Hindawi
Computational Intelligence and Neuroscience
Volume 2021, Article ID 8819333, 18 pages
https://doi.org/10.1155/2021/8819333

Research Article

UCPSO: A Uniform Initialized Particle Swarm Optimization Algorithm with Cosine Inertia Weight

Jian Zhang[1], **Jianan Sheng**[1], **Jiawei Lu**[1] and **Ling Shen**[2]

[1]*School of Mechanical Engineering, Tongji University, Shanghai 200092, China*
[2]*Shanghai University of Medicine & Health Sciences, Shanghai 201318, China*

Correspondence should be addressed to Jian Zhang; jianzh@tongji.edu.cn

Received 6 July 2020; Revised 8 February 2021; Accepted 2 March 2021; Published 19 March 2021

Academic Editor: Paolo Gastaldo

The particle swarm optimization algorithm (PSO) is a meta-heuristic algorithm with swarm intelligence. It has the advantages of easy implementation, high convergence accuracy, and fast convergence speed. However, PSO suffers from falling into a local optimum or premature convergence, and a better performance of PSO is desired. Some methods adopt improvements in PSO parameters, particle initialization, or topological structure to enhance the global search ability and performance of PSO. These methods contribute to solving the problems above. Inspired by them, this paper proposes a variant of PSO with competitive performance called UCPSO. UCPSO combines three effective improvements: a cosine inertia weight, uniform initialization, and a rank-based strategy. The cosine inertia weight is an inertia weight in the form of a variable-period cosine function. It adopts a multistage strategy to balance exploration and exploitation. Uniform initialization can prevent the aggregation of initial particles. It distributes initial particles uniformly to avoid being trapped in a local optimum. A rank-based strategy is employed to adjust an individual particle's inertia weight. It enhances the swarm's capabilities of exploration and exploitation at the same time. Comparative experiments are conducted to validate the effectiveness of the three improvements. Experiments show that the UCPSO improvements can effectively improve global search ability and performance.

1. Introduction

Since the particle swarm optimization algorithm (PSO) was proposed by Kennedy and Eberhart in 1995 [1], it has obtained great achievements in finding the optimal value of continuous nonlinear equations [2]. PSO is a special branch of evolutionary algorithms. It adopts social learning among swarms and the self-cognition of individuals to replace the common theory of evolution algorithms (EAs)-survival of the fittest [3]. The bionics mechanism of PSO gives PSO swarm intelligence [4] and enables PSO to imitate the complex search behaviour of swarms, such as bird swarms, fish schools, and ant colonies. Different from EAs, PSO has a simpler iteration mechanism and fewer control parameters [5]. Therefore, it is widely used in practical engineering areas. For example, PSO is usually applied to image processing [6], parameter optimization [7, 8], scheduling optimization [9], clustering [10], and price forecasting [11].

One of the main advantages of PSO is its easy implementation [12]. A large number of numerical experiments also prove that PSO has high convergence accuracy and a fast convergence speed [13]. Consequently, researchers have carried out numerous studies on PSO. However, some limitations of PSO have been found during long-term research work. When facing complex functions, PSO is troubled by falling into a local optimum or premature convergence [14]. Meanwhile, the performance of the original PSO is inadequate and must be improved. To solve these problems, many researchers have proposed improvements. These improvements mainly focus on PSO parameters, particle initialization, and population topology.

Inertia weight is a very important parameter in the PSO algorithm. It controls the balance between the two critical behaviours of PSO: global and local search. Researchers have created various forms of inertia weight. These forms of inertia weight enhance the performance of PSO. Shi and

Eberhart proposed a parameter called inertia weight ω in PSO to balance exploration and exploitation [15]. The appearance of ω created a new way to improve the performance of PSO. Then, Eberhart and Shi introduced a linear decreasing inertia weight [16]. This linear decrease in inertia weight greatly improves the comprehensive performance of PSO. It relieves the problem of falling into a local optimum. Recently, Tian et al. adopted a multistage strategy to refine the change process of inertia weight. They split the curve of inertia weight into two stages to satisfy specific requirements. The variant proposed by them achieves excellent performance in the experiment [17].

The quality of the initial particles is related to the PSO results. Many works have been performed to distribute initial particles more dispersedly and make initial particles closer to the global optimum. Tian [18], Zhang [19], and Xu [20] adopted chaotic sequences for particle initialization to increase the diversity of initial particles. Chaotic initialization achieves certain success compared to random initialization under the same conditions. Rahnamayan [21] employed the symmetry strategy in swarm initialization. Symmetry initialization can prevent initial particles from being distant from the global optimum. DMPSO [22] combines chaotic initialization with opposition-based initialization. Experiments validate that the hybrid initialization can recognize the search area better. In addition, MCJPSO [23] randomly divides the entire search space and distributed particles over a search space in independent slots. This semirandom initialization can overcome the limitation of the original PSO. Rauf et al. used the Weibull probability sequence to generate numbers at random locations for swarm initialization. This method is able to enhance the diversity of swarms [24].

To enhance the global search ability and the comprehensive performance of PSO, a uniform initialized particle swarm optimization algorithm with cosine inertia weight (UCPSO) is proposed in this paper. UCPSO combines three effective improvements: an inertia weight in the form of a variable-period cosine function, uniform initialization, and a rank-based strategy for individual particle inertia weights. The cosine inertia weight introduced in this paper adopts the multistage strategy. It divides the change process of inertia weight into three stages. It can balance exploration and exploitation more specifically, help particles transform from global search to local search smoothly, and improve the convergence accuracy. Uniform initialization initializes a particle randomly as the basic point and then generates other initial particles based on this basic point. The initial particles are evenly distributed in each dimension, and the positions of each particle in each dimension are random. This mechanism can prevent the aggregation of initial particles. It distributes particles uniformly to recognize the search area more comprehensively. Uniform initialization is able to avoid falling into a local optimum and improve the search efficiency. In addition, this paper employs a rank-based strategy to adjust individual particle inertia weights. It makes the particles that are close to the swarm's best position focus on mining and makes particles that are far away from the swarm's best position keep exploring. It can enhance the global and local search ability of swarms at the same time.

In recent years, researchers have proposed some effective variants of PSO. Ye et al. proposed an improved multiswarm particle swarm optimization with dynamic learning strategy (PSO-DLS). It classified particles of each subswarm into ordinary particles and communication particles [25]. Lynn and Suganthan proposed the ensemble particle swarm optimizer (EPSO), which combines the characteristics of several PSO variants [26]. In EPSO, the best-performing algorithm for each generation can be determined by a self-adaptive scheme. Heterogeneous comprehensive learning particle swarm optimization (HCLPSO) divides the whole swarm into an exploration subpopulation and an exploitation subpopulation [27]. The CL strategy is used to breed learning exemplars for both of them. Gong et al. proposed genetic learning particle swarm optimization (GLPSO), which adopts selection, mutation, and selection [28]. By performing these operators on the historical information of particles, GLPSO is able to construct diversified and high-qualified learning exemplars to guide the swarm.

The purpose of designing UCPSO is to obtain a variant of PSO that has a good comprehensive performance and the ability to escape from a local optimum. In addition, three improvements in UCPSO should be easy to use. They are introduced to help researchers improve the global search ability and performance of PSO. A large number of comparative experiments based on benchmark functions were used to validate the effectiveness of the UCPSO improvements.

This paper is organized as follows. Section 2 introduces the standard PSO and related research on inertia weight and particle initialization. Section 3 describes the UCPSO and the three improvements in detail. Experiments are presented in Section 4. The conclusion is given in the Section 5.

2. Particle Swarm Optimization Algorithms

2.1. Standard PSO. PSO is a stochastic algorithm based on population [15]. It finds the optimal solution in a given range by mimicking the behaviour of birds. The particles in the swarm are potential solutions, and n is the total number of particles in the swarm. Every particle remembers its current position X_i, its current velocity V_i, and the best position that it has ever been $P\text{best}_i\,(1 \le i \le n)$. The swarm also remembers the swarm's best position $G\text{best}$. The $X_i, V_i, P\text{best}_i, G\text{best}$ are all D-dimensional vectors, $X_i = [x_{i1}, x_{i2}, \ldots, x_{id}, \ldots, x_{iD}]$, $V_i = [v_{i1}, v_{i2}, \ldots v_{iD}]$, $P\text{best}_i = [p\text{best}_{i1}, p\text{best}_{i2}, \ldots, p\text{best}_{id}, \ldots, p\text{best}_{iD}]$, $G\text{best} = [g\text{best}_1, g\text{best}_2, \ldots, g\text{best}_d, \ldots, g\ \text{best}_D]\,(1 \le d \le D)$. Particles find the optimal solution through iterations. The position and velocity of particles are updated as follows:

$$\begin{aligned} v_{id}(t+1) = {} & \omega \times v_{id}(t) + c_1 \times r_1 \\ & \times \left[p\text{best}_{id} - x_{id}(t)\right] + c_2 \\ & \times r_2 \times \left[g\text{best}_d - x_{id}(t)\right], \end{aligned} \tag{1}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \tag{2}$$

where t denotes the current iteration, $t \le t_{\max}$. In addition, $x_{id} \in [x_{d\min}, x_{d\max}]$ and $v_{id} \in [v_{d\min}, v_{d\max}]$. ω is the inertia weight. $c1$ and $c2$ are acceleration coefficients, which control the influence of $P\text{best}_i$, and $G\text{best}$ in the iteration. $r1$ and $r2$ are random numbers in the range 0, 1. PSO is usually terminated after reaching the allowed maximum number of iterations or meeting the stopping criterion. The best solution of a problem is the final $G\text{best}$. Figure 1 shows the concept of a particle's iteration in a graphical way.

In equation (1), $\omega \times v_{id}(t)$ represents the effect of inertia, $c_1 \times r_1 \times [p\text{best}_{id} - x_{id}(t)]$ represents the effect of self-cognition, and $c_2 \times r_2 \times [g\text{best}_d - x_{id}(t)]$ represents the effect of social learning. The cooperation between them contributes to finding the optimal solution. The function used to evaluate the position of a particle is usually called the fitness function $F(X_i)$. To prevent particles from exceeding the search area, the position and velocity of a particle are always limited in the allowed range. When a particle reaches the boundary of the search area, its velocity should be reversed to improve the search efficiency. The pseudocode of the standard PSO is shown as follows (Algorithm 1):

2.2. Different Forms of Inertia Weight. The inertia weight reflects the influence of the previous velocity $V_i(t)$ on the new velocity $V_i(t+1)$. A large inertia weight can prevent particles from going to the region of interest (hereafter called the ROI) immediately. This makes particles continue to search outside the ROI for a period of time. A small inertia weight can make particles go to the ROI immediately and search in the ROI. That is, a large inertia weight enhances the global search capability (hereafter called exploration), and a small inertia weight enhances the local search capability (hereafter called exploitation). Exploration can prevent particles from falling into a local optimum, but it also leads to low convergence accuracy and a slow convergence speed. Exploitation can accelerate the convergence speed and improve the convergence accuracy, but it makes the algorithm converge prematurely or become trapped in a local optimum easily [29]. These two functions greatly influence the performance of PSO. Therefore, it is very important to choose an appropriate inertia weight.

Since inertia weight was proposed, many researchers have made contributions in this field. Some classical forms of inertia weight have been proposed, such as time invariant [15], linear time variant [16, 30], nonlinear time variant [17, 31], and other forms of inertia weight [32–34]. The famous forms of inertia weight mentioned above are described in detail in the subsections below.

2.2.1. Time Invariant Inertia Weight. To improve the performance of the original PSO, Shi and Eberhart proposed a parameter called inertia weight ω to balance exploration and exploitation in 1998 [15]. First, the inertia weight appeared in the form of a constant. They found that a large inertia weight facilitates exploration, while a small inertia weight facilitates exploitation. The recommended range of inertia weight is [0.9, 1.2]. The computational results showed that the overall performance of PSO was improved empirically:

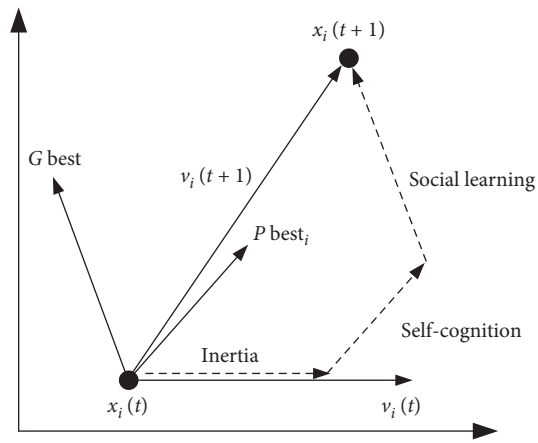

Figure 1: The concept of a particle's iteration.

$$\omega_0(t) = \text{constant}, \tag{3}$$

$\omega_0(t)$ is easy to implement, so it has been widely used.

2.2.2. Linear Time Variant Inertia Weight. After the concept of inertia weight was proposed, a linear time variant inertia weight was introduced in [16] to further improve the performance of the PSO algorithm. The mechanism of the linear time variant inertia weight $\omega_1(t)$ is shown in the following equation:

$$\omega_1(t) = \omega_{\text{ini}} - \frac{t}{t_{\max}}(\omega_{\text{ini}} - \omega_{\text{fin}}), \quad \omega_{\text{ini}} > \omega_{\text{fin}}. \tag{4}$$

The initial value ω_{ini} to the final value ω_{fin} as the number of iterations increases. The linear time variant inertia weight takes the demands of particles in different periods into account. The recommended values of ω_{ini} and ω_{fin} are $\omega_{\text{ini}} = 1.4$ and $\omega_{\text{fin}} = 0$.

There are many other mechanisms of the linear time variant inertia weight. Specifically, Zheng proposed an increasing linear time variant inertia weight $\omega_2(t)$ ($\omega_{\text{ini}} < \omega_{\text{fin}}$). It is proven that the increasing mechanism performs better than the decreasing mechanism in some test functions [30]. The mechanism of inertia weight $\omega_2(t)$ is

$$\omega_2(t) = \omega_{\text{ini}} + \frac{t}{t_{\max}}(\omega_{\text{fin}} - \omega_{\text{ini}}), \quad \omega_{\text{ini}} < \omega_{\text{fin}}. \tag{5}$$

2.2.3. Nonlinear Time Variant Inertia Weight. Based on the linear time variant inertia weight, some researchers think that the nonlinear mechanism is more suitable for the demand of particles. Therefore, many nonlinear time variant mechanisms have been proposed. Chatterjee [31] introduced a nonlinear time variant inertia weight combined with a quadratic function, and its mechanism is

$$\omega_3(t) = \omega_{\text{ini}} + \left(\frac{t}{t_{\max}}\right)^2 (\omega_{\text{ini}} - \omega_{\text{fin}}). \tag{6}$$

For the sake of a better inertia weight, some researchers abandoned the continuous function and began to research

```
(1) Initialize n, c1, c2, ω, particles' position X and velocity V, then find Pbest and Gbest, t = 1;
(2) while (t ≤ t_max or the precision is not met)
(3)    for i = 1 : n
(4)       for d = 1 : D
(5)          Update velocity v_id by equation (1);
(6)          Update position x_id by equation (2);
(7)       End
(8)       if F(X_i) < F(Pbest_i)
(9)        Pbest_i = X_i;
(10)        if F(Pbest_i) < F(Gbest)
(11)           Gbest = Pbest_i;
(12)        End
(13)      End
(14)   End
(15)   t = t + 1;
(16) End
```

Algorithm 1: Pseudocode of the standard PSO.

the multiple stages of inertia weight. Tian [17] proposed a sigmoid increasing inertia weight $\omega_4(t)$ and obtained an algorithm with satisfactory performance. The mechanism of inertia weight $\omega_4(t)$ is as follows:

$$\omega_4(t) = \begin{cases} \omega_{\text{ini}}, & t \le 0.2 \times t_{\max}, \\ \dfrac{1}{\left(1 + e^{\left((10t - 2t_{\max})/t_{\max}\right)}\right)} + \omega_{\text{fin}}, & \text{otherwise.} \end{cases} \tag{7}$$

2.2.4. Other Forms of Inertia Weight. Some researchers also proposed other effective strategies to adjust the inertia weight, such as random strategy, chaotic strategy, and self-adaptive strategy.

Randomness is the natural property of PSO, and it is also the reason why PSO can be applied to almost all optimization problems. It is difficult to predict whether exploration or exploitation would be better during the iteration. To address this problem, researchers thought of using random strategies to adjust the inertia weight. A random inertia weight $\omega_5(t)$ was introduced in [32]. The mechanism of inertia weight $\omega_5(t)$ is shown in

$$\omega_5(t) = 0.5 + \frac{\text{rand}}{2}, \tag{8}$$

where rand is a random number in the range [0, 1], so $0.5 \le \omega_5 \le 1$.

Feng [33] used a chaotic strategy to adjust the inertia weight and obtain a chaotic inertia weight $\omega_6(t)$. He added a chaotic term $z(t)$ to the linearly decreasing inertia weight. The mechanism of inertia weight $\omega_6(t)$ is

$$\omega_6(t) = \left(\omega_{\text{ini}} - \omega_{\text{fin}}\right)\frac{t_{\max} - t}{t_{\max}} + \omega_{\text{ini}} \times z(t), \tag{9}$$

$$z(t+1) = 4z(t)(1 - z(t)), \tag{10}$$

where the initial value of $z(t)$ is a random number in the range [0, 1], and $z(t) \neq 0, 0.25, 0.5, 0.75, 1$.

Different from the random strategy and chaotic strategy, some researchers chose an index to adjust the value of inertia weight in real time. This kind of index can provide feedback on the state of the swarm. Zhang [34] et al. adopted an index φi to monitor the state of each particle, and they proposed a self-adjusted inertia weight $\omega_7(t)$. The mechanism of inertia weight $\omega_7(t)$ is shown in the following equations:

$$\varphi_i(t+1) = \left|\frac{\text{Gbest}(t) - X_i(t)}{\text{Pbest}_i(t) - X_i(t)}\right|, \tag{11}$$

$$\omega_7(t+1) = \frac{\omega_{\text{ini}} - \omega_{\text{fin}}}{1 + e^{\varphi(t+1) \times \left(t - \left((1 + \ln(\varphi(t+1))) \times t_{\max}\right)/\mu\right)}} + \omega_{\text{fin}}, \tag{12}$$

where $\mu = 100$. On the right side of equation (11), the numerator is the Euclidean distance from the position of the i^{th} particle to *G*best, and the denominator is the Euclidean distance from the position of the i^{th} particle to *P*best$_i$. Therefore, the index φ_i can reflect the state of the i^{th} particle dynamically. The performance of the self-adaptive inertia weight in some test functions is excellent, but the mechanism of the self-adaptive inertia weight is usually very complicated. It is difficult to design a widely used index.

2.3. Particle Initialization. In addition, the quality of particle initialization is also critical to the performance of the PSO algorithm. The volatility of particle initialization is the primary cause of the volatility of convergence speed and accuracy.

2.3.1. Random Initialization. The original method of particle initialization is random initialization. The position of each dimension of each particle is distributed in the allowed range independently and randomly, as shown in the following equation:

$$x_{id} = \text{rand} \times (x_{d\max} - x_{d\min}) + x_{d\min}. \quad (13)$$

After initialization, if particles are close to the global optimum, PSO tends to have good performance; if the particles are concentrated near the local optimum, PSO tends to fail. The random strategy of particle initialization inevitably leads to volatility of initialization. However, if particle initialization abandons randomness, it is very difficult for PSO to solve various optimization problems without prior knowledge. Fixed initialization can only solve specific problems.

2.3.2. Chaotic Initialization. Chaotic initialization employs chaotic sequences to make particles more scattered. A common chaotic sequence called a logistic map is widely used because of its simple employment [35]. Its mechanism is as follows:

$$Z_N = a \times Z_{N-1} \times (1 - Z_{N-1}), \quad (14)$$

where $Z_N\,(1 \le N \le n)$ is the chaotic variable, Z_1 is a random value in the range 0, 1, and other chaotic variables are obtained by equation (14). To prevent chaotic variables from falling into a cycle, $Z_1 \ne 0$, 0.25, 0.5, 0.75, and 1. a is a constant that controls the level of chaos, and the recommended range for a is [3.5699, 4]. In this paper, Z_N is obtained by equation (14) for 10 iterations to make the initial swarm more chaotic. After $n \times D$ chaotic variables are obtained, the chaotic initialization can be completed by replacing the random matrix with chaotic variables during the process of initialization.

2.3.3. Opposition-Based Initialization. For two particles that are symmetrical about the centre of the search area, one particle of the two is closer to the global optimum than the other (it is a special case that the two distances are equal). Opposition-based initialization generates a subswarm randomly and combines it with its symmetric subswarm. Therefore, opposition-based initialization can avoid the situation in which all particles are far from the global optimum. Rahnamayan [21] introduced opposition-based initialization, and the mechanism of opposition-based initialization is shown in

$$x_{id} = (x_{d\max} + x_{d\min}) - x_{(n+1-i)d} \quad (15)$$

3. UCPSO Algorithm

Based on research on inertia weight and particle initialization, UCPSO is proposed to be a competitive variant of PSO. UCPSO adopts three new strategies, and their details are represented in the following sections.

3.1. Inertia Weight in the Form of Variable-Period Cosine Function. There is a popular form of nonlinear time variant inertia weight. It maintains a large value in the early stage and keeps a small value in the final stage. In common optimization problems, it can enhance the global search ability of PSO. Then, the multistage inertia weight was proposed. It puts forward more specific requirements for the change process of inertia weight: (a) initial stage: inertia weight keeps a large value for a period of time to carry out global search and reduces the probability of falling into local optimum (this stage is also called global search stage); (b) intermediate stage: inertia weight drops rapidly and transits from global search to local search (this stage is also called decelerating transition stage); (c) the final stage: inertia weight keeps a small value for a long time to help PSO converge to an accurate optimal solution quickly (this stage is also called the local search stage).

The change process of the cosine function in the range $[0, \pi]$ meets the requirements of the multistage inertia weight. In the range $[0, \pi/6]$, the cosine function maintains a large value $(\le \sqrt{3}/2)$ and changes slowly. In the range $[\pi/6, 5\pi/6]$, the cosine function declines rapidly. In the range $[5\pi/6, \pi]$, the cosine function maintains a small value $(\le -\sqrt{3}/2)$ and changes slowly. Because the cosine function is simple and easy to use, an inertia weight in cosine form is adopted in this paper. However, the original cosine function is not consistent with the requirements of the multistage inertia weight. Therefore, the original cosine function needs to be adjusted. An iterative term $I(t)$ is added into the cosine function to adjust the period and $\omega_{\cos}(t)$ is rescaled in the range $[\omega_{\text{fin}}, \omega_{\text{ini}}]$ as shown in the following equations:

$$\omega_{\cos}(t) = \frac{(\omega_{\text{ini}} + \omega_{\text{fin}})}{2} + \frac{(\omega_{\text{ini}} - \omega_{\text{fin}})}{2} \times \cos\left(\frac{I(t)\pi}{t_{\max}}\right), \quad (16)$$

$$I(t+1) = I(t) + a, \quad I(1) = 0, \quad (17)$$

$$a = \begin{cases} a_1, & \dfrac{I(t) \le t_{\max}}{6}, \\ a_2, & \dfrac{t_{\max}}{6} < I(t) \le \dfrac{5t_{\max}}{6}, \\ a_3, & \dfrac{5t_{\max}}{6} < \mathrm{I}(t) \le t_{\max}, \end{cases} \quad (18)$$

where a is a constant that can adjust the period of $\omega_{\cos}(t)$. The values of a_1, a_2, and a_3 can control the length of each stage in $\omega_{\cos}(t)$ According to the requirements of $\omega_{\cos}(t)$, we limit the phase $(I(t)\pi/t_{\max})$ in the range 0, π and $(I(t)\pi/t_{\max})$ is required to increase from 0 to π. While t increases from 0 to $t_{\max} I(t)$ needs to increase from 0 to $t_{\max}$. Therefore, a_1, a_2, and a_3 have to satisfy the following equation:

$$\frac{1}{6a_1} + \frac{2}{3a_2} + \frac{1}{6a_3} = 1. \quad (19)$$

The pseudocode of updating $\omega_{\cos}$ is shown as follows (Algorithm 2):

The parameter analysis experiment for a_1, a_2, and a_3 is shown in Section 4.2. The recommended configuration of a_1, a_2, and a_3 is $a_1 = (4/3), a_2 = (16/3)$, and $a_3 = (2/9)$. The curves of different inertia weights $\omega_0(t) \sim \omega_7(t)$ and $\omega_{\cos}(t)\,(a_1 = (4/3), a_2 = (16/3), a_3 = (2/9))$ are displayed in

```
(1) Let a = a1;
(2) for t = 1 : tmax
(3)     Update ωcos (t) by equation (16);
(4)     if I (t) ≥ (5tmax/6)
(5)         a = a3;
(6)     else if I (t) ≥ (tmax/6)
(7)         a = a2;
(8)     end
(9)     I (t + 1) = I (t) + a;
(10) End
```

Algorithm 2: Pseudocode of updating $\omega_{\cos}(t)$.

Figures 2–4. In addition, their parameters are also illustrated.

To make the inertia weight have the same value range, in these inertia weights, all ω_{ini} are set to 0.9, and all ω_{fin} are set to 0.4 (for $\omega_2(t)$, $\omega_{\text{ini}} = 0.4$, $\omega_{\text{fin}} = 0.9$), $\omega_0(t) = 0.9$, and $\omega_5(t) = 0.4 + \text{rand}/2$.

3.2. Uniform Initialization. There is no clear mechanism in random initialization, chaotic initialization, and opposition-based initialization to avoid the aggregation of initial particles. This situation will lead some areas to be searched repeatedly and some areas to be ignored. This reduces the search efficiency and the possibility of finding a global optimum.

To solve this problem, a particle initialization method with both randomness and uniformity (called uniform initialization) is proposed in this paper. In Algorithm 3, Line 1 initializes a particle randomly to be the base point, and $X_1 = [X_{11}, X_{12}, \ldots, X_{1D}]$ is the position of the base point. Lines 2–4 generate a $D \times (n-1)$ random matrix $R = [R_1, R_2, \ldots, R_D]$ to ensure the randomness of the initial particles. Lines 5–12 divide the length of each dimension by n to obtain the minimum distance between particles in the corresponding dimension. Lines 5–12 distribute particles in each dimension uniformly and avoid the aggregation of particles. If the position of a particle exceeds the allowed range of a dimension, it will subtract the range of the corresponding dimension. The pseudocode of uniform initialization is shown as follows:

Uniform initialization ensures that the distances between particles are larger than a certain value. It can avoid the aggregation of particles and distribute initial particles uniformly to recognize more areas at the beginning. Figure 5 represents the result of uniform initialization. We find that the level of aggregation is low, and the distribution of particles is uniform. Uniform initialization is a good combination of randomness and uniformity. Figure 5 is completed under the configuration: $n = 50, D = 2$, the search area is a 4×4 rectangle.

3.3. Rank-Based Strategy for Individual Particle's Inertia Weight (RIW). The forms of inertia weight mentioned above are all assigned numerical values according to the state of the whole swarm. However, in fact, particles' states are diverse.

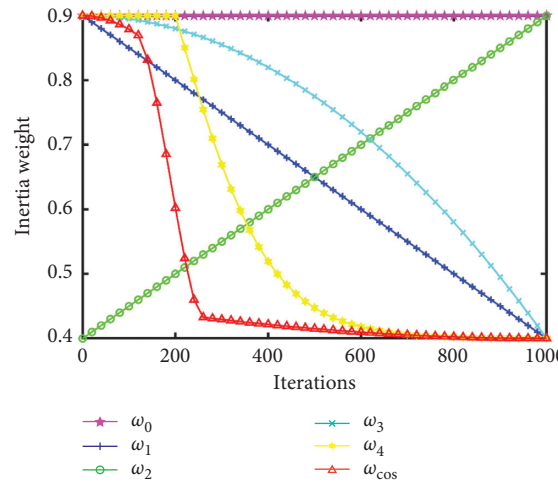

Figure 2: Curves of inertia weight $\omega_0(t) \sim \omega_4(t)$ and $\omega_{\cos}(t)$.

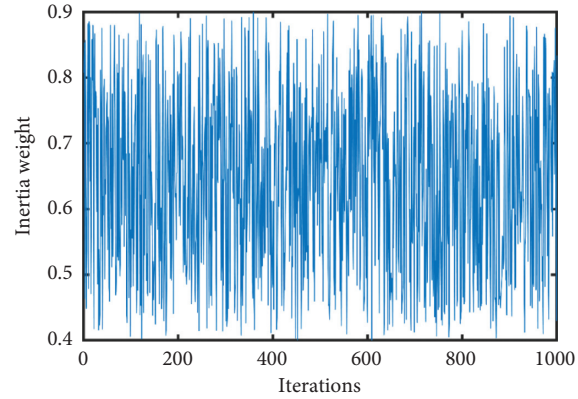

Figure 3: Curve of inertia weight $\omega_5(t)$.

The individual particle's need may not follow the swarm's need. Particles that are already in the ROI need a small inertia weight to exploit. Particles that are far away from the ROI need a large inertia weight to explore. The single value of inertia weight cannot satisfy both requirements at the same time. Cooperation between these two kinds of particles can maximize the benefit of the whole swarm.

A rank-based strategy is adopted by this paper to solve the problem. Generally, the particles with small fitness values

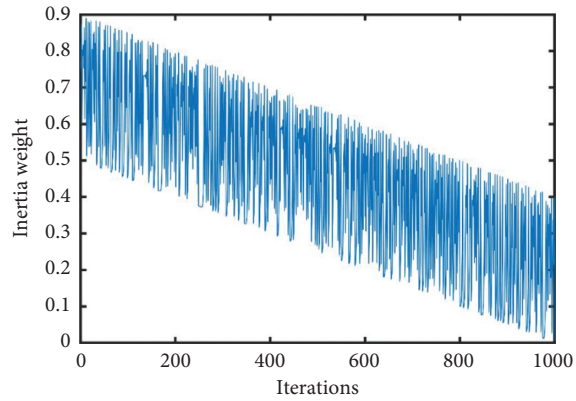

Figure 4: Curve of inertia weight $\omega_6(t)$.

(1) Initialize X_1 in the search area randomly by equation (13);
(2) **for** $d = 1 : D$
(3) Randomly rearrange $[1, 2, \ldots, (n-1)]$ to get $R_d = [R_{1d}, R_{2d}, \ldots, R_{(n-1)d}]$;
(4) **End**
(5) **for** $i = 1 : n$
(6) **for** $d = 1 : D$
(7) $X_{id} = X_{1d} + R_{(i-1)d} \div n \times (X_{d\max} - X_{d\min})$;
(8) **if** $X_{id} > X_{d\max}$
(9) $X_{id} = X_{id} - (X_{d\max} - X_{d\min})$;
(10) **End**
(11) **End**
(12) **End**

Algorithm 3: Pseudocode of uniform initialization.

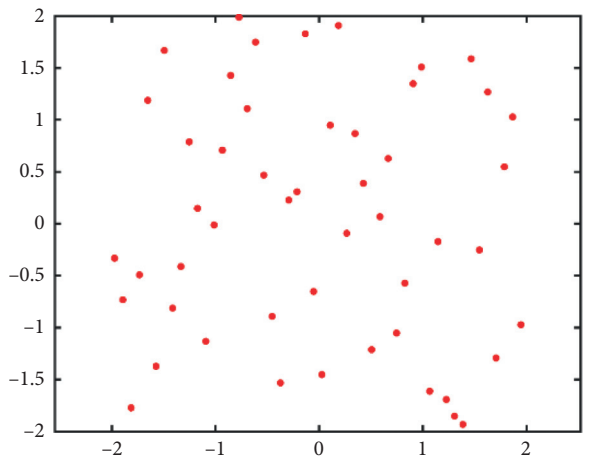

Figure 5: Results of the uniform initialization (2D).

$(F(X_i))$ are in the current ROI, while the particles with large $F(X_i)$ are outside the ROI. Particles are sorted by fitness value from small to large. Adding a rank-based strategy to inertia weight can take both the overall and individual requirements into account simultaneously. The mechanism of the RIWs is shown in equations (20) and (21):

$$\omega_i = b_i \times \omega_{\text{swarm}}, \tag{20}$$

$$b_i = \begin{cases} b_1, & \text{rank}_i \leq \dfrac{n}{4}, \\ b_2, & \text{otherwise}, \\ b_3, & \text{rank}_i \geq \dfrac{3n}{4}, \end{cases} \tag{21}$$

where ω_i denotes the i^{th} particle's inertia weight, ω_{swarm} denotes the swarm's inertia weight, b_i is the adjustment factor for the i^{th} particle's inertia weight, and rank_i denotes the ranking of the i^{th} particle according to the fitness value.

The pseudocode of the RIWs is shown as follows (Algorithm 4):

The parameter analysis experiment for b_1, b_3 is shown in Section 4.2. The recommended configuration of b_1, b_2, b_3 is $b_1 = (2/3), b_2 = 1, b_3 = 1.5$.

This paper adopts the above three mechanisms in the standard PSO and proposes a uniform initialized particle swarm optimization with cosine inertia weight (UCPSO). To elaborate the mechanism of UCPSO, the pseudocode of UCPSO is shown as follows (Algorithm 5):

```
(1) while (t ≤ t_max or the precision is not met)
(2)    Sort F(X) to get rank = [rank_1, rank_2, ..., rank_n];
(3)    for i = 1 : n
(4)       Update ω_swarm;
(5)       Update b_i by equation (21);
(6)       Update ω_i by equation (20);
(7)    End
(8)    t = t + 1;
(9) End
```

Algorithm 4: Pseudocode of RIWs.

```
(1) Initialize n, c_1, c_2, ω, particles' velocity V and initialize particles' position X by Algorithm 2, then find Pbest and Gbest, t = 1;
(2) Calculate ω_cos by Algorithm 3;
(3) while (t ≤ t_max or the precision is not met)
(4)    Sort F(X) to get rank = [rank_1, rank_2, ..., rank_n];
(5)    for i = 1 : n
(6)       Update b_i by equation (21);
(7)       ω = b_i × ω_cos(t);
(8)       for d = 1 : D
(9)          Update velocity v_id by equation (1);
(10)         Restrict v_id in [v_dmin, v_dmax];
(11)         Update position x_id by equation (2);
(12)         Restrict x_id in [x_dmin, x_dmax];
(13)         if x_id == x_dmin or x_id == x_dmax
(14)            v_id = −v_id;
(15)         End
(16)      end
(17)      if F(X_i) < F(Pbest_i)
(18)         Pbest_i = X_i;
(19)         if F(Pbest_i) < F(Gbest)
(20)            Gbest = Pbest_i;
(21)         End
(22)      End
(23)   End
(24)   t = t + 1;
(25) End
```

Algorithm 5: Pseudocode of UCPSO.

4. Experimental Results and Discussion

4.1. Experimental Setup. The experiments in Sections 4.2–4.4 are performed on benchmark functions $f_1 - f_6$ [36]. The details of $f_1 - f_6$ are specified in Table 1. $f_1 - f_6$ include 2 many-local-minima functions, 2 bowl-shaped functions, and 2 valley-shaped functions. Therefore, $f_1 - f_6$ are representative. The experiment to compare UCPSO with PSO, MCJPSO [23], PSO-DLS [25], EPSO [26], HCLPSO [27], and GLPSO [28] and is based on CEC2020 benchmark functions, as shown in Table 2. The parameter configurations of the other six algorithms are set according to their original references, which are shown in Table 3.

The nonparametric Wilcoxon signed-rank test is used to examine the significant difference between algorithms. In this article, a Wilcoxon signed-rank test at a 5% significance level is used. A pairwise comparison is conducted over the results obtained through several runs. The symbol "+" indicates that the proposed algorithm performs significantly better than the compared algorithms. The symbol "=" indicates that the proposed algorithm is not significantly different from the compared algorithm. The symbol "–" indicates that the compared algorithm performs significantly better than the proposed algorithm.

The criteria include the mean of the best solutions (Mean), the standard deviation of the best solutions (SD), success rate (SR) [37] and the average number of iterations (Average number). SR reflects the probability of obtaining a satisfactory result. To reduce the impact of extreme values, only the successful iterations are counted when calculating average number. When the algorithm iterates successfully, the current number of iterations will be recorded to calculate average number, and the algorithm will continue to iterate. Average number reflects the convergence speed effectively

Table 1: Commonly used benchmark functions f_1~f_6.

Benchmark function	Expression	D	Search range	$F(X^*)$
Rotated hyper-ellipsoid (10)	$f_1(x) = \sum_{i=1}^{10} \sum_{j=1}^{i} x_j^2$	10	$[-100, 100]^{10}$	0
Sphere (10)	$f_2(x) = \sum_{i=1}^{10} x_i^2$	10	$[-100, 100]^{10}$	0
Griewank (20)	$f_6(x) = \sum_{i=1}^{20} (x_i^2/4000) - \prod_{i=1}^{20} \cos(x_i/\sqrt{i}) + 1$	20	$[-600, 600]^{20}$	0
Sum squares (20)	$f_4(x) = \sum_{i=1}^{20} i x_i^2$	20	$[-100, 100]^{20}$	0
Dixon-price (30)	$f_5(x) = (x_1 - 1)^2 + \sum_{i=2}^{30} i(2x_i^2 - x_{i-1})^2$	30	$[-100, 100]^{30}$	0
Sum of different powers (30)	$f_6(x) = \sum_{i=1}^{30} \lvert x_i \rvert^{i+1}$	30	$[-100, 100]^{30}$	0

Table 2: CEC2020 benchmark functions.

Type	No.	Functions	Search range
Unimodal	$F1$	Shifted and rotated bent cigar function	$[-100, 100]^D$
Basic	$F2$	Shifted and rotated Schwefel's function	$[-100, 100]^D$
	$F3$	Shifted and rotated lunacek bi-Rastrigin function	$[-100, 100]^D$
	$F4$	Expanded Rosenbrock's plus Griewank's function	$[-100, 100]^D$
Hybrid	$F5$	Hybrid function 1 ($N = 3$)	$[-100, 100]^D$
	$F6$	Hybrid function 2 ($N = 4$)	$[-100, 100]^D$
	$F7$	Hybrid function 3 ($N = 5$)	$[-100, 100]^D$
Composition	$F8$	Composition function 1 ($N = 3$)	$[-100, 100]^D$
	$F9$	Composition function 2 ($N = 4$)	$[-100, 100]^D$
	$F10$	Composition function 3 ($N = 5$)	$[-100, 100]^D$

Table 3: Parameters settings for the involved algorithms.

Algorithms	Parameter settings	Reference
PSO	ω : 0.9~0.4, $c_1 = c_2 = 2$	[15]
MCJPSO	ω : 1~0, c_1 : 1.5~0, c_2 : 0~1.5, c_3 : 1.5~0, $k = 5$, $\alpha = 30$, $\varepsilon = 0.0001$	[23]
PSO-DLS	ω : 0.9~0.4, $c_1 = c_2 = 1.49445$, $M = 4$	[25]
EPSO	PSO: ω : 0.9~0.2, c_1 : 2.5~0.5, c_2 : 0.5~2.5 LIPS: ? = 0.729, $c = 2$, $n$size = 3 FDR-PSO: ω : 0.9~0.2, ? = 0.729, $c_1 = c_2 = 1$, $c_3 = 2$ CLPSO: ω : 0.9~0.2, c : 3–1.5 sHPSO: $\omega = 0.72$, c_1 : 2.5~0.5, c_2 : 0.5~2.5	[26]
HCLPSO	ω : 0.99~0.2, c_1 : 2.5~0.5, c_2 = 0.5~2.5, c : 3~1.5	[27]
GLPSO	$\omega = 0.7298$, $c = 1.49618$, pm = 0.01, sg = 7	[28]

when combined with SR. Whether the algorithm iterates successfully or unsuccessfully is judged according to the following content:

$$\varepsilon = \begin{cases} 0.001, & F(X^*) = 0, \\ 0.001 \times \left| F(X^*) \right|, & F(X^*) \neq 0, \end{cases} \tag{22}$$

where ε is the allowed maximum error. ε is often set as 0.001 in the engineering field. If $|F(G\text{best}) - F(X^*)| \leq \varepsilon$, the current iteration is successful; if $|F(G\text{best}) - F(X^*)| > \varepsilon$, the current iteration is unsuccessful.

For a fair comparison, all algorithms use the following unified configuration: $n = 20, t_{\max} = 2000, |v_{d\min}| = v_{d\max} = 0.1 \times (x_{d\max} - x_{d\min})$, and the initial velocity v_{id1} is set to zero. For each test function, 1000 independent runs are performed in Sections 4.2–4.4, and 30 independent runs are performed in Section 4.5. The allowed maximum number of iterations is used as the termination criterion for all algorithms. All algorithms are implemented in MATLAB R2018b and executed on the same PC with an Intel® Core(TM) i7-7700 CPU @ 3.6 GHz and 16 GB RAM.

4.2. Parameter Analysis. Based on the standard PSO with $\omega_{\cos}(t)$, 15 different configurations of a_1, a_2, and a_3 are compared on the benchmark functions f_1~f_6. In these configurations, the duration of each stage changes in the step size of $t_{\max}/8$. Therefore, a relatively good parameter configuration can be obtained. All variants are tested on the experimental settings in Section 4.1.

In Table 4, when $a_1 = 4/3, a_2 = 16/3, a_3 = 2/9$, the best performance is obtained. In Figure 2, we can see that the curve of $\omega_{\cos}(t)$ with $a_1 = (4/3), a_2 = (16/3)$, and $a_3 = (2/9)$ satisfies the requirements of the multistage inertia weight.

The inertia weights of particles that are ranked in the middle remain constant, so the value of b_2 is equal to 1. According to the requirements of the inertia weight of the particles, $b_1 < 1$, $b_3 > 1$. Particles that are already in the ROI need a small inertia weight to exploit. Particles that are far away from the ROI need a large inertia weight to explore. Therefore, b_1 should be less than 1 and b_3 should be greater than 1. A minor adjustment of inertia weight enhances the cooperation between the swarm. If the inertia weight becomes too small, the diversity of the swarm will decrease.

Table 4: Performance of PSO with different a_1, a_2, and a_3.

(a_1, a_2, a_3)	f_1	f_2	f_3	f_4	f_5	f_6
(4/3, 8/9, 4/3)	$2.456E-47$	$4.123E-47$	$3.035E-02$	$2.533E-18$	$2.706E+02$	$1.010E+29$
(4/3, 16/15, 2/3)	$9.234E-57$	$1.880E-57$	$2.884E-02$	$5.596E-23$	$3.315E+02$	$2.000E+31$
(4/3, 4/3, 4/9)	$1.125E-65$	$4.724E-67$	$2.720E-02$	$2.273E-27$	$2.604E+02$	$3.051E+25$
(4/3, 16/9, 1/3)	$3.091E-76$	$9.313E-77$	$2.743E-02$	$2.000E+01$	$2.907E+02$	$1.011E+25$
(4/3, 8/3, 4/15)	$6.126E-86$	$4.224E-86$	$2.563E-02$	$3.611E-36$	$4.825E+02$	$2.041E+27$
(4/3, 16/3, 2/9)	**$2.466E-96$**	**$2.674E-95$**	$2.397E-02$	**$7.000E-40$**	$2.111E+02$	$2.010E+27$
(2/3, 16/15, 4/3)	$8.895E-42$	$5.300E-42$	$2.878E-02$	$2.078E-16$	$3.607E+02$	$1.011E+29$
(2/3, 4/3, 2/3)	$4.436E-50$	$1.295E-51$	$2.847E-02$	$1.623E-20$	$4.508E+02$	$1.031E+25$
(2/3, 16/9, 4/9)	$3.922E-61$	$2.144E-61$	$2.672E-02$	$6.873E-25$	**$1.900E+02$**	$2.010E+27$
(2/3, 8/3, 1/3)	$3.936E-70$	$8.225E-71$	$2.411E-02$	$6.847E-29$	$2.215E+02$	$1.010E+29$
(2/3, 16/3, 4/15)	$4.251E-81$	$6.035E-80$	$2.447E-02$	$4.645E-33$	$2.604E+02$	**$4.101E+23$**
(4/9, 4/3, 4/3)	$1.855E-35$	$9.454E-37$	$2.865E-02$	$3.827E-14$	$2.522E+02$	$2.010E+29$
(4/9, 16/9, 2/3)	$1.404E-45$	$2.728E-46$	$2.577E-02$	$2.206E-18$	$4.120E+02$	$2.030E+27$
(4/9, 8/3, 4/9)	$3.625E-55$	$8.133E-56$	$2.413E-02$	$9.497E-23$	$2.509E+02$	$1.001E+29$
(4/9, 16/3, 1/3)	$3.834E-65$	$3.807E-65$	**$2.330E-02$**	$2.433E-27$	$3.512E+02$	$1.000E+31$

The best mean of fitness value is shown in bold.

This situation will lead to premature convergence. If the inertia weight becomes too large, it will lead to low convergence accuracy because particles outside the ROI insufficiently participate in local search.

The effects of b_1 and b_3 are relatively independent. Therefore, the effects of different values of b_1 or b_3 on the standard PSO with RIWs are compared separately. The comparative experiments are based on benchmark functions $f_1 - f_6$. All variants are tested on the experimental settings in Section 4.1.

In Table 5, when $b_1 = (2/3)$, the three best results are obtained. In Table 6, when $b_3 = 1.5$, the three best results are obtained. When $b_1 = 2/3$ and $b_3 = 1.5$, the adjustment of inertia weight is not large. Therefore, the recommended parameters of RIWs are selected as $b_1 = (2/3), b_2 = 1$, and $b_3 = 1.5$.

4.3. Comparing Inertia Weight $\omega_{\cos}(t)$ with Other Forms of Inertia Weight. To compare inertia weights $\omega_0(t) \sim \omega_7(t)$ and $\omega_{\cos}(t)$, they are added into the standard PSO. The configurations of $\omega_0(t) - \omega_6(t)$ and $\omega_{\cos}(t)$ are elaborated in Section 3.1. In the inertia weight $\omega_7(t)$, $\omega_{\text{ini}} = 0.9$ and $\omega_{\text{fin}} = 0.4$. For brevity, the standard PSO with inertia weight $\omega_0(t)$ is abbreviated as PSO-ω_0 and so on. All variants are tested on the experimental settings in Section 4.1.

Table 7 shows the performance of standard PSO variants with nine different forms of inertia weight. If the *SR* of a variant is 0, its average number will be omitted. We can find that PSO-$\omega_{\cos}$ has the highest *SR* in $f_1 - f_5$. In f_6 PSO-$\omega_{\cos}$ still obtains the second highest SR. $\omega_{\cos}(t)$ maintains a large value for a period of time, so its global search ability is improved. PSO-ω_{cos} has the smallest mean of the four benchmark functions and the smallest SD of the three benchmark functions. In f_5 and f_6, PSO-ω_2 has the smallest mean and SD, but its SR is still lower than PSO-$\omega_{\cos}$'s *SR*. $\omega_{\cos}(t)$ adopted a multistage strategy to meticulously guide the behaviour of particles at different stages. Therefore, PSO-$\omega_{\cos}$ obtains a better convergence quality than other variants. PSO-$\omega_{\cos}$ has the fastest convergence speed in f_5 and a moderate convergence speed in the other five benchmark functions.

Figure 6 shows the convergence characteristics of the nine variants. The convergence speed of PSO-$\omega_{\cos}$ is not very fast at the beginning, but PSO-$\omega_{\cos}$ converges to the smallest fitness value in all six benchmark functions. In f_4 and f_6, PSO-$\omega_{\cos}$ outperforms the other variants. PSO-$\omega_{\cos}$ is able to avoid becoming trapped in the local optimum $\omega_{\cos}(t)$ converts to local search quickly and smoothly. These factors help PSO-$\omega_{\cos}$ converge to an accurate solution.

4.4. Comparing Uniform Initialization with Other Particles Initializations. Random initialization, chaotic initialization, opposition-based initialization, and uniform initialization are compared on the benchmark functions $f_1 - f_6$. The four particle initializations are all added to the standard PSO-ω_1 to be compared. The configurations of random initialization, chaotic initialization and opposition-based initialization are elaborated in Section 2.3. For chaotic initialization, a is set to 4. Uniform initialization adopts the recommended configuration in Section 3.2. The four variants are abbreviated as PSO-rand, PSO-chaotic, PSO-opposition, and PSO-uniform for simplicity. All variants are tested on the experimental settings in Section 4.1.

From Table 8, it can be seen that PSO-uniform performs better than the compared variants. PSO-uniform obtains the smallest mean in five benchmark functions and the highest SR in four benchmark functions. If the initial particles gather around a local optimum, the iteration is very likely to converge prematurely. Uniform initialization reduces the possibility of this case. It distributes initial particles more uniformly and makes them more likely to approach the global optimum. This improves the SR and convergence quality of the PSO algorithm. Except for f_5, PSO-uniform has at least the second smallest average number. In addition, it has the smallest Average number in f_1. It can be concluded that PSO-uniform has a better global search ability and a better convergence speed in $f_1 - f_6$.

Table 5: Performance of PSO with different b_1.

b_1	f_1	f_2	f_3	f_4	f_5	f_6
2/3	**7.364E − 65**	**2.737E − 65**	**4.243E − 02**	9.559E − 28	1.807E + 02	1.000E + 31
½	1.360E − 64	5.581E − 65	6.441E − 02	**1.114E − 28**	1.198E + 02	1.000E + 29
2/5	1.692E − 63	1.464E − 64	7.144E − 02	1.618E − 28	1.984E + 02	1.000E + 27
1/3	3.013E − 64	1.470E − 64	5.802E − 02	3.746E − 28	2.286E + 02	**1.030E + 25**
2/7	1.061E − 63	2.657E − 63	5.469E − 02	1.588E − 27	1.696E + 02	2.020E + 25
¼	1.296E − 62	2.972E − 64	5.336E − 02	6.460E − 28	1.305E + 02	1.010E + 27
2/9	4.161E − 63	1.133E − 63	5.252E − 02	1.327E − 27	1.192E + 02	2.010E + 25
1/5	5.382E − 63	1.012E − 62	5.041E − 02	1.996E − 26	**6.069E + 01**	1.000E + 27

The best mean of fitness value is shown in bold.

Table 6: Performance of PSO with different b_3.

b_3	f_1	f_2	f_3	f_4	f_5	f_6
1.5	4.099E − 40	**3.495E − 41**	3.277E − 02	**2.894E − 15**	**4.328E + 02**	2.010E + 29
2	6.318E − 40	6.602E − 41	3.169E − 02	4.699E − 15	4.623E + 02	4.040E + 27
2.5	4.138E − 40	1.639E − 40	3.209E − 02	4.261E − 15	6.133E + 02	1.000E + 31
3	1.355E − 40	1.940E − 40	**3.078E − 02**	2.000E + 01	5.915E + 02	1.020E + 29
3.5	3.958E − 40	5.518E − 41	3.165E − 02	4.699E − 15	5.639E + 02	1.010E + 31
4	2.140E − 40	4.549E − 41	3.364E − 02	3.311E − 15	5.128E + 02	**1.041E + 27**
4.5	4.637E − 40	4.830E − 41	3.131E − 02	4.246E − 15	6.023E + 02	2.041E + 27
5	**7.087E − 41**	6.363E − 41	3.340E − 02	2.908E − 15	7.840E + 02	1.000E + 33

The best mean of fitness value is shown in bold.

Table 7: Results of the experiment on benchmark functions $f_1 \sim f_6$.

Benchmark function	Variant	Mean	SD	SR (%)	Average number
f_1	PSO-ω_0	1.433E + 02(+)	4.434E + 01	0	—
	PSO-ω_1	1.266E − 43(+)	2.313E − 42	**100**	9.702E + 02
	PSO-ω_2	3.008E − 43(+)	2.350E − 42	**100**	**9.732E + 01**
	PSO-ω_3	1.924E − 25(+)	2.446E − 24	**100**	1.436E + 03
	PSO-ω_4	3.686E − 76(+)	6.118E − 75	**100**	7.431E + 02
	PSO-ω_5	8.088E − 31(+)	4.899E − 30	**100**	2.950E + 02
	PSO-ω_6	6.636E − 95(+)	1.041E − 93	**100**	3.195E + 02
	PSO-ω_7	2.146E − 10(+)	4.798E − 10	**100**	1.443E + 03
	PSO-ω_{cos}	**3.471E − 96**	**7.484E − 95**	**100**	4.878E + 02
f_2	PSO-ω_0	3.265E + 01(+)	9.727E + 00	0	—
	PSO-ω_1	1.322E − 43(+)	3.828E − 42	**100**	9.298E + 02
	PSO-ω_2	1.420E − 43(+)	2.048E − 42	**100**	**8.715E + 01**
	PSO-ω_3	5.474E − 26(+)	6.638E − 25	**100**	1.403E + 03
	PSO-ω_4	2.540E − 76(+)	6.720E − 75	**100**	7.250E + 02
	PSO-ω_5	2.549E − 31(+)	1.781E − 30	**100**	2.555E + 02
	PSO-ω_6	2.688E − 95(+)	4.673E − 94	**100**	2.897E + 02
	PSO-ω_7	2.774E − 11(+)	4.776E − 11	**100**	1.382E + 03
	PSO-ω_{cos}	**8.009E − 97**	**9.211E − 96**	**100**	4.759E + 02
f_3	PSO-ω_0	3.524E + 00(+)	4.407E − 01	0	—
	PSO-ω_1	3.281E − 02(+)	2.894E − 02	13.4	1.372E + 03
	PSO-ω_2	2.582E − 02(+)	2.629E − 02	19.7	**2.413E + 02**
	PSO-ω_3	3.306E − 02(+)	3.054E − 02	12	1.738E + 03
	PSO-ω_4	2.494E − 02(+)	**2.377E − 02**	17.8	9.392E + 02
	PSO-ω_5	3.599E − 02(+)	4.522E − 02	12.7	9.810E + 02
	PSO-ω_6	2.825E − 02(+)	2.657E − 02	15.9	6.970E + 02
	PSO-ω_7	7.983E − 02(+)	7.891E − 02	1.7	1.950E + 03
	PSO-ω_{cos}	**2.217E − 02**	2.520E − 02	**22.8**	6.320E + 02

Table 7: Continued.

Benchmark function	Variant	Mean	SD	SR (%)	Average number
f_4	PSO-ω_0	$2.626E+03(+)$	$4.838E+02$	0	—
	PSO-ω_1	$4.126E-17(+)$	$2.101E-16$	**100**	$1.356E+03$
	PSO-ω_2	$1.092E-15(+)$	$1.001E-14$	**100**	$\mathbf{2.571E+02}$
	PSO-ω_3	$4.221E-09(+)$	$1.786E-08$	**100**	$1.749E+03$
	PSO-ω_4	$9.650E-32(+)$	$1.930E-30$	**100**	$9.382E+02$
	PSO-ω_5	$5.235E-09(+)$	$2.027E-08$	**100**	$9.729E+02$
	PSO-ω_6	$7.056E-38(+)$	$1.150E-36$	**100**	$6.657E+02$
	PSO-ω_7	$5.250E-03(+)$	$3.345E-02$	9.2	$1.976E+03$
	PSO-ω_{cos}	$\mathbf{6.008E-40}$	$\mathbf{9.875E-39}$	**100**	$6.351E+02$
f_5	PSO-ω_0	$2.787E+06(+)$	$7.504E+05$	0	—
	PSO-ω_1	$2.106E+02(-)$	$2.115E+03$	0	—
	PSO-ω_2	$\mathbf{1.456E+00}(-)$	$\mathbf{1.213E+00}$	0	—
	PSO-ω_3	$1.797E+02(-)$	$2.037E+03$	0	—
	PSO-ω_4	$3.315E+02(+)$	$2.841E+03$	0.9	$1.756E+03$
	PSO-ω_5	$1.065E+02(-)$	$1.680E+03$	0	—
	PSO-ω_6	$1.204E+02(-)$	$1.548E+03$	0.8	$1.514E+03$
	PSO-ω_7	$5.127E+02(+)$	$3.494E+03$	0	—
	PSO-ω_{cos}	$2.410E+02$	$2.449E+03$	**1.2**	$\mathbf{1.438E+03}$
f_6	PSO-ω_0	$2.031E+25(+)$	$4.468E+26$	0	—
	PSO-ω_1	$1.000E+29(+)$	$3.161E+30$	65.4	$1.810E+03$
	PSO-ω_2	$\mathbf{5.031E+13}(-)$	$\mathbf{7.054E+14}$	72.2	$\mathbf{8.460E+02}$
	PSO-ω_3	$1.000E+29(+)$	$3.161E+30$	3.9	$1.977E+03$
	PSO-ω_4	$1.080E+25(+)$	$3.162E+26$	72.1	$1.317E+03$
	PSO-ω_5	$1.000E+19(-)$	$3.161E+20$	0.4	$1.827E+03$
	PSO-ω_6	$2.000E+23(-)$	$4.468E+24$	**84.8**	$1.159E+03$
	PSO-ω_7	$1.030E+25(=)$	$3.161E+26$	0	—
	PSO-ω_{cos}	$1.031E+25$	$3.161E+26$	76.4	$1.015E+03$

The best results are shown in bold.

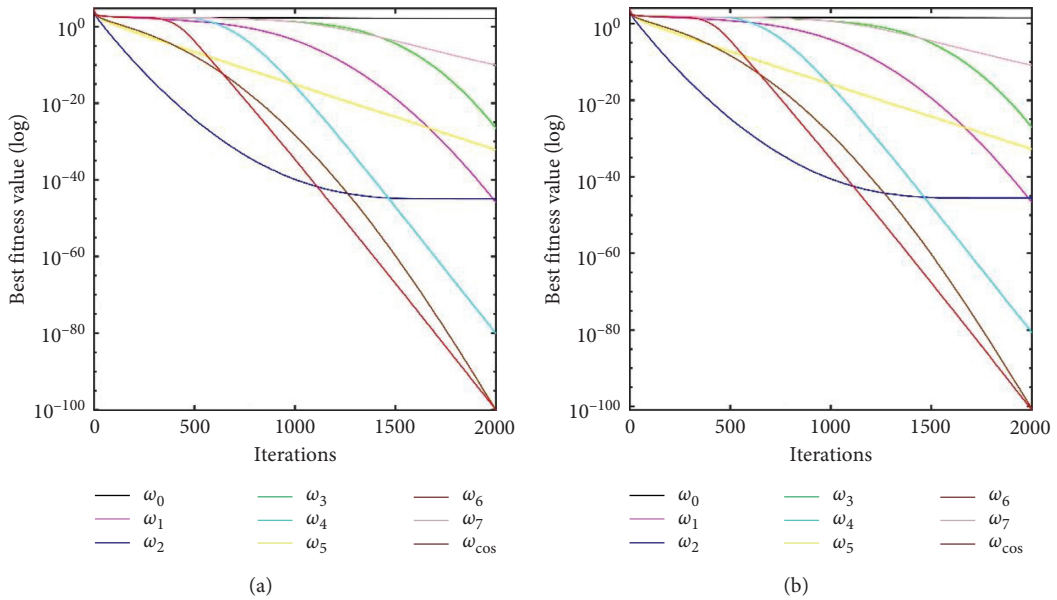

Figure 6: Continued.

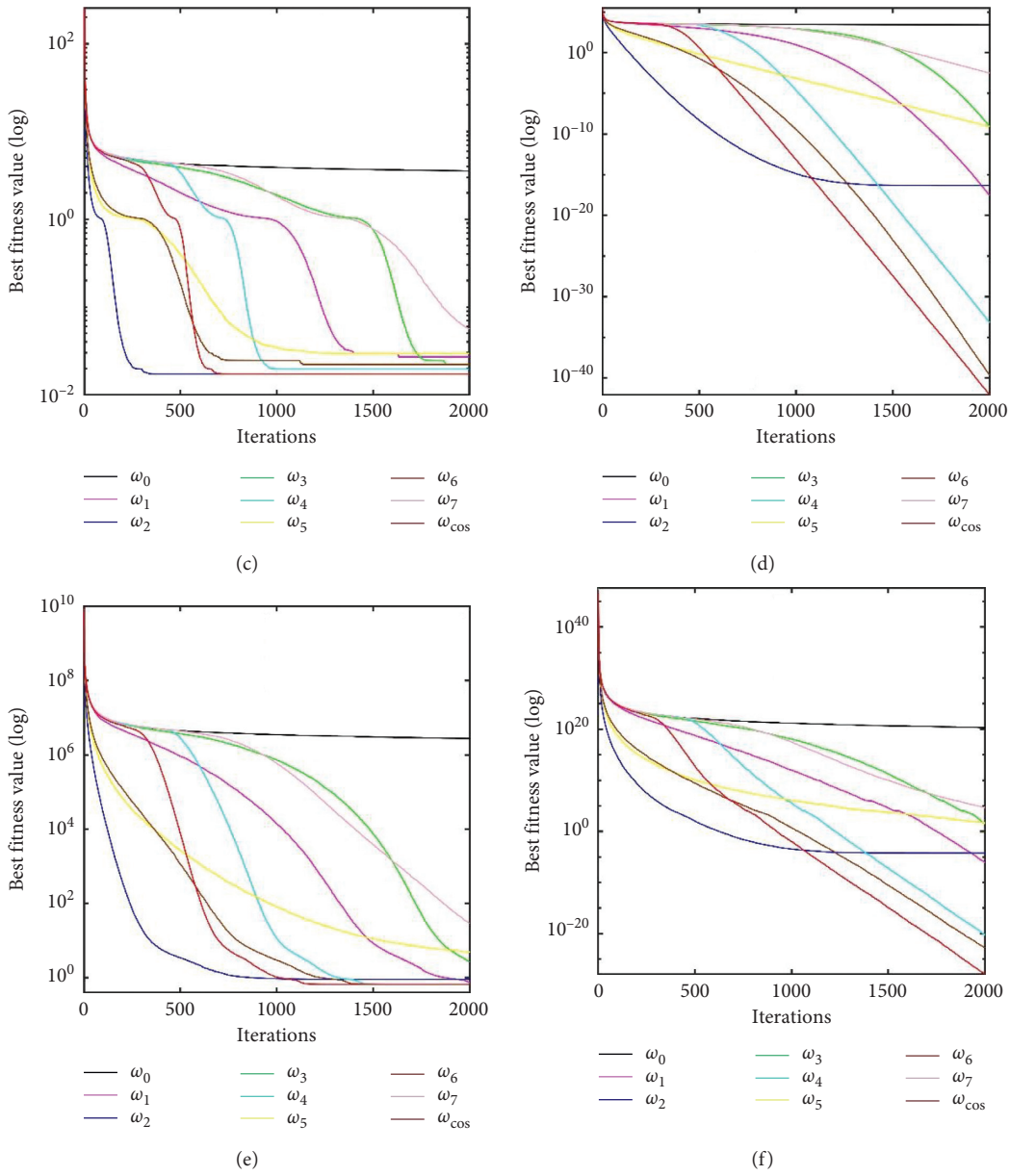

Figure 6: The median convergence curves of the standard PSO with inertia weight $\omega_0(t) \sim \omega_7(t)$ and $\omega_{cos}(t)$ on benchmark functions $f_1 \sim f_6$. (a) f_1. (b) f_2. (c) f_3. (d) f_4. (e) f_5. (f) f_6.

4.5. Comparing UCPSO with Other Variants of PSO. To analyse the performance of UCPSO, UCPSO is compared with PSO, MCJPSO [23], PSO-DLS [25], EPSO [26], HCLPSO [27], and GLPSO [28] on the CEC2020 benchmark functions. The problem dimension of the experiment is set to $D = 20$. UCPSO adopts the following configuration: $a_1 = (4/3)$, $a_2 = (16/3)$, $a_3 = (2/9)$, $b_1 = (2/3)$, $b_2 = 1$, $b_3 = 1.5$, $\omega_{\text{ini}} = 0.9$, and $\omega_{\text{fin}} = 0.4$. All algorithms are tested on the experimental settings in Section 4.1. For each test function, 30 independent runs are performed.

From Table 9, it can be observed that UCPSO outperforms PSO and PSO-DLS in 20-dimensional CEC2020 benchmark functions. Except for F_1, F_8, and F_9, UCPSO has better results than MCJPSO. This indicates that the performance of PSO is enhanced by the proposed three improvements. HCLPSO achieves an excellent result in this experiment. In most benchmark functions, UCPSO can keep up with HCLPSO. In F_4 and F_{10}, UCPSO performs almost as well as HCLPSO. In F_5, UCPSO achieves the best result. In hybrid and composition functions, UCPSO obtains good results and outperforms GLPSO. This proves that the performance of UCPSO is very competitive and that the proposed three improvements are effective.

In Figure 7, the median convergence curves of the four types of benchmark functions are shown. We can see that UCPSO has good convergence performance, especially in the top 500 iterations. UCPSO converges fast and has a high level of convergence accuracy, especially in F_4 and F_5. UCPSO can maintain a strong exploration and exploitation ability and converge to an accurate solution in a short time.

Table 8: Results of the experiment on benchmark functions $f_1 \sim f_6$.

Benchmark function	Variant	Mean	SD	SR (%)	Average number
f_1	PSO-rand	$7.069E-44$(=)	$7.309E-43$	**100**	$9.682E+02$
	PSO-chaotic	$6.130E-44$(=)	$7.711E-43$	**100**	$9.693E+02$
	PSO-opposition	$2.596E-43$(+)	$6.463E-42$	**100**	$9.687E+02$
	PSO-uniform	$\mathbf{5.674E-44}$	$\mathbf{6.344E-43}$	**100**	$\mathbf{9.679E+02}$
f_2	PSO-rand	$1.167E-44$(=)	$1.436E-43$	**100**	$\mathbf{9.288E+02}$
	PSO-chaotic	$1.573E-44$(+)	$2.384E-43$	**100**	$9.304E+02$
	PSO-opposition	$\mathbf{1.056E-44}$(=)	$1.637E-43$	**100**	$9.309E+02$
	PSO-uniform	$1.157E-44$	$\mathbf{1.163E-43}$	**100**	$9.295E+02$
f_3	PSO-rand	$3.290E-02$(=)	$2.851E-02$	12.2	$\mathbf{1.359E+03}$
	PSO-chaotic	$3.547E-02$(+)	$3.015E-02$	11.9	$1.387E+03$
	PSO-opposition	$3.231E-02$(=)	$\mathbf{2.805E-02}$	11.9	$1.378E+03$
	PSO-uniform	$\mathbf{3.217E-02}$	$2.925E-02$	**13.6**	$1.361E+03$
f_4	PSO-rand	$3.935E-17$(=)	$\mathbf{1.690E-16}$	**100**	$\mathbf{1.356E+03}$
	PSO-chaotic	$3.705E-17$(=)	$1.998E-16$	**100**	$1.358E+03$
	PSO-opposition	$4.928E-17$(+)	$3.238E-16$	**100**	$1.357E+03$
	PSO-uniform	$\mathbf{3.601E-17}$	$1.704E-16$	**100**	$1.356E+03$
f_5	PSO-rand	$1.011E+02$(=)	$1.484E+03$	0	—
	PSO-chaotic	$1.105E+02$(+)	$1.296E+03$	**0.2**	$\mathbf{1.908E+03}$
	PSO-opposition	$1.315E+02$(=)	$1.763E+03$	0	—
	PSO-uniform	$\mathbf{9.081E+01}$	$\mathbf{1.220E+03}$	0	—
f_6	PSO-rand	$1.020E+27$(+)	$3.161E+28$	64.5	$1.814E+03$
	PSO-chaotic	$1.040E+25$(+)	$3.161E+26$	68.2	$1.808E+03$
	PSO-opposition	$1.010E+21$(+)	$3.161E+22$	**83.8**	$\mathbf{1.794E+03}$
	PSO-uniform	$\mathbf{5.061E+19}$	$\mathbf{7.053E+20}$	71.4	$1.805E+03$

The best results are shown in bold.

Table 9: Results of the experiment on 20-Dimensional CEC2020 benchmark functions.

No.	Term	PSO	MCJPSO	PSO-DLS	EPSO	HCLPSO	GLPSO	UCPSO
F_1	Mean	$7.657E+09$(+)	$8.746E+04$(−)	$4.963E+08$(+)	$2.724E+03$(−)	$\mathbf{1.876E+03}$(−)	$5.278E+03$(−)	$1.623E+08$
	SD	$5.706E+09$	$1.160E+05$	$1.125E+09$	$\mathbf{1.727E+03}$	$1.945E+03$	$4.114E+03$	$3.203E+08$
F_2	Mean	$3.133E+03$(+)	$3.547E+03$(+)	$4.748E+03$(+)	$1.839E+03$(−)	$\mathbf{1.480E+03}$(−)	$1.631E+03$(−)	$2.595E+03$
	SD	$4.421E+02$	$5.693E+02$	$3.487E+02$	$3.804E+02$	$\mathbf{1.779E+02}$	$2.585E+02$	$2.013E+02$
F_3	Mean	$7.972E+02$(+)	$7.640E+02$(+)	$7.646E+02$(+)	$7.502E+02$(=)	$\mathbf{7.338E+02}$(−)	$7.373E+02$(−)	$7.505E+02$
	SD	$6.769E+01$	$1.579E+01$	$3.554E+01$	$1.768E+01$	$\mathbf{3.273E+00}$	$6.877E+00$	$8.305E+00$
F_4	Mean	$2.972E+04$(+)	$1.905E+03$(=)	$4.636E+03$(+)	$1.903E+03$(=)	$\mathbf{1.902E+03}$(=)	$1.903E+03$(=)	$1.904E+03$
	SD	$7.159E+04$	$1.195E+00$	$1.150E+04$	$1.064E+00$	$\mathbf{6.273E-01}$	$1.613E+00$	$1.336E+00$
F_5	Mean	$1.766E+06$(+)	$3.715E+05$(+)	$3.620E+05$(+)	$2.233E+05$(+)	$2.040E+05$(+)	$1.875E+05$(+)	$\mathbf{1.153E+05}$
	SD	$2.832E+06$	$2.189E+05$	$3.937E+05$	$1.786E+05$	$1.480E+05$	$2.730E+05$	$\mathbf{7.666E+04}$
F_6	Mean	$2.212E+03$(+)	$2.293E+03$(+)	$1.852E+03$(=)	$1.617E+03$(−)	$1.615E+03$(−)	$\mathbf{1.603E+03(-)}$	$1.898E+03$
	SD	$2.999E+02$	$1.341E+02$	$2.004E+02$	$1.884E+01$	$3.539E+01$	$1.471E+00$	$\mathbf{1.314E+00}$
F_7	Mean	$6.251E+05$(+)	$8.681E+04$(−)	$9.943E+04$(=)	$8.347E+04$(=)	$\mathbf{7.621E+04}$(−)	$1.724E+05$(+)	$8.730E+04$
	SD	$7.390E+05$	$6.811E+04$	$1.245E+05$	$7.031E+04$	$\mathbf{6.545E+04}$	$3.510E+05$	$1.186E+05$
F_8	Mean	$4.349E+03$(+)	$2.309E+03$(−)	$3.839E+03$(+)	$\mathbf{2.301E+03}$(−)	$\mathbf{2.301E+03}$(−)	$3.234E+03$(+)	$2.740E+03$
	SD	$9.377E+02$	$1.335E+01$	$1.881E+03$	$\mathbf{7.741E-01}$	$9.830E-01$	$1.241E+03$	$8.641E+02$
F_9	Mean	$3.019E+03$(+)	$2.858E+03$(=)	$2.896E+03$(+)	$2.768E+03$(−)	$\mathbf{2.745E+03}$(−)	$2.868E+03$(=)	$2.856E+03$
	SD	$9.117E+01$	$1.012E+02$	$2.139E+01$	$1.166E+02$	$1.198E+02$	$1.922E+01$	$\mathbf{1.865E+01}$
F_{10}	Mean	$3.181E+03$(+)	$2.962E+03$(=)	$2.936E+03$(=)	$2.948E+03$(=)	$\mathbf{2.923E+03}$(−)	$2.947E+03$(=)	$2.947E+03$
	SD	$2.558E+02$	$2.810E+01$	$2.327E+01$	$2.485E+01$	$\mathbf{1.467E+01}$	$3.382E+01$	$3.501E+01$

The best results are shown in bold.

4.6. Algorithm Complexity. This section will analyse the computational complexity of UCPSO. The computational cost of the original PSO involves the initialization (T_{ini}), evaluation (T_{eva}), velocity, and position update (T_{upd}) for each particle. D is the dimensionality of the search space, and $t_{\max}$ is the allowed maximum number of iterations. The computational complexity of PSO can be estimated as $T(D) = T_{\text{ini}} + (T_{\text{eva}} + T_{\text{upd}})t_{\max} = D + (D + 2 \cdot D)\, t_{\max} = D\ (1 + 3t_{\max})$. Therefore, the computational complexity of the original PSO is $O(D \cdot t_{\max})$ $\omega_{\cos}(t)$ can be calculated in advance, so it is used directly during the iteration process. Uniform initialization adds a process of sorting before the iteration begins. RIWs adds the process of

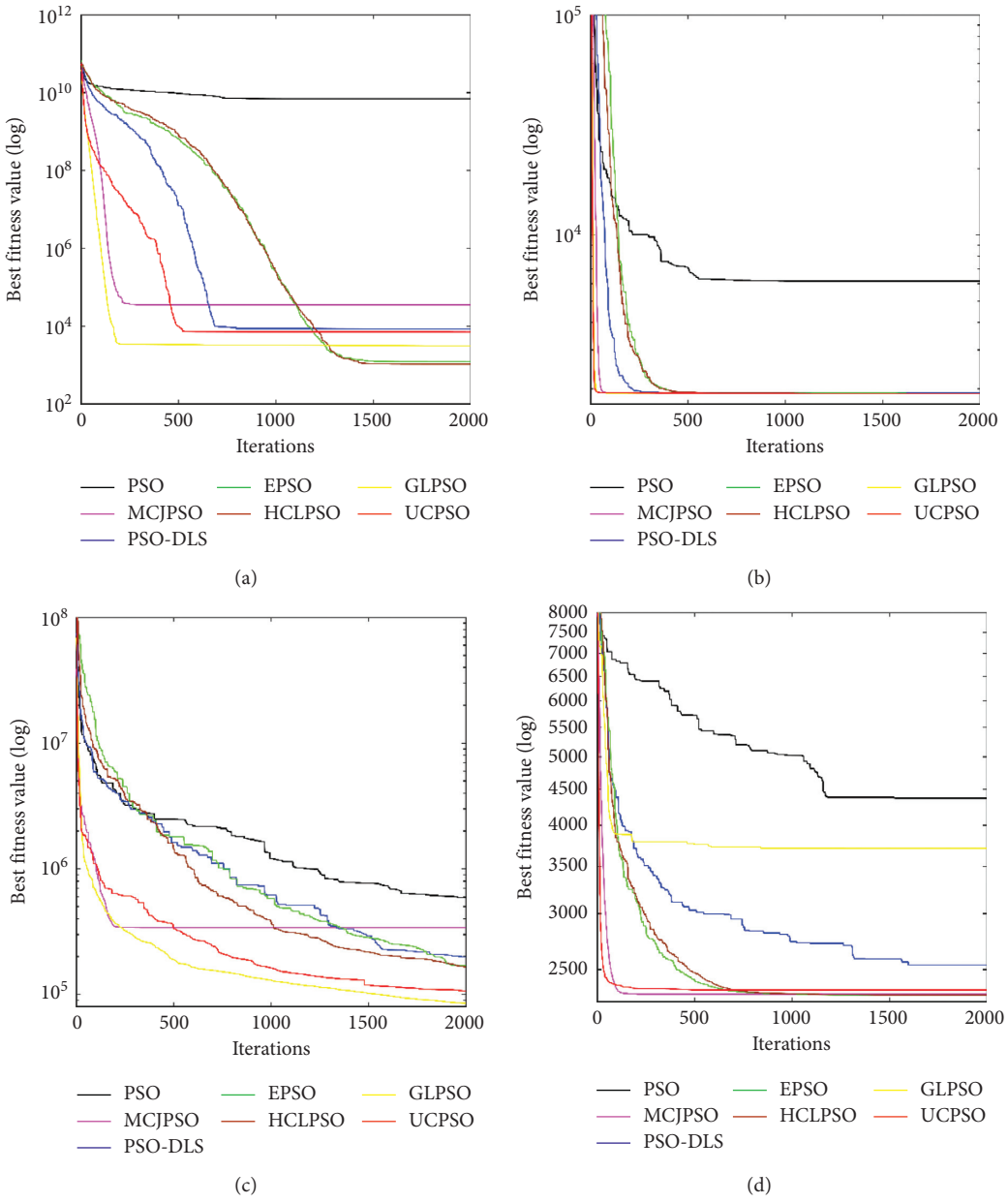

Figure 7: The median convergence curves on 20-Dimensional CEC2020 benchmark functions (a) F_1. (b) F_4. (c) F_5. (d) F_8.

sorting and assignment into each iteration. Therefore, the computational complexity of UCPSO can be estimated as follows: $T(D) = T_{\text{ini}} + (T_{\text{eva}} + T_{\text{upd}})\ t_{\max} = (D + D \cdot \log(n)) + (D + 2 \cdot D + \log(n) + D)t_{\max} = D(1 + \log(n) + 4t_{\max}) + \log(n)\ t_{\max}$. Because the number of particles n is usually small, the computational complexity of UCPSO is $O(D \cdot t_{\max})$too.

An experiment to compare the computational complexity of UCPSO with other PSO variants is carried out. T_0 is the time to run the following codes:

For $i = 1: 200000$,

$x = \text{i} + 5.5;$

$x = x + x;$

$x = \dfrac{x}{2};$

$x = x * x;$

$x = \text{sqrt}(x);$

$x = \log(x);$

$\text{x} = \exp(x);$

$y = \dfrac{x}{x};$

End. (23)

Table 10: Algorithm complexity of five algorithms on F_1 of 20-dimensional CEC2020 benchmark functions.

Algorithm	T_0 (s)	T_1 (s)	T_2 (s)	$(T_2 - T_1)/T_0$
PSO	0.014	0.023	0.196	12.357
PSO-DLS	0.014	0.023	0.546	37.357
EPSO	0.014	0.023	0.493	33.571
HCLPSO	0.014	0.023	0.363	24.286
UCPSO	0.014	0.023	0.298	19.643

Table 11: CEC2011 real-world optimization problems.

No.	Problems	D
P_1	Parameter estimation for frequency-modulated (FM) sound waves	6
P_4	Optimal control of a nonlinear stirred tank reactor	1

Table 12: Results on P_1 of CEC2011 real-world optimization problems.

Algorithm	Term	a_1	ω_1	a_2	ω_2	a_3	ω_3	$f(x)$
PSO	Best	−1.000	−5.000	−1.500	−4.800	−2.000	4.900	0
UCPSO		1.000	5.000	1.500	−4.800	−2.000	4.900	$4.563E-28$
PSO	Worst	0.424	4.948	0.698	−2.680	−6.400	6.248	$2.333E+01$
UCPSO		0.407	−0.992	−2.218	1.160	3.079	−2.667	$2.277E+01$

where T_1 is the time to execute 40,000 evaluations of benchmark function F_1 by itself with 20 dimensions and T_2 is the mean time to execute the algorithm with 40,000 evaluations of F_1 with 20 dimensions over 30 times. The number of particles is 20.

According to Table 10, UCPSO spends the least time on F_1 apart from PSO. PSO-DLS is up to 2 times slower in terms of time than UCPSO. The computational complexity of UCPSO is lower than those of PSO-DLS, EPSO, and HCLPSO. Therefore, UCPSO is a relatively fast PSO algorithm with competitive performance. The three improvements in UCPSO will not greatly increase the computational complexity.

4.7. Application to Real-World Problems. In this part, UCPSO is applied to solve real-world engineering optimization problems. PSO and UCPSO are tested on P_1 and P_4 of the CEC2011 real-world optimization problems, as shown in Table 11. For each test problem, 30 independent runs are performed. The population size is set to 20, and the maximum number of iterations is set to 2000. The allowed maximum number of iterations is used as the termination criterion for all algorithms.

4.7.1. Parameter Estimation for Frequency-Modulated (FM) Sound Waves. Frequency-modulated (FM) sound wave synthesis has an important role in several modern music systems [38]. The object of this problem is to minimize the summation of square errors between the following equations:

$$y(t) = a_1 \sin\left(\omega_1 \cdot t \cdot \theta - a_2 \cdot \sin\left(\omega_2 \cdot t \cdot \theta + a_3 \cdot \sin\left(\omega_3 \cdot t \cdot \theta\right)\right)\right), \tag{24}$$

$$y_0(t) = 1.0 \sin\left(5.0 \cdot t \cdot \theta - 1.5 \cdot \sin\left(4.8 \cdot t \cdot \theta + 2.0 \cdot \sin\left(4.9 \cdot t \cdot \theta\right)\right)\right), \tag{25}$$

where $\theta = (2\pi/100)$. The dimension of this problem is 6, and the search range is $[-6.4, 6.35]$. The fitness function is shown as

$$f(x) = \sum_{t=0}^{100} \left(y(t) - y_0(t)\right)^2. \tag{26}$$

4.7.2. Optimal Control of a Nonlinear Stirred Tank Reactor. This problem is a multimodal optimal control problem. It describes a first-order irreversible chemical reaction carried out in a continuous stirred tank reactor [38]. This chemical process is modelled by two nonlinear differential equations:

$$\dot{x}_1 = -(2+u)\left(x_1 + 0.25\right) + \left(x_2\right.$$

$$\dot{x}_2 = 0.5 - x_2 - \left(x_2 + 0.5\right)\exp\left(\frac{25x_1}{(x_1+2)}\right), \tag{27}$$

where u is the flow rate of the cooling fluid, x_1 is the dimensionless steady-state temperature, and x_2 is the deviation from the dimensionless steady-state concentration. The fitness function of this problem is

$$J = \int_0^{t_f=0.72} \left(x_1^2 + x_2^2 + 0.1u^2\right)\mathrm{d}t. \tag{28}$$

The initial condition is $x_1 = x_2 = 0.09$. The search range is unconstrained, but the initial range of u is $[0, 5]$. The dimension of this problem is 1.

As shown in Table 12, the best result of UCPSO is slightly worse than the best result of PSO, but the worst result of UCPSO is better than the worst result of PSO. In Table 13, the best and worst results of UCPSO are better than the best and worst results of PSO, respectively. According to the

Table 13: Results on P_4 of CEC2011 real-world optimization problems.

Algorithm	Term	u	$f(x)$
PSO	Best	1.213	$1.433E+01$
UCPSO		0.960	$1.377E+01$
PSO	Worst	1.792	$2.108E+01$
UCPSO		1.795	$2.096E+01$

results on P_1 and P_4, UCPSO has the ability to solve real-world engineering optimization problems.

5. Conclusion

In this paper, UCPSO is proposed to prevent PSO from falling into a local optimum and improve the comprehensive performance of PSO. It adopts a variable-period cosine inertia weight $\omega_{\cos}(t)$, uniform initialization, and rank-based strategy for individual particle inertia weights (RIWs). $\omega_{\cos}(t)$ can satisfy the requirements of inertia weight at different stages and balance exploration and exploitation better. Uniform initialization is able to avoid the aggregation of initial particles. RIWs increase the diversity of swarms and improves exploration and exploitation at the same time. The above three improvements enhance the global search ability of PSO and ensure a competitive comprehensive performance of UCPSO. Extensive tests based on benchmark functions validate the effectiveness of the improvements and the performance of UCPSO.

In future work, the authors will perform more experiments to obtain a better configuration of parameters. We intend to apply UCPSO to practical engineering fields, such as clustering, parameter optimization, image segmentation, and industry scheduling. After that, we will continue to study other forms of inertia weight and research more effective improvements that are easy to implement.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (2018YFB1308400).

References

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," 1995.

[2] A. A. Karim, N. A. Matsa, and W. H. Lim, "Modified particle swarm optimization with effective guides," *IEEE Access*, vol. 8, pp. 188699–188725, 2020.

[3] W. M. Spears and K. A. De Jong, "An overview of evolutionary computation," *The European Conference on Machine Learning*, vol. 20, pp. 442–459, 1993.

[4] S. Alam and G. Dobbie, "Research on particle swarm optimization based clustering: a systematic review of literature and techniques," *Swarm and Evolutionary Computation*, vol. 17, pp. 1–13, 2004.

[5] Q. Zhao and C. Li, "Two-stage multi-swarm particle swarm optimizer for unconstrained and constrained global optimization," *IEEE Access*, vol. 8, pp. 124905–124927, 2020.

[6] T. X. Pham, P. Siarry, and H. Oulhadj, "Integrating fuzzy entropy clustering with an improved PSO for MRI brain image segmentation," *Applied Soft Computing*, vol. 65, pp. 230–242, 2018.

[7] D. Yousri, D. Allam, M. B. Eteiba, and P. N. Suganthan, "Static and dynamic photovoltaic models' parameters identification using chaotic heterogeneous comprehensive learning particle swarm optimizer variants," *Energy Conversion and Management*, vol. 182, pp. 546–563, 2019.

[8] E. H. Dursun, H. Koyuncu, and A. A. Kulaksiz, "A novel unified maximum power extraction framework for PMSG based WECS using chaotic particle swarm optimization derivatives," *Engineering Science and Technology, an International Journal*, vol. 24, no. 1, 2021.

[9] Z. Zhou and J. Chang, "A modified PSO algorithm for task scheduling optimization in cloud computing," *Concurrency Computat Pract Exper*, vol. 30, no. 24, Article ID e4970, 2018.

[10] Z. H. Wu, Z. C. Wu, and J. Zhang, "An improved FCM algorithm with adaptive weights based on SA-PSO," *Neural Computer & Applications*, vol. 28, pp. 3113–3118, 2017.

[11] R. P. Anamika and N. Kumar, "Electricity price forecasting and classification through wavelet–dynamic weighted PSO–FFNN approach," *IEEE Systems Journal*, vol. 12, no. 4, 2018.

[12] W. H. Lim and N. A. M. Isa, "Particle swarm optimization with dual-level task allocation," *Engineering Applications of Artificial Intelligence*, vol. 38, pp. 88–110, 2015.

[13] M. S. Nobile, P. Cazzaniga, D. Besozzi et al., "Fuzzy Self-Tuning PSO: a settings-free algorithm for global optimization," *Swarm and Evolutionary Computation*, vol. 39, pp. 70–85, 2018.

[14] J. Gou, Y.-X. Lei, W.-P. Guo, C. Wang, Y.-Q. Cai, and W. Luo, "A novel improved particle swarm optimization algorithm based on individual difference evolution," *Applied Soft Computing*, vol. 57, pp. 468–481, 2017.

[15] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," 1998.

[16] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," 2000.

[17] D. Tian and Z. Shi, "MPSO: modified particle swarm optimization and its applications," *Swarm and Evolutionary Computation*, vol. 41, pp. 49–68, 2018.

[18] D. Tian, "Particle swarm optimization with chaos-based initialization for numerical optimization," *Intelligent Automation and Soft Computing*, vol. 24, no. 2, pp. 331–342, 2018.

[19] H. Zhang, J. Xie, Q. Hu, L. Shao, and T. Chen, "A hybrid DPSO with Levy flight for scheduling MIMO radar tasks," *Applied Soft Computing*, vol. 71, pp. 242–254, 2018.

[20] X. Xu, H. Rong, M. Trovati, M. Liptrott, and N. Bessis, "CS-PSO: chaotic particle swarm optimization algorithm for solving combinatorial optimization problems," *Soft Computing*, vol. 22, no. 3, pp. 783–795, 2018.

[21] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *Advances in Differential Evolution*, vol. 143, pp. 155–171, 2008.

[22] D. Tian, X. Zhao, and Z. Shi, "DMPSO: diversity-guided multi-mutation particle swarm optimizer," *IEEE Access*, vol. 7, pp. 124008–124025, 2019.
[23] A. U. Rehman, A. Islam, and S. B. Belhaouari, "Multi-cluster jumping particle swarm optimization for fast convergence," *IEEE Access*, vol. 8, pp. 189382–189394, 2020.
[24] H. T. Rauf, U. Shoaib, M. I. Lali, M. Alhaisoni, M. N. Irfan, and M. A. Khan, "Particle swarm optimization with probability sequence for global optimization," *IEEE Access*, vol. 8, pp. 110535–110549, 2020.
[25] W. Ye, W. Feng, and S. Fan, "A novel multi-swarm particle swarm optimization with dynamic learning strategy," *Applied Soft Computing*, vol. 61, pp. 832–843, 2017.
[26] N. Lynn and P. N. Suganthan, "Ensemble particle swarm optimizer," *Applied Soft Computing*, vol. 55, pp. 533–548, 2017.
[27] N. Lynn and P. N. Suganthan, "Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation," *Swarm and Evolutionary Computation*, vol. 24, pp. 11–24, 2015.
[28] Y.-J. Gong, J.-J. Li, Y. Zhou et al., "Genetic learning particle swarm optimization," *IEEE Transactions on Cybernetics*, vol. 46, no. 10, pp. 2277–2290, 2016.
[29] W. H. Lim, N. A. M. Isa, S. S. Tiang et al., "A self-adaptive topologically connected-based particle swarm optimization," *IEEE Access*, vol. 6, pp. 65347–65366, 2018.
[30] Y. Zheng, L. Ma, L. Zhang, and J. Qian, "Empirical study of particle swarm optimizer with an increasing inertia weight," *IEEE Congress on Evolutionary Computation*, vol. 2003, 2003.
[31] A. Chatterjee and P. Siarry, "Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization," *Computers & Operations Research*, vol. 33, no. 3, pp. 859–871, 2006.
[32] R. C. Eberhart and Y. H. Shi, "Tracking and optimizing dynamic systems with particle swarms," *Congress on Evolutionary Computation*, vol. 2001, 2001.
[33] Y. Feng, Y. M. Yao, and A. Wang, "Comparing with chaotic inertia weights in particle swarm optimization," 2007.
[34] Y. C. Zhang, X. Xiong, and Q. D. Zhang, "An improved self-adaptive PSO algorithm with detection function for multi-modal function optimization problems"" *Mathematical Problems in Engineering*, vol. 2013, 2013.
[35] D. Tian, X. Zhao, and Z. Shi, "Chaotic particle swarm optimization with sigmoid-based acceleration coefficients for numerical function optimization," *Swarm and Evolutionary Computation*, vol. 51, 2019.
[36] A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh, "A novel particle swarm optimization algorithm with adaptive inertia weight," *Applied Soft Computing*, vol. 11, no. 4, pp. 3658–3670, 2011.
[37] S.-K. S. Fan and Y.-Y. Chiu, "A decreasing inertia weight particle swarm optimizer," *Engineering Optimization*, vol. 39, no. 2, pp. 203–228, 2007.
[38] S. Das and P. N. Suganthan, *Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems*, Jadavpur University, Nanyang Technological University, Kolkata, India, 2010.