions requiring complex mathematical operations, it will dominate the time complexity of PSO.

Considering these factors, the average overall time complexity T of PSO can be expressed as shown in equation 1.

$$T = O(N * I * D * C) \quad (1)$$

Where N is the number of particles (swarm size), I is the number of iterations, D is the dimensionality of the problem and C is the complexity of evaluating the fitness function. In general, PSO is considered as efficient algorithm. However, as C grows in complexity, for example in cases of computationally intensive inertia weight and constriction factor strategies, the overall runtime can increase significantly.

3. Different Inertia Weight Strategies in PSO

In Particle Swarm Optimization (PSO), inertia weight is a crucial factor that controls the trade-off between exploration (global search) and exploitation (local search) abilities of the algorithm [4]. Various strategies for adjusting the inertia weight have been developed to improve the performance of PSO, with each strategy aimed at achieving a balance between exploration and exploitation. The choice of strategy largely depends on the nature of the optimization problem, desired convergence speed, and complexity. The key is finding a balance between exploration and exploitation while keeping the computational complexity the barest minimum. Some inertia weight strategies are commonly used in many optimization problems, while some advanced techniques and hybrid methods have been proposed to further enhance PSO's performance [23]. These advanced strategies are often tailored to specific problems or used in combination with other mechanisms to increase the algorithm's adaptability. In this section, the different inertia weights presented in literature are analyzed in the light of their computational time cost and also summarized in table 1.

3.1 Constant Inertia Weight (CIW)

In this strategy, the inertia weight is kept constant throughout the entire optimization process.

It is simple to implement, however, a fixed value might not be ideal for all stages of the optimization process. It can result in poor convergence behavior (either too much exploration or premature convergence). Constant inertia weight was first introduced into PSO by Shi and Ebernart in 1998 with values between 0.8 and 1.2 as shown in equation 6, [4]. They concluded that inertia weight values outside the range of [0.8, 1.2] may not yield the optimal result. Since then other researchers have developed different variants of the constant inertia weight. In [24] a constant inertia weight of value 0.85 (equation 7), was used to improve the convergence capacity of multi-objective PSO in conjunction with some other nature-inspired algorithms, for green mobility in electric vehicle charging. They used a MOPSO algorithm with a constant inertia weight to tackle the challenge of simultaneous optimization of conflicting objectives such as minimizing costs, maximizing energy production, and reducing environmental impact. The equations for the two studies are represented in equations 2 and 3. Due to its simplicity, constant inertia weight incurs negligible computational cost on the algorithm because it consists only of one assignment statement per iteration

$$\omega = 0.8, 1.2 \quad (2)$$

$$\omega = 0.85 \quad (3)$$

Where ω = Inertia weight

3.2 Linearly Decreasing Inertia Weight (LDIW)

This inertia weight is decreased linearly over time, starting with a high value, encouraging exploration at the beginning,

and decreases to a smaller value, encouraging exploitation as the algorithm progresses. It helps maintain a balance between exploration in the early stages and exploitation in the later stages. It might not be flexible enough to adapt to the problem's complexity. In [6], a linearly decreasing inertia weight was proposed as shown in equation 4. The algorithm performed best where ω_1 is larger than ω_2 thereby decreasing the inertia weight slowly as the iteration progresses. In their experiments the parameters were set as $\omega_1 = 0.9$, $\omega_2 = 0.4$, $T_{max} = 300$ and $\alpha = 2$. In [25] a linear decreasing inertia weight was implemented combined with a chaotic inertia weight. Their variant of LDIW is shown in equation 5. The two inertia weights were diffused into one by finding a common lowest denominator for each equation. This strategy was also implemented in the study [26].

$$\omega(t) = \omega_1 - (\omega_1 - \omega_2)(t/T_{max})^{\alpha} \quad (4)$$

Where $\omega(t)$ is the inertia weight related to the number of iterations, ω_1 is the initial inertia weight, ω_2 is the maximum inertia weight, t is the current number of iterations, T_{max} *is the m*aximum number of iterations, α is the exponential coefficient.

$$\omega_l = \omega_{max} - \left(\frac{\omega_{max} - \omega_{min}}{I_{max}}\right) * t \quad (5)$$

Where ω_l is the current inertia weight, ω_{max} is the maximum inertia weight, ω_{min} is the minimum inertia weight I_{max} is the maximum iteration and t is the current iteration.
In most cases, the complexity of LDIW consists of simple assignment statement and normal arithmetic operations such as addition, division and subtractions. However, in some variants such as [6], a complex exponential factor is introduced which significantly increases the computation overhead of the algorithm as the number of iteration increases.

3.3 Nonlinear Decreasing Inertia Weight (NDIW)

In this strategy, the inertia weight decreases nonlinearly (e.g., exponentially, logarithmically) over time. The nonlinear decay allows for more flexibility than the linear approach. It provides more fine-grained control over the exploration-exploitation trade-off, but may require tuning of decay parameters. In [27] a non-linear decreasing inertia weight was proposed as shown in equation 6. In their study, the velocity formula was modified by using time varying inertia weight, social and cognitive variables. Another non-linear strategy was implemented in [7] as shown in equation 7. The strategy was defined based on the best global position, the best fitness, the minimal fitness and the current iteration.

$$\omega = \omega_{min} + (\omega_{max} - \omega_{min})((k_{max} - k)/k_{max})^{\beta} \quad (6)$$

Where ω_{min} is the minimum inertia weight = 0.4, ω_{max} is the maximum inertia weight = 0.9, k_{max} is the maximum iteration, β is the exponential coefficient = 0.4.

$$\omega = ((gb - f_{max})^{-1}/(-ln(f_{min}))/100000fr \quad (7)$$

where gb is the best position found in the entire swarm, f_{max} is maximal fitness in the current iteration and f_{min} is the minimal fitness value in the current iteration respectively. Parameter fr is the randomly generated number in the range [0,1]. The time complexity of NDIW is similar to that of LDIW. However, in NDIW the non-linear factor is amplified by the extra arithmetic operator in each iteration. For example, in [7], the inverse function of global best and the minimal fitness adds extra complexity to the algorithm.

3.4 Random Inertia Weight (RIW)

This this strategy, the inertia weight is chosen randomly within a predefined range. This adds stochasticity, which can help avoid local minima and enhance exploration, but may lead to unstable or erratic behavior without proper tuning. In [28] a random inertia weight was proposed where N(0,1) obeys the standard state distribution as shown in equation 8.

$$\omega = \left(\mu_{min} + (\mu_{max} - \mu_{min}) * rand(0,1)\right) + \sigma * N(0,1) \quad (8)$$

Where μ_{max} is set = 0.8, μ_{min} = 0.4 and σ = 0.8.

In another study [8], a stochastic inertia weight was proposed as given in equation 9, where ω_{min} is the minimum inertia weight = 0.4, ω_{max} is the maximum inertia weight = 0.9, k is the current iteration and T_{miax} is maximum iteration, rb is a random number in the interval range (-0.1,0.1).

$$\omega = \omega_{max} + (\omega_{max} - \omega_{min}) * (k/T_{max}) * e^{rb} \quad (9)$$

The complexity of random inertia weight strategy consists of basic arithmetic operations in combination with a random operator and complex exponential operator. For example in [8], the complex exponential factor e^{rb} is combined with the division of current iteration k and maximum iteration T_{max}

3.5 Adaptive Inertia Weight (AIW)

In the adaptive strategy, the inertia weight is adjusted dynamically based on the performance of the particles or other indicators such as velocity or fitness. It is more adaptive to different optimization stages and problem complexities. However, it is more complex to implement and requires careful design.
For example, if the particles are not improving, the inertia weight may be increased to encourage exploration, or decreased if the particles are showing good convergence. In [29] a Bayesian method was used to design an adaptive inertia weight, where the weight could change according to the initial position of particle. However, the Bayesian approach has a drawback of local optima which was solved using a mutation operation. In [30] a stability-based adaptive inertia weight (SAIW) was also introduced. In order to adjust the inertia weight for each particle, feedback from the particle is used, based on the fact that the particle's performance is memorized. The value of inertia weight is considered different for each

dimension, so that the convergence speed increases, especially in asymmetric environments. The value of the inertia weight in each dimension is used to compute the acceleration coefficients adaptively.
In [9], the inertia weight value determines the speed of the search particles during each iteration which is directly related to the adaptive values of the current iteration number. Therefore the adaptive weight is calculated as shown in equation 10 – 13.

$$p_1 = \omega_0 + (f_i - f_{avg})/(f_{max} - f_{min})\, f > f_{avg} \quad (10)$$

$$p_1 = \omega_0 + (f_{avg} - f_i)/(f_{max} - f_{min})\, f \leq f_{avg} \quad (11)$$

$$p_2 = t/t_{max} \quad (12)$$

$$\omega = e^{-p2} + p_1 \quad (13)$$

Where t_{max} is the maximum iteration number and t is the current iteration. ω_0 is the starting inertia weight. f_i is the current fitness value of the ith particle, f_{avg} is the average fitness value of the whole group, f_{max} is the best adaptive value and f_{min} is the worst adaptive value of a particle at iteration t. In [31] an adaptive inertia weight strategy was proposed where the inertia weight is updated by the global best value and the exponential function of the current value as in equations 15 and 16. For each iteration, if the relative difference between the current *w* value and the global optimal value is large, the inertia weight increases, and vice versa. Other adaptive inertia weight strategies include [32], [33], [34].

$$\omega(t+1) = \omega_{start} + (\omega_{end} + \omega_{start}) * (\omega_{end} + \omega_{start})^{\frac{e^{mi(t)}-1}{e^{mi(t)}+1}} \quad (14)$$

$$mi(t) = (gB^t - x_i^t)/(gB^t + x_i^t) \quad (15)$$

Where *t* is the current number of iteration and w_{start} and w_{end} are the starting and ending inertia weight respectively. The adaptive inertia weight strategies consist of several complex arithmetic operations. For example, in [31], the velocity update in each iteration contains complex exponential factors of global best and previous position. It also includes the exponentiation of the sum of starting and ending inertia weights. These complex operations impose significant level of computational complexity on the algorithm.

3.6 Chaotic Inertia Weight (CHIW)

This approach utilizes chaotic sequences to adjust the inertia weight. Chaotic sequences provide randomness but are deterministic and can help avoid stagnation in optimization. It combines deterministic and random behaviors, helping to explore and exploit the search space more effectively. It requires careful selection of chaotic maps and parameters. In [11] a chaotic inertia weight was proposed to integrate the fitness based dynamic inertia weight with velocity update as shown in equation 16. Other studies on chaotic inertia weight include [25],

$$\omega = 1.1 - \left(\frac{0.9 * f_i}{f_{best} + 0.1}\right) \quad (16)$$

Where:

f_i = Fitness of the *i*th particle
f_{best} = Fitness of global best particle

3.7 Time-Varying Inertia Weight (TVIW)

In this strategy, the inertia weight varies based on time or iteration, often through pre-designed schedules. It provides a smooth transition between exploration and exploitation phases, but may require tuning of the schedule. A time varying inertia weight was proposed in [10] which is a modification of adaptive inertia weigh, SAIW. However, the proposed inertia weight decreases faster that the SAIW and is shown in equation 17.

$$\omega = (\omega_{max} - \omega_{min}) * 0.85^t + \omega_{min} \quad (17)$$

Where ω_{max} is the maximum inertia weight, ω_{max} is the minimum inertia weight and t is the current iteration.

3.8 Exponential Inertia Weight (EIW)

In this strategy, the inertia weight follows a quadratic or polynomial function rather than a linear one. This gives more flexibility in controlling the rate at which inertia weight decreases. It has more control over the speed of inertia weight decay, allowing for gradual or steep transitions between exploration and exploitation, but requires parameter tuning for the polynomial degree and coefficients. In [5], a quadratic inertia weigh was proposed in which ω decreased from higher value initially for greater exploration and then tends to be constant throughout the middle range of iterations. This strategy is shown in equation 18 - 20.

$$\wedge = ((\omega_{start})^2 - (\omega_{end})^2)/2(\omega_{start})^2 \quad (18)$$

$$\omega_{iter} = \omega_{start}((1 - 2 * \wedge * (r/_{maxiter})^k)^{1/2}) \quad (19)$$

Where the starting inertia weight ω is set as 0.7 and the ending is 0.4. The *maxiter* is the maximum number of iterations set as 200 and r is the increment variable starting from
the value 0 to *maxiter*. *k* is the profile variable set =15. In [35], the authors proposed a natural exponential (base e) inertia weight strategy as expressed in equation 24. It is assumed that ω_{start} is = 0.9, ω_{end} is = 0., *Tmax* is fixed at 3000. Other similar inertia weight strategies were proposed in [23] and [36].

$$\omega(t) = \omega_{min} + (\omega_{max} - \omega_{min}) * e^{-t/(\frac{maxiter}{10})} \quad (20)$$

Table I
Summary of Different Inertia Weight Strategies

NAME	REF	VALUE (EQUATION)
Constant inertia weight	[4]	ω= [0.8,1.2]
	[24]	ω= [0.85]
Linearly decreasing inertia weight	[6]	$\omega(t) = \omega_1 - (\omega_1 - \omega_2)(t/T_{max})^{\alpha}$
	[25]	$\omega_l = \omega_{max} - \left(\frac{\omega_{max} - \omega_{min}}{I_{max}}\right) * t$
Nonlinear decreasing inertia weight	[27]	$\omega = \omega_{min} + (\omega_{max} - \omega_{min})((k_{max} - k)/k_{max})^{\beta}$ (10)
	[7]	$\omega = ((gb - f_{max})^{-1}/(-ln(f_{min}))/100000fr$
Random inertia weight	[28]	$\omega = \left(\mu_{min} + (\mu_{max} - \mu_{min}) * rand(0,1)\right) + \sigma * N(0,1)$
	[8]	$\omega = \omega_{max} + (\omega_{max} - \omega_{min}) * (k/T_{max}) * e^{rb}$
Adaptive inertia weight	[9]	$\omega = e^{-p2} + p_1$
	[31]	$\omega(t+1) = \omega_{start} + (\omega_{end} + \omega_{start}) * (\omega_{end} + \omega_{start})^{\frac{e^{mi(t)}-1}{e^{mi(t)}+1}}$ $mi(t) = (gB^t - x_i^t)/(gB^t + x_i^t)$
Time-varying inertia weight	[10]	$\omega = (\omega_{max} - \omega_{min}) * 0.85^t + \omega_{min}$
Chaotic inertia weight	[11]	$\omega = 1.1 - \left(\frac{0.9 * f_i}{f_{best} + 0.1}\right)$
Exponential inertia weight	[5]	$\omega_{iter} = \omega_{start}((1 - 2 * \wedge * (r/_{maxiter})^k)^{1/2})$
	[35]	$\omega(t) = \omega_{min} + (\omega_{max} - \omega_{min}) * e^{-t/(\frac{maxiter}{10})}$ (22)

4. The Experiments

4.1 Simulation Environment

To compare the computational time of different inertia weight strategies, the PSO algorithm was implemented

in Cloudsim simulation environment. Cloudsim is an open source cloud simulation tool specifically designed to model and analyze the performance of cloud computing environment [37]. It is a software framework based on Java programming language that allows researchers, cloud service providers, and cloud users to simulate and evaluate various aspects of cloud systems. The main purpose of Cloudsim is to assist in understanding the behavior and performance characteristics of cloud computing systems under different conditions. It helps users assess the impact of various factors such as workload patterns, resource allocation, strategies, scheduling policies, and system configurations on the performance and efficiency of cloud deployments [38]. The experiments were carried out in a computer with the following configuration: OS – Windows 10 (x64), Intel® Core™ i5-4210U CPU, 1.70GHz, 2.40GHz. RAM – 6GB

4.2 Parameter Settings

The PSO algorithm parameters in this study are selected uniformly for all inertia weight strategies examined. This is to ensure a fair comparison between the strategies. The population sized is selected as 1,000,

while $c_1 = c_2 = 2$. r_1 is a random number between [0,1], r_2 is also a random number between [0,11] but selected independent of r_1. Each experiment was executed 15 times (15 cycles), and each cycle contains 805 loops of 100 iterations. Eight inertia weight strategies were used for the actual experiments as listed in table 2. Each equation was encoded and implemented separately in PSO algorithm to run the experiments and the results recorded for comparison.

4.3 Objective Function

In this study, the PSO algorithm was implemented to optimize one objective function which is task execution time (TET) by scheduling incoming tasks to available virtual machines in cloud computing infrastructure. The mathematical model of TET is shown in equations 21 and 22. The first metric examined is the task execution rate of each virtual machine, which can be expressed as a function of the virtual machine speed and the number of CPU in the virtual machine [17]. This metric is part of the task execution time as shown in equation 22.

$$R_j = S_j * C_j \quad (21)$$

Where R_j is thhe task execution rate of the j^{th} virtual machine, S_j is the speed of the the j^{th} virtual machine, and C_j = The number of CPU in the the j^{th} virtual machine.

The task execution time (TET) is the amount of time expended in executing a single task on a given virtual machine [39]. It is a function of the task length and the rate of execution of the virtual machine as shown in equation 22. This objective functions is to be minimized.

$$TET_{ij} = TL_i / R_j \quad (22)$$

Where, TET_{ij} is the time taken to execute task *i* on virtual machine *j*, TL_i is the length of task *i, and* R_j is the task execution rate of virtual machine *j*.

Table II
Inertia Weight Strategies Used in Actual Experiments

SN	NAME	VALUE (EQUATION)	REF	SETTING
1	CIW	ω= [0.85]	[24]	ω = 0.85
2	LDIW	$\omega(t) = \omega_1 - (\omega_1 - \omega_2) * (t/T_{max})^{\alpha}$	[6]	$\omega_1 = 0.9$, $\omega_2 = 0.4$, T_{max} = 100 and $\alpha = 2$.
3	NDIW	$\omega = \omega_{min} + (\omega_{max} - \omega_{min}) * ((k_{max} - k)/k_{max})^{\beta}$	[27]	ω_{min} *0.4,* ω_{max} *=0.9,* β*=0.4*
4	RIW	$\omega = \omega_{max} + (\omega_{max} - \omega_{min}) * (k/T_{max}) * e^{rb}$	[8]	$\omega_{min} = 0.4$, $\omega_{max} = 0.9$, rb = rand(0,1).
5	AIW	$\omega(t+1) = \omega_{start} + (\omega_{end} + \omega_{start}) * (\omega_{end} + \omega_{start})^{\frac{e^{mi(t)}}{e^{mi(t)}+1}}$ $mi(t) = (gB^t - x_i^t)/(gB^t + x_i^t)$	[31]	ω_{start} *0.9,* ω_{end} *=0.4,*
6	CHIW	$\omega = 1.1 - \left(\frac{0.9 * f_i}{f_{best} + 0.1}\right)$	[11]	f_i = *i*th fitness f_{best} = global best
7	TVIW	$\omega = (\omega_{max} - \omega_{min}) * 0.85^t + \omega_{min}$	[10]	ω_{min} *0.4,* ω_{max} *=0.9*
8	EIW	$\wedge = ((\omega_{start})^2 - (\omega_{end})^2)/2(\omega_{start})^2$ $\omega_{iter} = \omega_{start}((1 - 2 * \wedge * (r/_{maxiter})^k)^{1/2})$	[5]	$\omega_{start} = 0.9$, $\omega_{end} = 0.4$, *maxiter* = 100, k = 15

5. Discussion of Results

This study analyzed thirteen different inertia weight strategies grouped into eight categories. However, eight of them were selected, one from each category for comparative analysis. These include, constant inertia weight [24], linear decreasing inertia weight [6], nonlinear decreasing inertia weight [27], random inertia weight [8], adaptive inertia weight [31], chaotic inertia weight [11], time varying inertia weight [10] and exponential [5] inertia weigh strategies. The strategies were compared based on computational time cost in Particle Swarm Optimization (PSO) algorithm.

The results recorded in table 3 represent the actual runtime of the algorithm for each inertia weight strategy, measured in microseconds. Each experiment was carried out 15 times, based on 100 iterations per cycle, to calculate the average, standard deviation, minimum and maximum runtime. The average time for each inertia weight strategy are, CIW = 3611044.09, LDIW = 3611035.95, NDIW = 3612029.75, RIW = 3612520.84, AIW = 3611539.87, CHIW 3610552.27, TVIW = 3611035.51 and EIW = 3612524.36. This represents the average time taken to run the PSO algorithm for optimizing the task execution time (TET) in load balancing of a cloud infrastructure. In addition, the standard deviations for the eight strategies are 3824.25, 3833.25, 3616.09, 3397.32, 3752.70, 3822.45, 3833.74, and 3391.26 respectively.

Considering the line graph in Fig. 1, it indicates that the CIW first hit the minimum value of 3607044.92 at two points before a maximum value of 3614500.00 at two points also. This indicates that the CIW maintained a regular behavior in time consumption during the entire experiment. As shown in Fig. 2, the LDIW shows consistently varying time consumption in all cycles, going between lowest and highest values of 3607044.92 and respectively. As for NDIW, Fig. 3 indicates an initial oscillation between a low value of 3607068.42 and high value of 3614500.00, before maintaining a stable high value. The RIW maintained its high computations time of 3614500.00 in most cycles, touching its lowest value 3607044.92 only at the beginning and once at the middle as shown in Fig. 4. The AIW stays on either high value or low values consistently. It does not swing between highs and lows regularly like RIW. It has its highest value at 3614500.00 and lowest value at 3607048.17 as shown in Fig. 5. The TVIW showed a similar behavior as the LDIW and AIW as shown in Fig. 6. This is expected since a combination of linear decreasing and adaptive behavior will lead to time varying tendencies. The EIW as shown in Fig. 7 displays a balance between continuous high values of 3614500.00 and continuously oscillating between low values of 3607044.92. Fig. 8 showed that the CHIW maintained its chaotic nature by not following any regular pattern but moving between high and low values. Finally, Fig. 9 summarized the general behavior of the strategies, showing that they all remained between a certain range of 3607044.92 to 3614500.00 microseconds.

In summary, the chaotic inertia weight strategy has the lowest computational time followed by time varying strategy based on the results. The eight inertia weight strategies can therefore the ranked in descending order of computational time cost as follows CHIW, TVIW, LDIW, CIW, AIW, NDIW, RIW, and EIW. The study in in [40] ranked 6 different inertia weight strategies based on convergence behavior and concluded that AIW performed the best. Again the study in in [41] which compared 15 inertia weight strategies and concluded that the CHIW strategy recorded the best performance. In other words, the results in this study show that the superior computational complexity of CHIW strategy corresponds with its optimal performance in convergence as recorded [41].

Table III
Results of Actual Computation Time for Different Inertia Weight Strategies

Cycle	CIW	LDIW	NDIW	RIW	AIW	CHIW	TVIW	EIW
1	3614500.00	3614500.00	3614500.00	3607044.92	3614500.00	3607128.66	3607095.92	3614500.00
2	3607113.71	3607095.92	3614500.00	3614500.00	3614500.00	3614500.00	3607095.92	3614500.00
3	3607068.42	3614500.00	3607095.93	3614500.00	3614500.00	3607044.92	3614500.00	3614500.00
4	3607044.92	3614500.00	3607099.74	3614500.00	3614500.00	3607123.49	3614500.00	3614500.00
5	3607095.93	3607044.92	3614500.00	3614500.00	3607123.50	3607123.49	3614500.00	3614500.00
6	3614500.00	3607095.64	3607113.71	3614500.00	3607113.71	3614500.00	3607048.17	3607048.17
7	3614500.00	3614500.00	3614500.00	3607048.17	3607048.17	3614500.00	3607128.66	3614500.00
8	3614500.00	3614500.00	3614500.00	3614500.00	3607113.71	3607048.17	3614500.00	3607123.49
9	3607095.93	3614500.00	3614500.00	3614500.00	3614500.00	3607123.49	3614500.00	3614500.00
10	3614500.00	3607095.93	3614500.00	3614500.00	3614500.00	3614500.00	3614500.00	3614500.00
11	3614500.00	3614500.00	3607068.42	3607123.50	3614500.00	3614500.00	3607048.17	3614500.00
12	3607128.67	3607044.92	3614500.00	3614500.00	3607070.25	3607095.92	3614500.00	3607123.49
13	3607113.71	3614500.00	3614500.00	3607095.93	3614500.00	3614500.00	3607044.92	3607070.25
14	3614500.00	3607048.17	3607068.42	3614500.00	3614500.00	3614500.00	3614500.00	3614500.00
15	3614500.00	3607113.71	3614500.00	3614500.00	3607128.66	3607095.92	3607070.92	3614500.00
AVG	**3611044.09**	**3611035.95**	**3612029.75**	**3612520.84**	**3611539.87**	**3610552.27**	**3611035.51**	**3612524.36**
STD	**3824.25**	**3833.25**	**3616.09**	**3397.32**	**3752.70**	**3822.45**	**3833.74**	**3391.26**
MIN	**3607044.92**	**3607044.92**	**3607068.42**	**3607044.92**	**3607048.17**	**3607044.92**	**3607044.92**	**3607048.17**
MAX	**3614500.00**	**3614500.00**	**3614500.00**	**3614500.00**	**3614500.00**	**3614500.00**	**3614500.00**	**3614500.00**

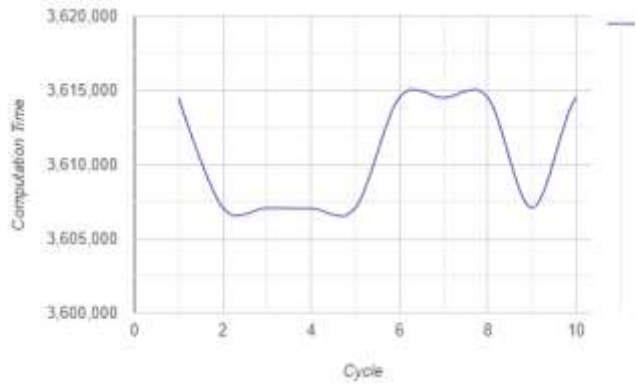

Fig. 1 Computation time of PSO based on constant inertia weight

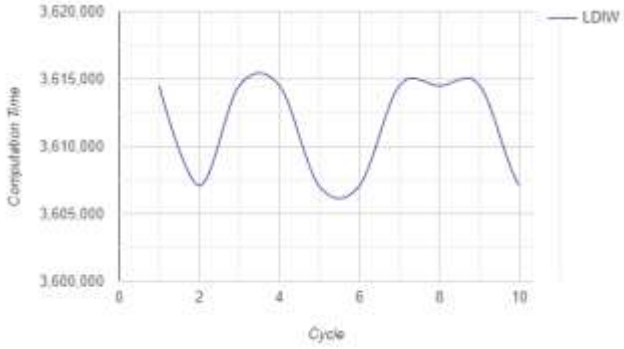

Fig. 2 Computation time of PSO based on linearly decreasing inertia weight

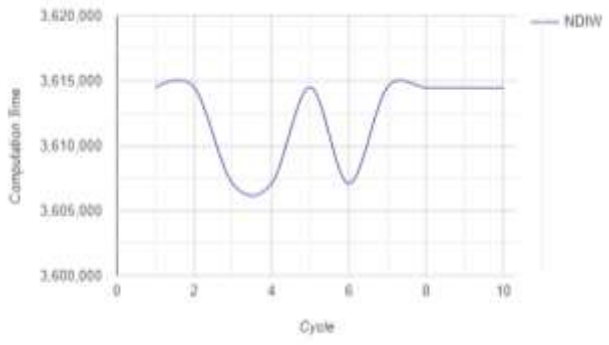

Fig. 3 Computation time of PSO based on nonlinear decreasing inertia weight

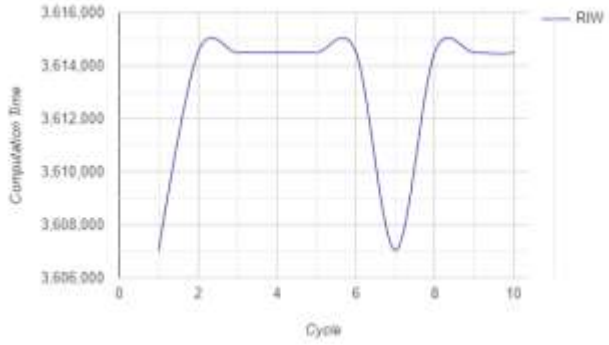

Fig. 4 Computation time of PSO based on random inertia weight

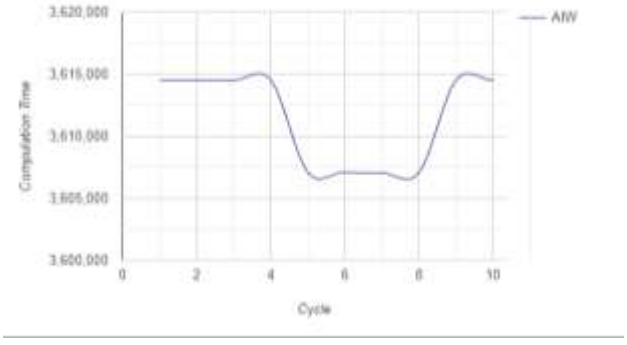

Fig. 5 Computation time of PSO based on adaptive inertia weight

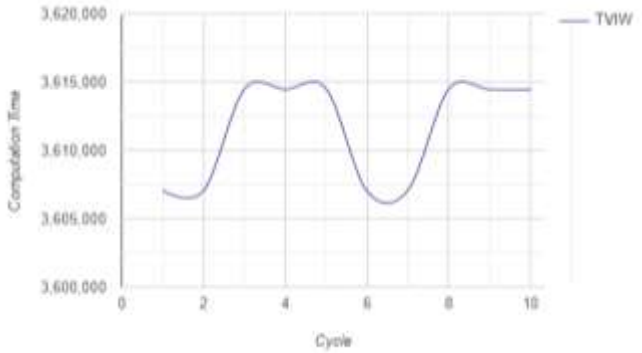

Fig. 6 Computation time of PSO based on time varying inertia weight

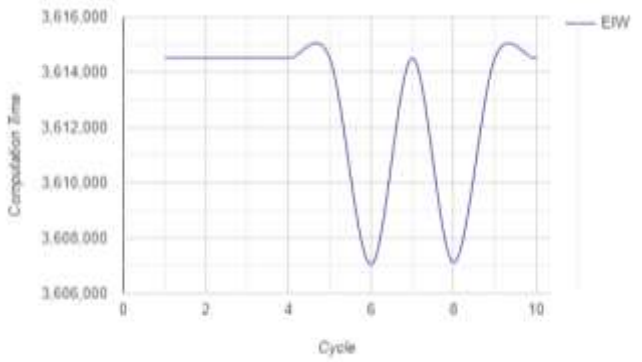

Fig. 7 Computation time of PSO based on exponential inertia weight

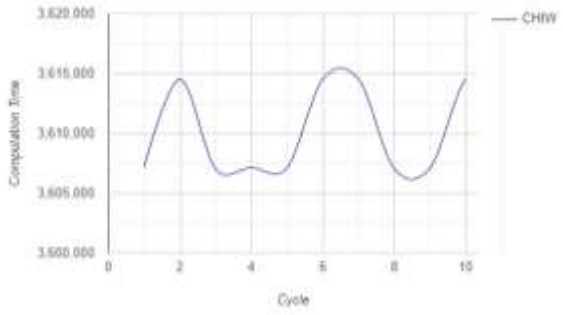

Fig. 8 Computation time of PSO based on chaotic inertia weight

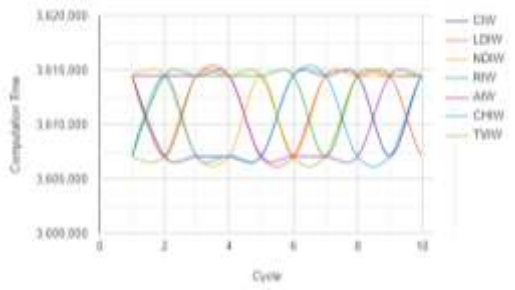

Fig. 9 Comparative computation time for PSO based on different inertia weight strategies

Fig. 10 Comparative computation time for PSO based on different inertia weight strategies

6. Conclusion

In this study, eight different PSO-based inertia weight strategies were compared on the basis of their actual computational time cost. These strategies include, constant inertia weight (CIW), linearly decreasing inertia weight (LDIW), nonlinear decreasing inertia weight (NDIW), random inertia weight (RIW), adaptive (AIW), chaotic inertia weight (CHIW), time varying inertia weight (TVIW) and exponential inertia weight (EIW) strategies. The experiments compared the strategies by observing the computational runtime of each algorithm in optimizing the task execution time in a cloud infrastructure using Cloudsim simulation tool. In summary, the chaotic inertia weight strategy has the lowest average runtime of 3610552.27 microseconds, followed by TVIW = 3611035.51 LDIW = 3611035.95, CIW = 3611044.09, AIW = 3611539.87, NDIW = 3612029.75, RIW = 3612520.84, and EIW = 3612524.36.

Therefore the inertia weight strategies arranged in descending order of performance are CHIW, TVIW, LDIW, CIW, AIW, NDIW, RIW, and EIW. Researchers seeking to choose from existing variants of PSO and other swarm based algorithms may find this study useful. In a future work, it will be important to consider the impact of different constriction factor strategies on the performance of Particle Swarm Optimization.

References

[1] J. Kennedy and R. C. Eberhart, "A DISCRETE BINARY VERSION OF THE PARTICLE SWARM ALGORITHM," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, 1997, pp. 4–8.

[2] S. N. Langazane and A. K. Saha, "A Comparative Review of Current Optimization Algorithms for Maximizing Overcurrent Relay Selectivity and Speed," *IEEE Access*, vol. 12, no. March, pp. 53205–53223, 2024, doi: 10.1109/ACCESS.2024.3387704.

[3] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," *Proc. IEEE Conf. Evol. Comput. ICEC*, vol. 1, pp. 94–100, 2001, doi: 10.1109/cec.2001.934376.

[4] S. Yuhui and E. Russel, "A Modified Particle Swarm Optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence*, 1998, doi: 10.1109/ICEC.1998.699146.

[5] S. Saxena, "A New Non Linear Inertia Weight Approach in PSO for Faster Rigid Image Registration," *2019 6th Int. Conf. Signal Process. Integr. Networks*, pp. 607–612, 2019.

[6] J. Guan and W. Zhang, "Improved Topological Optimization Method Based on Particle Swarm Optimization Algorithm," *IEEE Access*, vol. 10, pp. 52067–52074, 2022, doi: 10.1109/ACCESS.2022.3174602.

[7] B. Borowska, "Nonlinear Inertia Weight in Particle Swarm Optimization," in *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, 2017, vol. 729, pp. 5–8, doi: 10.1109/STC-CSIT.2017.8098790.

[8] C. Huang, Y. Zhao, M. Zhang, and H. Yang, "APSO : An A ∗ -PSO Hybrid Algorithm for Mobile Robot Path Planning," *IEEE Access*, vol. 11, no. May, pp. 43238–43256, 2023, doi: 10.1109/ACCESS.2023.3272223.

[9] Y. Zhang, Y. Zhao, X. Fu, and J. Xu, "A feature extraction method of the particle swarm optimization algorithm based on adaptive inertia weight and chaos optimization for Brillouin scattering spectra," *Opt. Commun.*, vol. 376, pp. 56–66, 2016, doi: 10.1016/j.optcom.2016.04.049.

[10] D. E. Ratnawati, M. Marjono, W. Widodo, and S. Anam, "PSO-ELM with Time-varying Inertia Weight for Classification of SMILES Codes," *Int. J. Intell. Eng. Syst.*, vol. 13, no. 6, pp. 522–532, 2020, doi: 10.22266/ijies2020.1231.46.

[11] K. K. Bharti and P. K. Singh, "Opposition chaotic fitness mutation based adaptive inertia weight BPSO for feature selection in text clustering," *Appl. Soft Comput. J.*, pp. 1–15, 2016, doi: 10.1016/j.asoc.2016.01.019.

[12] J. Qiao, S. Li, M. Liu, Z. Yang, J. Chen, and P. Liu, "OPEN A modified particle swarm optimization algorithm for a vehicle scheduling problem with soft time windows," *Sci. Rep.*, pp. 1–18, 2023, doi: 10.1038/s41598-023-45543-z.

[13] J. Pepe, B. Mapetu, Z. Chen, and L. Kong, "Low-time complexity and low-cost binary particle swarm optimization algorithm for task scheduling and load balancing in cloud computing," *Springer Sci.*, 2019, doi: /doi.org/10.1007/s10489-019-01448-x Low-time.

[14] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[15] T. M. Shami, A. A. El-saleh, M. Alswaitti, and S. Mirjalili, "Particle Swarm Optimization : A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 10031–10061, 2022, doi: 10.1109/ACCESS.2022.3142859.

[16] T. Shaqarin and B. Noack, "A Fast Converging Particle Swarm Optimization through," *Multidiscip. Digit. Publ. Inst.*, vol. 1, pp. 1–13, 2022.

[17] A. Pradhan and S. K. Bisoy, "A novel load balancing technique for cloud computing platform based on PSO," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 7, pp. 3988–3995, 2022, doi: 10.1016/j.jksuci.2020.10.016.

[18] D. Saxena and A. K. Singh, "Communication Cost Aware Resource Efficient Load Balancing (CARE-LB) Framework for Cloud Datacenter Communication Cost Aware Resource Efficient Load Balancing (CARE- LB) Framework for Cloud Datacenter," *Recent Av. Comput. Sci. Commun.*, no. April 2021, 2020, doi: 10.2174/2666255813999200818173107.

[19] T. Alfakih, M. M. Hassan, and M. Al-razgan, "Multi-Objective Accelerated Particle Swarm Optimization With Dynamic Programing Technique for Resource Allocation in Mobile Edge Computing," *IEEE Access*, vol. 9, pp. 167503–167520, 2021, doi: 10.1109/ACCESS.2021.3134941.

[20] A. G. Gad, *Particle Swarm Optimization Algorithm and Its Applications : A Systematic Review*, vol. 29, no. 5. Springer Netherlands, 2022.

[21] X. Zhang and D. Zou, "A Novel Simple Particle Swarm Optimization Algorithm for Global Optimization," *MDPI Math.*, 2018, doi: 10.3390/math6120287.

[22] E. Algorithms, M. K. Kakkar, and J. Singla, "Performance comparison of genetic algorithms and particle swarm optimization for model integer programming bus timetabling problem Performance comparison of genetic algorithms and particle swarm optimization for model integer programming bus timetabling p," in *IOP Conference Series: Materials Science and Engineering*, 2018, doi: 10.1088/1757-899X/332/1/012020.

[23] H. T. Liang and F. H. Kang, "Adaptive Mutation Particle Swarm Algorithm with Dynamic Nonlinear Changed Inertia Weight," *Opt. - Int. J. Light Electron Opt.*, 2016, doi: 10.1016/j.ijleo.2016.06.002.

[24] S. Barakat, A. I. Osman, E. Tag-eldin, A. A. Telba, H. M. Abdel, and M. M. Samy, "Achieving green mobility : Multi-objective optimization for sustainable electric vehicle charging," *Energy Strateg. Rev.*, vol. 53, no. December 2023, p. 101351, 2024, doi: 10.1016/j.esr.2024.101351.

[25] T. B. Nkwanyana and Z. Wang, "Improved Particle Swarm Optimization Base on the Combination of Linear Decreasing and Chaotic Inertia Weights," in *12th International Conference on Computational Intelligence and Communication Networks*, 2021, no. 1, pp. 460–465, doi: 10.1109/CICN.2020.82.

[26] P. Aksornsri and S. Wongsa, "Valve Stiction Quantification using Particle Swarm Optimisation with Linear Decrease Inertia Weight," in *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2016, pp. 0–5.

[27] R. Grewal, "Modified Pso Algorithm With Non-Linearly Decreasing Inertia Weight," *Int. J. Res. Inf. Sci. Appl. Tech.*, vol. 5, no. 8, pp. 21586–215815, 2021, doi: 10.46828/ocpu/ijrisat/21586.

[28] M. Lin, Z. Wang, and F. Wang, "Hybrid Differential Evolution and Particle Swarm Optimization Algorithm Based on Random Inertia Weight," *IEEE*, pp. 411–414, 2019.

[29] L. Zhang, Y. Tang, C. Hua, and X. Guan, "A new particle swarm optimization algorithm with adaptive inertia weight based on Bayesian techniques," *Appl. Soft Comput. J.*, vol. 28, pp. 138–149, 2015, doi: 10.1016/j.asoc.2014.11.018.

[30] M. Taherkhani and R. Safabakhsh, "A novel stability-based adaptive inertia weight for particle swarm optimization," *Appl. Soft Comput. J.*, pp. 1–15, 2015, doi: 10.1016/j.asoc.2015.10.004.

[31] X. Gu, M. Huang, and X. U. Liang, "A Discrete Particle Swarm Optimization Algorithm With Adaptive Inertia Weight for Solving Multiobjective Flexible Job-shop Scheduling Problem," *IEEE Access*, vol. 8, pp. 33125–33136, 2020, doi: 10.1109/ACCESS.2020.2974014.

[32] Y. Shi, "Optimization of PID Parameters of Hydroelectric Generator Based on Adaptive Inertia Weight PSO," in *IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC 2019)*, 2019, no. Itaic, pp. 1854–1857.

[33] A. K. Shukla, P. Singh, and M. Vardhan, "An adaptive inertia weight teaching-learning-based optimization algorithm and its applications," *Elsevier*, vol. 77, pp. 309–326, 2020, doi: 10.1016/j.apm.2019.07.046.

[34] T. S. Li and P. Kuo, "Intelligent Control Strategy for Robotic Arm by Using Adaptive Inertia Weight and Acceleration Coefficients Particle Swarm Optimization," *IEEE Access*, vol. 7, pp. 126929–126940, 2019, doi: 10.1109/ACCESS.2019.2939050.

[35] C. Guimin, M. Zhengfeng, J. Jianyuan, and X. Huang, "Self-active inertia weight strategy in particle swarm optimization algorithm," *Proc. World Congr. Intell. Control Autom.*, vol. 1, no. 2002, pp. 3686–3689, 2006, doi: 10.1109/WCICA.2006.1713058.

[36] C. Yang, W. Gao, N. Liu, and C. Song, "Low-discrepancy sequence initialized particle swarm optimization algorithm with high-order nonlinear time-varying inertia weight," *Appl. Soft Comput. J.*, vol. 29, pp. 386–394, 2015, doi: 10.1016/j.asoc.2015.01.004.

[37] R. Swathy, B. Vinayagasundaram, G. Rajesh, and A. Nayyar, "Game theoretical approach for load balancing using SGMLB model in cloud environment," *PLoS One*, pp. 1–22, 2020, doi: 10.1371/journal.pone.0231708.

[38] F. T. Johora, I. Ahmed, A. I. Shajal, and R. Chowdhory, "A load balancing strategy for reducing data loss risk on cloud using remodified throttled algorithm," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 3, pp. 3217–3225, 2022, doi: 10.11591/ijece.v12i3.pp3217-3225.

[39] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, "Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing," *Proc. - Conf. Local Comput. Networks, LCN*, pp. 17–24, 2016, doi: 10.1109/LCN.2016.024.

[40] J. Chrouta, F. Farhani, A. Zaafouri, and M. Jemli, "Comparing inertia weights in Multi-swarm Particle Swarm Optimization," *IEEE Xplore*, pp. 278–283, 2019.

[41] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia Weight Strategies in Particle Swarm Optimization," *IEEE*, pp. 633–640, 2011.