symmetry

MDPI

Article

A Novel Reinforcement Learning-Based Particle Swarm Optimization Algorithm for Better Symmetry between Convergence Speed and Diversity

Fan Zhang [1] and Zhongsheng Chen [2,*]

[1] The Sixty-Third Research Institute, National University of Defense Technology, Nanjing 210001, China; 120810119_zf@alu.hit.edu.cn
[2] College of Automotive Engineering, Changzhou Institute of Technology, Changzhou 213032, China
* Correspondence: czhongsheng0803@163.com

Abstract: This paper introduces a novel Particle Swarm Optimization (RLPSO) algorithm based on reinforcement learning, embodying a fundamental symmetry between global and local search processes. This symmetry aims at addressing the trade-off issue between convergence speed and diversity in traditional algorithms. Traditional Particle Swarm Optimization (PSO) algorithms often struggle to maintain good convergence speed and particle diversity when solving multi-modal function problems. To tackle this challenge, we propose a new algorithm that incorporates the principles of reinforcement learning, enabling particles to intelligently learn and adjust their behavior for better convergence speed and richer exploration of the search space. This algorithm guides particle learning behavior through online updating of a Q-table, allowing particles to selectively learn effective information from other particles and dynamically adjust their strategies during the learning process, thus finding a better balance between convergence speed and diversity. The results demonstrate the superior performance of this algorithm on 16 benchmark functions of the CEC2005 test suite compared to three other algorithms. The RLPSO algorithm can find all global optimum solutions within a certain error range on all 16 benchmark functions, exhibiting outstanding performance and better robustness. Additionally, the algorithm's performance was tested on 13 benchmark functions from CEC2017, where it outperformed six other algorithms by achieving the minimum value on 11 benchmark functions. Overall, the RLPSO algorithm shows significant improvements and advantages over traditional PSO algorithms in aspects such as local search strategy, parameter adaptive adjustment, convergence speed, and multi-modal problem handling, resulting in better performance and robustness in solving optimization problems. This study provides new insights and methods for the further development of Particle Swarm Optimization algorithms.

Keywords: PSO algorithm; convergence; diversity; reinforcement learning; Q-learning; Q-table

Citation: Zhang, F.; Chen, Z. A Novel Reinforcement Learning-Based Particle Swarm Optimization Algorithm for Better Symmetry between Convergence Speed and Diversity. *Symmetry* **2024**, *16*, 1290. https://doi.org/10.3390/sym16101290

Academic Editors: Tomohiro Inagaki and Sergei Odintsov

Received: 31 July 2024
Revised: 11 September 2024
Accepted: 25 September 2024
Published: 1 October 2024

1. Introduction

The PSO algorithm is a population-based stochastic optimization technique developed by Kennedy and Eberhart in 1995 [1]. The algorithm is inspired by the behavior of birds foraging, and finds the optimal solution by simulating the process of particles moving in the search space. The PSO algorithm has the advantages of a simple implementation, high computational efficiency, and easy convergence, and has quickly become an effective tool to solve complex optimization problems. The core idea of the PSO algorithm is to find the optimal solution by simulating the process of particles flying in the search space. In the algorithm, each particle represents a potential solution, and the position of the particle is updated by adjusting its position and velocity. The particle adjusts its position according to its own experience and the experience of the group to find the global optimal solution. This way of simulating the behavior of swarm intelligence in nature makes the

PSO algorithm perform well in dealing with high-dimensional, nonlinear and multi-modal optimization problems.

With its fast computation speed and relatively good stability, the PSO algorithm has been widely applied in various fields, such as neural network training [2,3], fault diagnosis [4,5], economics and pattern recognition, power system optimization, signal processing, data mining, image processing, finance, and medicine. It can be applied to multiple aspects of power systems, including power generation scheduling, grid planning, and power market analysis. It can help optimize the efficiency, reliability, and economy of power systems. Although the traditional PSO algorithm performs well in many problems, it also has some limitations and disadvantages. The traditional PSO algorithm tends to fall into local optima when dealing with complex optimization problems, especially for high-dimensional, non-convex, nonlinear problems, where particles may prematurely converge to local optima and fail to discover the global optimum. The convergence speed of the PSO algorithm may be slow, and especially when the search space of the optimization problem is large or the number of particles is small, it may take a long time to converge. The traditional PSO algorithm involves many parameters, such as inertia weight and acceleration coefficient, whose values have a significant impact on the performance of the algorithm. Lacking a universal selection method, experimentation and adjustment are required to determine the optimal parameter values. Additionally, the PSO algorithm is sensitive to the initial conditions and parameter settings of the problem, which makes the algorithm unstable in some cases and susceptible to noise and interference. As regards premature convergence and premature stagnation, in some cases, the PSO algorithm may miss better solutions due to premature convergence, or experience premature stagnation, leading to algorithmic stagnation.

Currently, various improved and optimized PSO algorithms have been proposed by researchers to address the shortcomings of traditional PSO algorithms, such as with parameter adjustments [6–9], multi-strategy cooperative PSOs [10–17], and hybrid evolutionary algorithms [18–24]. Based on existing research, improvements in PSO algorithms can be mainly classified into the following three categories:

Class one: Adaptive parameter adjustment. This category of algorithms mainly adjusts the design parameters of the PSO algorithm, such as the inertia weight of velocity and the weights between individual best (*pbest*) and global best (*gbest*), to change the convergence speed of the algorithm. For example, the PSO with inertia weight (PSO-w) algorithm is used to increase the convergence speed [6], while Ratnaweera et al. proposed various inertia weight adjustment strategies [7]. Additionally, some algorithms such as the Q-learning-based Particle Swarm Optimization (QLPSO) [8] and Adaptive Weighted PSO (AWPSO) [9] algorithms adjust the convergence speed by controlling algorithm parameters.

Class two: Comprehensive learning from other particles. The main idea of such algorithms is to fully utilize information from other particles to update the flight speed and position of the current particle, including the current positions and *pbest*s of other particles. Every particle in the swarm can comprehensively learn from other particles. In the Fully Informed Particle Swarm (FIPS) algorithm, each particle utilizes information from all neighboring particles, not just from *gbest* [10]. In the PSO-w-local algorithm, each particle compares the performances of every other member in the social network structure and imitates the best-performing particle [11]. The Cooperative Particle Swarm Optimization (CPSO) algorithm divides the particle swarm into multiple subgroups to optimize different components of the solution vector cooperatively. In each iteration, the CPSO algorithm uses each dimension of particles to update *gbest*, avoiding the issue of "two steps forward, one step back" [12]. The Comprehensive Learning Particle Swarm Optimization (CLPSO) algorithm utilizes *pbest*s of all other particles to update the velocity of particles. This learning strategy can prevent premature convergence [13]. The Example-based Learning Particle Swarm Optimization (ELPSO) algorithm proposes a strategy to update particle positions using an example set of multiple global best particles [14]. The Heterogeneous Comprehensive Learning Particle Swarm Optimization (HCLPSO)

algorithm divides the swarm into two subgroups, one focusing on exploration and the other on exploitation [15]. The Terminal Crossover and Direction-based Particle Swarm Optimization (TCSPSO) utilizes *pbest* to enhance population diversity at the terminal stage of iteration, helping particles jump out of local optima [16]. G.P. Xu et al. proposed the Two-Swarm Learning PSO (TSLPSO) algorithm, which is a dimensional learning strategy (DLS) for discovering and integrating promising information of the population best solution [17].

Class three: Hybrid particle swarm optimization. This category of algorithms integrates different optimization ideas to improve traditional PSO algorithms. For example, the PSO-GA algorithm incorporates mutation operators from genetic algorithms into the PSO algorithm [18,19]. Uriarte A et al. integrated the gradient descent method (BP algorithm) as an operator into the PSO algorithm [20]. Additionally, hybrid particle swarm optimization algorithms combine the PSO algorithm with other optimization techniques such as simulated annealing [21]. Aydilek et al. proposed a hybrid (HFPSO) algorithm that combines advantages of the firefly algorithm and PSO algorithm [22]. Moreover, the PSO-CL algorithm adopts a crossover learning strategy, utilizing a comprehensive learning strategy (CCL) and stochastic example learning strategy (SEL), to balance global exploration and local exploitation capabilities [23]. Zhang et al. constructed the TLS-PSO algorithm, utilizing a worst–best example learning strategy, to achieve a hybrid learning mechanism with three learning strategies in PSO [24].

Despite the improved performance of PSO algorithms in respective problems, they still have limitations when solving complex problems. For example, a high convergence speed can quickly approach individual best points (*pbest*) and global best points (*gbest*), but it may lead to the loss of diversity in the particle swarm, especially when *gbest* and *pbest* are far from the global optimum but close to each other [14]. The good diversity of the particle swarm ensures the algorithm's global search capability but may result in slow convergence. According to the "no free lunch" theorem [25], many improved PSO algorithms may still fall into local optima or converge too slowly when solving complex problems. Especially when balancing convergence speed and diversity, existing algorithms often fail to achieve ideal results.

The RLPSO algorithm proposed in this paper is inspired by the comprehensive learning strategy of CLPSO and integrates reinforcement learning policies. Through intelligent learning strategies, it achieves a better balance between convergence speed and diversity, endowing particles with more intelligent learning strategies, effective global search capabilities, dynamic adjustment of learning strategies, and a good balance between convergence speed and diversity, as well as wide applicability. This enables more effective information sharing and utilization during the learning process, providing an efficient and intelligent optimization method for problem-solving. RLPSO has a wide range of applications and can play an important role in engineering [4], science [11], finance [19], medicine [26], and other fields by improving the performance and robustness of optimization algorithms, thus providing effective solutions to practical problems.

The remaining parts of this paper are as follows. In Section 2, the theoretical foundation of the RLPSO algorithm is briefly introduced. In Section 3, the execution process of the RLPSO algorithm is explained in detail. In Section 4, we discuss parameter selection and the role of Q-learning, while also selecting 29 benchmark functions to validate the RLPSO algorithm. Finally, conclusions with a discussion and summary are given in Section 5.

2. The Basic Principle of the CLPSO Algorithm

In the PSO algorithm, the velocity V_i^d update and position X_i^d update of the dth dimension of the particle i are represented by Equation (1) and Equation (2), respectively [19].

$$V_i^d = V_i^d + c_1 \times r_1^d \times (pbest_i^d - X_i^d) + c_2 \times r_2^d \times (gbest^d - X_i^d) \quad (1)$$

$$X_i^d = X_i^d + V_i^d \Delta t \quad (2)$$

$X_i = (X_i^1, X_i^2, \ldots, X_i^D)$ and $V_i = (V_i^1, V_i^2, \ldots, V_i^D)$ represent the position and velocity of the particle i, D is the number of dimensions, $pbest_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^D)$ is the best-so-far position of the particle i, $gbest = (gbest^1, gbest^2, \ldots, gbest^D)$ is the best-so-far position of the whole swarm, c_1 and c_2 are constant weights for *pbest* and *gbest*, respectively, r_1^d and r_2^d are two random numbers in the range [0,1]; $\Delta t = 1$. If $\left|V_i^d\right| > V_{max}^d$, then $V_i^d = V_{max}^d \text{sign}(V_i^d)$, where V_{max}^d is maximum allowable velocity of the dth dimension.

The CLPSO algorithm has a solid theoretical foundation, wide application, and effectively utilizes swarm intelligence to address multimodal problems, demonstrating a certain degree of reliability and effectiveness, especially in tackling multimodal optimization problems. The CLPSO algorithm [13] can maintain a good diversity within the swarm; it excels particularly in tackling multimodal problems. Therefore, as the theoretical foundation of the RLPSO algorithm, CLPSO possesses a certain degree of reliability and effectiveness in addressing multimodal optimization problems. At the core of the CLPSO algorithm lies the updating rule, which serves as its main concept. The specific formulas are given as Equations (3) and (4).

$$V_i^d = w_k V_i^d + c \times r \times (pbest_{f_i(d)}^d - X_i^d) \tag{3}$$

$$X_i^d = X_i^d + V_i^d \Delta t \tag{4}$$

ω_k is inertia weight when the number of iterations is k; $f_i = [f_i(1), f_i(2), \ldots, f_i(\text{D})]$ defines the particles' *pbest*s that the particle i should follow. $pbest_{f_i(d)}^d$ can be the corresponding dimension of any particle's *pbest* as the selection of $f_i(d)$ depends on the probability Pc_i. The probability of selecting the i particle's *pbest* is $(1 - Pc_i)$, while the probability of selecting other particles' *pbest* is Pc_i. The value of Pc_i is computed as in Equation (5).

$$Pc_i = 0.05 + 0.45 \times \left[\exp\left(\frac{10(i-1)}{ps-1}\right) - 1\right] / [\exp(10) - 1] \tag{5}$$

where ps is the population size of the swarm, and i is the particle's id counter.

In this paper, the objective of the PSO algorithm and its various variants is to locate the global minimum [3,4]. Figure 1 illustrates the flowchart of the CLPSO algorithm.

Figure 1 illustrates how the CLPSO algorithm updates *pbest* and *gbest* by learning from other particles, although the particles being learned are randomly selected. This may lead to certain particles being unable to learn from superior ones and deriving no valuable insights from inferior ones. Consequently, learning may occasionally prove ineffective, causing the CLPSO algorithm to converge slowly and underutilize swarm information. For instance, considering the equation $f(X) = \sum_{i=1}^{d} \{X^i\}^2$, we assume the following conditions:

$$V_1 = (0,0,0), w = 1, c = 2, r = 0.5, X_1 = (6,2,0),$$
$$pbest_1 = (4,1,4), pbest_2 = (1,2,5), pbest_3 = (1,2,1)$$

The first, second, and third dimensions of this particle learn from $pbest_1$, $pbest_2$, and $pbest_3$, respectively. X_1 is updated according to Equations (3) and (4); we can obtain a new $X_{1\_new} = (4,2,1)$ and $f(X_{1\_new}) = 21$, which is better than $f(X_1) = 40$ and $f(pbest_1) = 33$. So, $pbest_1 = (4,2,1)$ is updated to $pbest_1 = X_{1\_new} = (4,2,1)$. For this case, although the new fitness value is better, the second dimension of X_{1_new} is not updated (still at 2), and the third dimension is updated to be farther away from the optimal point (0, 0, 0), shifting from 0 to 1. Each dimension of a particle has the potential to learn from a different particle, but the particles learned may not necessarily be optimal within the context of the CLPSO algorithm. Therefore, we adjust the particle learning strategy based on the CLPSO algorithm, optimizing the learning objects and enhancing learning efficiency.

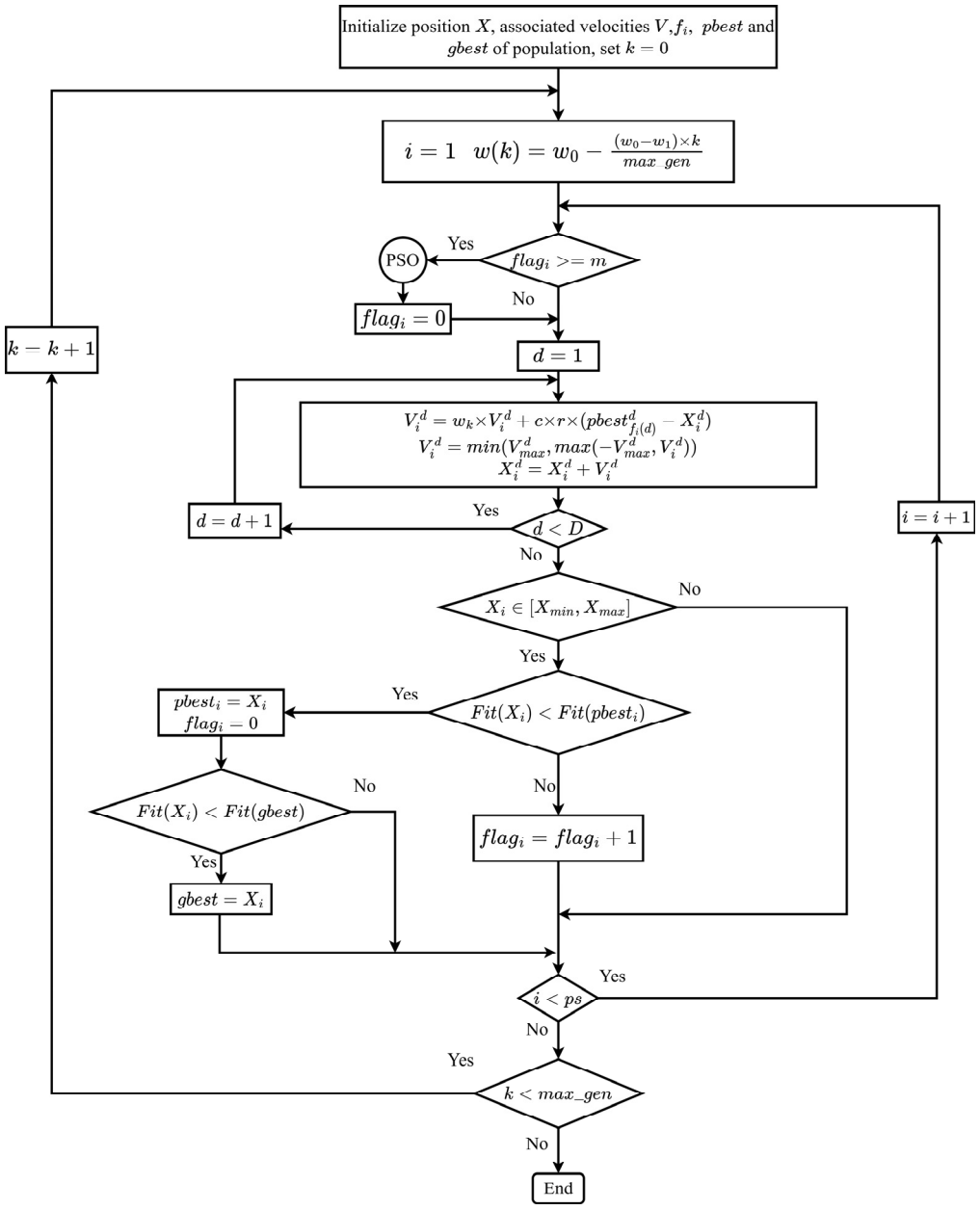


Figure 1. A flowchart of the CLPSO algorithm. Note: the meanings of the parameters in the figure are shown in Table 1.

Table 1. The meanings of the parameters for the CLPSO algorithm.

Parameters	The Meaning of the Parameters
ω_0	inertia weight of the first iteration
ω_1	inertia weight of the last iteration
PSO	the PSO algorithm
ps	population size
max_gen	maximum generations
k	generation counter
i	particle's id counter
d	dimension
$\omega(k)$	inertia weight
$gbest^d$	the *d*th dimension's value of *gbest*
$flag_i$	the number of generations for which the particle *i* has not improved its *pbest*, and the initial value for $flag_i$ of the particle *i* is 0.

3. Reinforcement Learning-Based PSO Algorithm

To address the limitations of the aforementioned PSO algorithms, we introduce a novel PSO algorithm incorporating reinforcement-learning principles. In the RLPSO algorithm, particles consistently learn from superior peers while preserving diversity. Rather than randomly selecting particles from the swarm, each particle chooses its learned peers based on the Q table. Moreover, to maximize the effectiveness of each learning instance, the RLPSO algorithm updates every dimension of the global best (*gbest*) using all recently updated personal bests (*pbests*). Further insights into the RLPSO algorithm are provided below.

3.1. Q-Learning in RLPSO

Reinforcement learning (RL) is a branch of machine learning that dictates how machines should behave in their current environment to maximize cumulative rewards. Within machine learning, there are three fundamental components: state, action, and reward [26]. Q-learning, first introduced by Watkins in 1989 [27], is a specific type of RL algorithm. Q-learning involves the creation and updating of a Q table, which guides the machine's actions based on the current state. In some scenarios, the machine selects the action with the highest Q value from the Q table, updating the Q table during training. The updated policy is illustrated in Equation (6).

$$Q(s,a) = Q(s,a) + \alpha \times [R(s,a) + \gamma \times max_{a\prime} Q(s\prime, a\prime) - Q(s,a)] \quad (6)$$

α is the learning rate, γ is the discount factor, $R(s, a)$ is the immediate reward acquired from executing action a under the state s, $Q(s, a)$ is an accumulative reward, and s' is the next state of the state s when executing action a under the state s; a' is an optional action under the state s' and $max_{a'}(s', a')$ is the maximum Q value that can be obtained at the state s'. The model of Q-learning is shown in Figure 2.

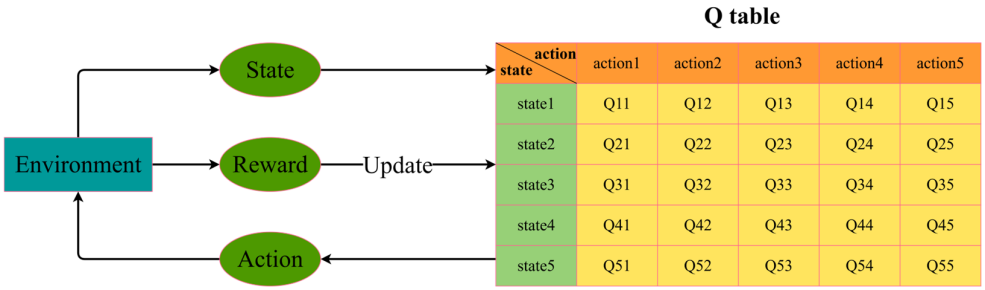


Figure 2. The model of Q-learning.

In the RLPSO algorithm, we randomly generate a ($D \times ps$)-size Q table for each particle of the swarm. D is the number of particle dimensions; ps is the population size of the swarm. The Q table dictates from which particle's *pbest* each dimension of every particle learns during each iteration. When updating each dimension of the particle, the particle with the highest Q value becomes the focal point of learning. In instances where the particle with the highest Q value is itself, the particle can still learn from its own past experiences. Each particle maintains its own Q table, as illustrated in Figure 3.

	particle 1	particle 2	particle 3	...	particle k	...	particle ps
1st dimension	$Q_{1,1}$	$Q_{1,2}$	$Q_{1,3}$	...	$Q_{1,k}$	...	$Q_{1,ps}$
2nd dimension	$Q_{2,1}$	$Q_{2,2}$	$Q_{2,3}$	...	$Q_{2,k}$	...	$Q_{2,ps}$
3rd dimension	$Q_{3,1}$	$Q_{3,2}$	$Q_{3,3}$	...	$Q_{3,k}$	...	$Q_{3,ps}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
dth dimension	$Q_{d,1}$	$Q_{d,2}$	$Q_{d,3}$	...	$Q_{d,k}$	...	$Q_{d,ps}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Dth dimension	$Q_{D,1}$	$Q_{D,2}$	$Q_{D,3}$	...	$Q_{D,k}$	...	$Q_{D,ps}$

Figure 3. The Q table of the particle i.

As shown in Figure 3, each dimension of the particle i has only one state, which is how each dimension learns from other particles' *pbests*. Every dimension of the particle i has ps actions, that is, selects which particle's *pbest* to learn from ps particles. Thus, Equation (6) can be simplified to Equation (7):

$$Q_{d,f_i(d)} = Q_{d,f_i(d)} + \alpha \times [R + \gamma \times maxQ_d - Q_{d,f_i(d)}] \quad (7)$$

d represents the dth dimension of the particle i, $f_i(d)$ represents which particle's *pbest* the dth dimension of the particle i learns from, $Q_{d,fi(d)}$ is the Q value that the dth dimension of the particle i can obtain when learning from the *pbest* of particle $f_i(d)$, and $maxQ_d$ is the largest Q value of the dth dimension in the Q table of the particle i.

To effectively harness the collective knowledge within the particle swarm, we employ Q-learning for selecting learned particles, rather than resorting to random selection from the swarm. Within the RLPSO algorithm, particles initially choose particles to learn from randomly with a certain probability, thereby exploring the solution space extensively and updating the Q table at the onset of iterations. As the iterations progress, particles gradually adjust their selection of particles to learn from based on the Q table, thereby expediting convergence and preventing divergence in certain dimensions.

Depending on the outcomes of the updates, different rewards are assigned during the Q table update process. The update strategy is outlined as follows: when *gbest* undergoes an update, it receives the highest "global reward" as an immediate reward for updating the Q table. Conversely, when *pbest* is updated, it receives a larger "local reward" as an immediate incentive. In cases where no update occurs, a "penalty" is assigned. The Q table is updated according to Equation (7).

3.2. The Strategy of Selecting Learned Particles

During each particle's updating process, the RLPSO algorithm randomly generates a number dim_{up} from 1 to D as the number of dimensions which need updating, and randomly selects dim_{up} dimensions from the particle to be updated, leaving the unselected dimensions unchanged. In the learning process of each dimension within the particle, the algorithm randomly selects a particle from the swarm of learning with a certain probability ε_k and selects a particle to learn from according to the Q table with probability $(1 - \varepsilon_k)$. Then, V_i^d and X_i^d are updated according to Equation (3) and Equation (4), respectively. After each iteration, ε_k updates according to Equation (8); *des* is the magnitude of the descent per iteration, and k is the number of iterations.

$$\varepsilon_{k+1} = \varepsilon_k \times (1 - des) \quad (8)$$

Equation (8) reveals that as iterations progress, the likelihood of selecting learned particles based on the Q table gradually escalates. Consequently, particles begin to incrementally glean insights from superior counterparts, as dictated by the Q table.

In contrast to the CLPSO algorithm, which relies on the fitness values of two randomly selected particles to determine learned particles, the RLPSO algorithm expedites this process by leveraging the Q table. The selection process of $f_i(d)$ for the particle i is shown in Figure 4. Simultaneously, we update the Q table of particle i in response to rewards or penalties incurred during the updating of *gbest* and *pbest*. If the dimension that requires updating is denoted by d, we update $Q_{d,fi(d)}$ in accordance with Equation (7). Otherwise, no updates are made. We repeat this process from d = 1 until d = D, resulting in the complete update of particle i's Q table.

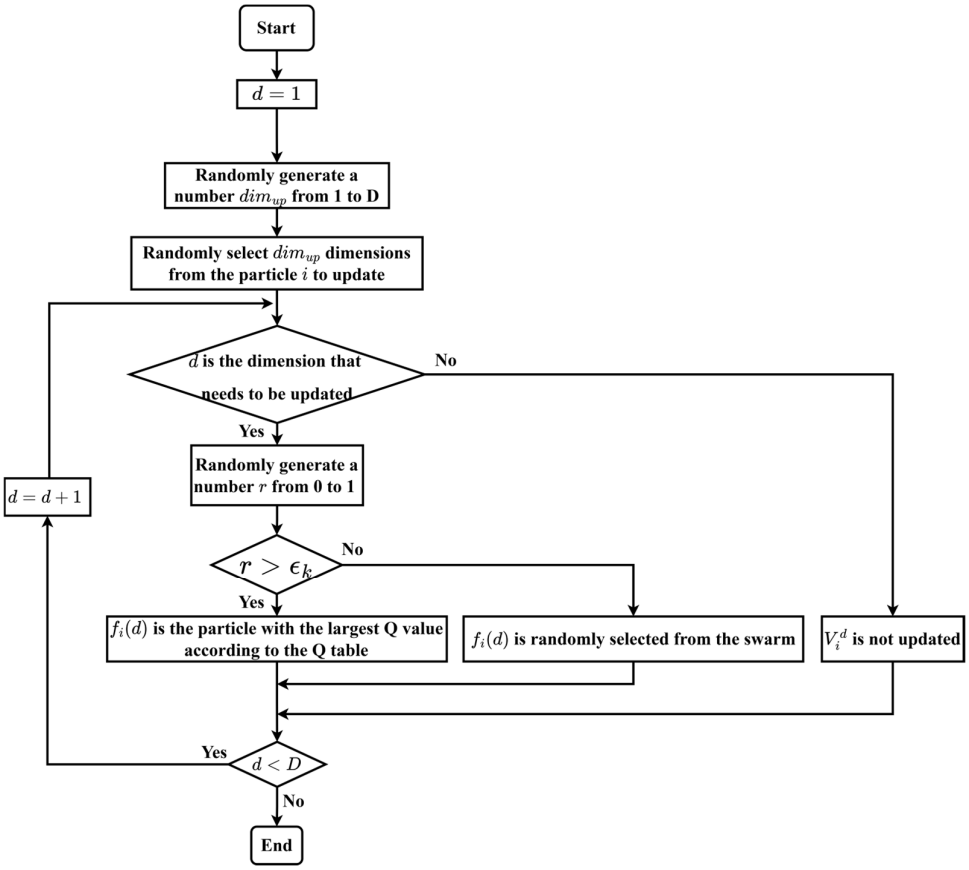


Figure 4. The flowchart of selecting learned particles for the particle i.

3.3. The Full Flow of the RLPSO Algorithm

In this section, we present the specific process of RLPSO, outlined as follows:

Step 1: We initialize the parameters of the RLPSO algorithm, including the position X, associated velocities V, Q table, ε_0, *pbest*, and *gbest* of population; set k = 0.

Step 2: Each dimension of the particle i selects the learned particle $f_i(d)$ according to Figure 4 to obtain a new velocity V_i and position X_i.

Step 3: For the particle i, if the *pbest* is updated, we update each dimension of *gbest* based on the 1 to D dimensions of the *pbest*. This ensures that certain dimensions of *gbest* do not stray too far from the optimal point. We can view each dimension of *pbest* as a gene, with *gbest* selectively inheriting useful genes from *pbest*. If *gbest* is not updated, we set $flag_i = flag_i + 1$. If $flag_i \geq$ m, run the PSO algorithm and reset $flag_i = 0$. Different from the CLPSO algorithm where $flag_i$ is reset to 0 when *pbest* is updated, the RLPSO algorithm only resets $flag_i$ to 0 when *gbest* is updated. This encourages particles to choose learned particles based on Q-learning more frequently.

Step 4: If *gbest* is updated, the particle *i* receives a global reward; if *gbest* is not updated but *pbest* is, the particle *i* receives a local reward; if neither *gbest* nor *pbest* is updated, particle *i* incurs a penalty. Subsequently, we update the Q table of particle *i* based on the rewards or penalties obtained from updating *gbest* and *pbest*.

Step 5: We set $i = i + 1$; repeat Step 2, Step 3, and Step 4 until *i* equals the population size *ps*.

Step 6: We set $k = k + 1$; repeat Step 2, Step 3, Step 4, and Step 5 until *d* equals the maximum number of iterations.

A detailed description of algorithm pseudocode is shown in Algorithm 1, and the entire flowchart of the RLPSO algorithm is illustrated in Figure 5. The parameters shown in Algorithm 1 and Figure 5 are consistent with those set in Figure 1.

Algorithm 1: RLPSO algorithm

Input: Initialize position *X*, associated velocities *V*, Q table, *ε0*, *pbest* and *gbest* of population, and set k = 0, flag = 0, m = 10;
Output: Optimal solution;

```
for k < max_gen do
    for I < ps do
        if flag_i ≥ m then
            run PSO algorithm and reset flag_i = 0;
        End
        for d < D do
            Each dimension of the particle i selects the learned particle f_i(d) according
            to Figure 4 to get new velocity V_i and position X_i;
        End
        if Fit (X_i) < Fit (pbest)
            Update the pbest of particle i;
        End
        if the pbest of particle i is updated then
            we will update each dimension of gbest basis with 1 to D dimensions of
            the pbest from particle i;
        else
            we set flag_i = flag_i + 1
        End
        if gbest is updated, then
            R = global reward;
        else if gbest is not updated but pbest is updated, then
            R = local reward;
        else if both gbest nor pbest are not updated, then
            R = get penalty;
        End
        Update the Q table of particle i according to reward or penalty getting from
        updating of gbest and pbest;
        i = i + 1;
    End
    k = k + 1
End
Output: gbest
```

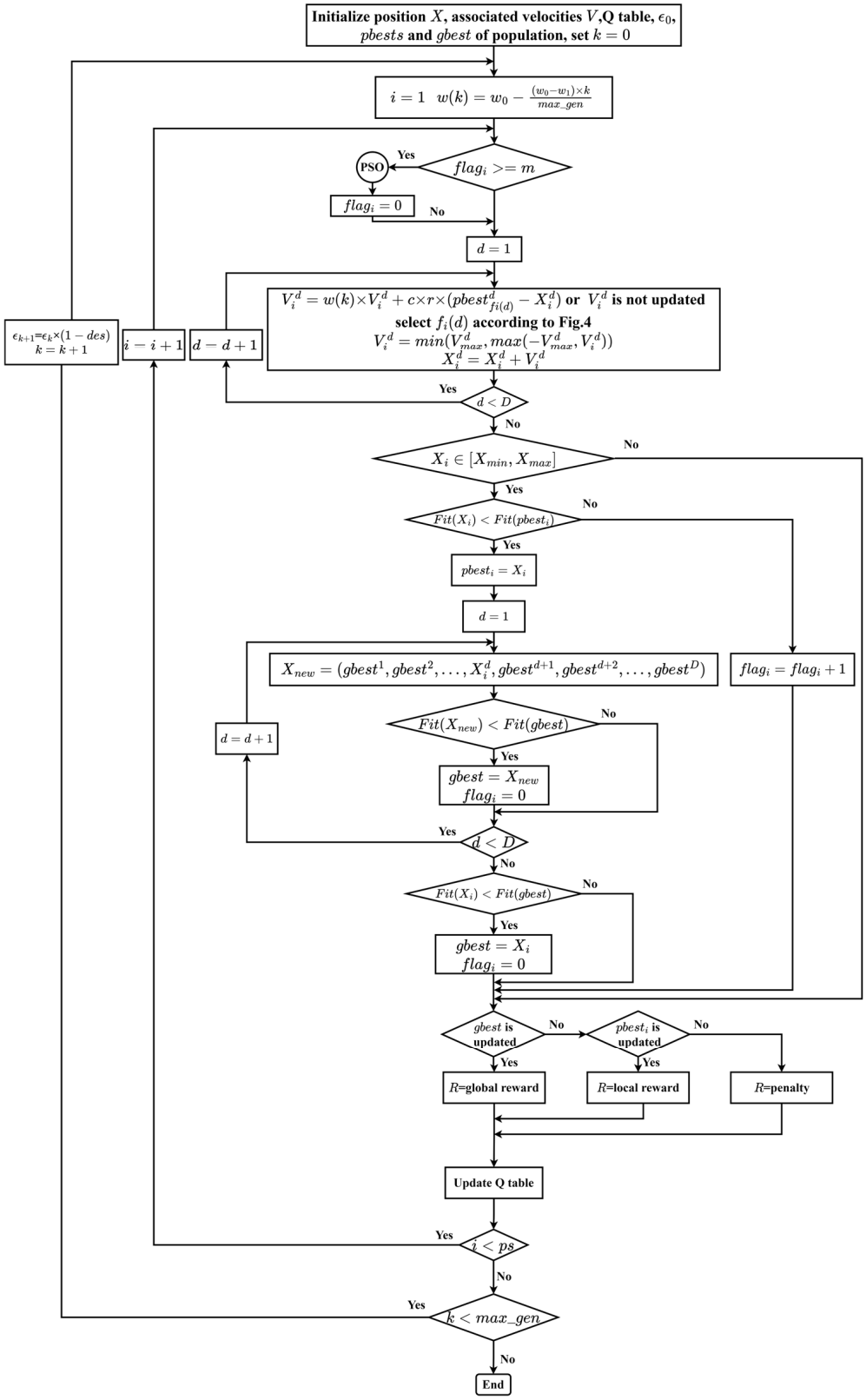


Figure 5. A Flowchart of the RLPSO algorithm, cite from Figure 4.

The pseudocode of the RLPSO algorithm reveals a structure consisting of two nested loops. Each iteration within the outer loop corresponds to a particle within the swarm, and within each particle's iteration, every dimension is iterated over. Thus, the time complexity of the algorithm is influenced by the particle dimension D, the population size ps, and the number of algorithm iterations max_gen. Consequently, the algorithm's time complexity can be inferred as O($max\_gen \times ps \times D$).

4. Experimental Validations

For testing the performance of the RLPSO algorithm, a total of 16 famous benchmark functions were selected from CEC2005 [28], and the performance of the RLPSO algorithm was compared with other PSO algorithms. The 16 benchmark functions include seven unimodal functions and nine multimodal functions to ensure the comprehensiveness of the experiment. The presented algorithm is implemented in Python 3.9 and the program has been run on a i7-8565U @1.80 GHz Intel(R) Core(TM) 4 Duo processor with 8 GB of Random Access Memory (RAM). The tested benchmark functions are listed as follows:

Unimodal functions:

(1) f_1: sphere model:

$$f_1(x) = \sum_{i=1}^{30} x_i^2,\ -100 \le x_i \le 100,\ \min(f_1) = f_1(0,\ldots,0) = 0$$

(2) f_2: Schwefel's problem 2.22:

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|,\ -10 \le x_i \le 10,\ \min(f_2) = f_2(0,\ \ldots,0) = 0$$

(3) f_3: Schwefel's problem 1.2:

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2,\ -100 \le x_i \le 100,\ \min(f_3) = f_3(0,\ \ldots,0) = 0$$

(4) f_4: Schwefel's problem 2.21:

$$f_4(x) = \max\{|x_i|, 1 \le i \le 30\},\ -100 \le x_i \le 100,\ \min(f_4) = f_4(0,\ \ldots,0) = 0$$

(5) f_5: generalized Rosenbrock's function:

$$f_5(x) = \sum_{i=1}^{29} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2],\ -30 \le x_i \le 30,\ \min(f_5) = f_5(1,\ \ldots,1) = 0$$

(6) f_6: step function:

$$f_6(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2,\ -100 \le x_i \le 100,\ \min(f_6) = f_6(0,\ \ldots,0) = 0$$

(7) f_7: quartic function, i.e., noise:

$$f_7(x) = \sum_{i=1}^{30} i x_i^4 + \text{random}[0,1),\ -1.28 \le x_i \le 1.28,\ \min(f_7) = f_7(0,\ \ldots,0) = 0$$

Multimodal functions:

(8) f_8: generalized Schwefel's problem 2.26:

$$f_8(x) = \sum_{i=1}^{30} \left(x_i \sin(\sqrt{|x_i|}) \right),\ -500 \le x_i \le 500,\ \min(f_8) = f_8(420.9687,\ldots,420.9687) = -12569.5$$

(9) f_9: generalized Rastrigin's function:

$$f_9(x) = \sum_{i=1}^{30} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right],\ -5.12 \le x_i \le 5.12,\ \min(f_9) = f_9(0, \ldots, 0) = 0$$

(10) f_{10}: Ackley's function:

$$f_{10}(x) = -20exp(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}) - \exp(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i) + 20 + e,\ -32 \le x_i \le 32,$$
$$\min(f_{10}) = f_{10}(0, \ldots, 0) = 0$$

(11) f_{11}: generalized Griewank function:

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30}\cos(\frac{x_i}{\sqrt{i}}) + 1,\ -600 \le x_i \le 600,\ \min(f_{11}) = f_{11}(0, \ldots, 0) = 0$$

(12) f_{12}: generalized penalized function:

$$f_{12}(x) = \frac{\pi}{30}\left\{ 10\sin^2(\pi y_1) + \sum_{i=1}^{29} \{(y_i - 1)^2 \times [1 + 10\sin^2(\pi y_{i+1})]\} + (y_n - 1)^2 \right\}$$
$$+ \sum_{i=1}^{30} u_i(x_i, 10, 100, 4),$$
$$y_i = 1 + \frac{1}{4}(x_i + 1), u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$$
$$-50 \le x_i \le 50,\ \min(f_{12}) = f_{12}(-1, \ldots, -1) = 0$$

(13) f_{13}: generalized penalized function:

$$f_{13}(x) = 0.1\left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{29} \{(x_i - 1)^2 \times [1 + 10\sin^2(3\pi x_{i+1})]\} + (x_n - 1)^2 \times [1 + \sin^2(2\pi x_{30})] \right\}$$
$$+ \sum_{i=1}^{30} u_i(x_i, 5, 100, 4),\ -50 \le x_i \le 50,\ \min(f_{13}) = f_{13}(1, \ldots, 1) = 0$$

The function u is the same as above.

(14) f_{14}: six-hump camel-back function:

$$f_{14}(x) = 4x_1^2 - 2.1x_1^4 + \tfrac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4,\ -5 \le x_i \le 5$$
$$x_{\min} = (0.08983, -0.7126), (-0.08983, 0.7126), \min(f_{14}) = -1.0316285$$

(15) f_{15}: Branin function:

$$f_{15}(x) = (x_2 - \tfrac{5.1}{4\pi^2}x_1^2 + \tfrac{5}{\pi}x_1 - 6)^2 + 10(1 - \tfrac{1}{8\pi})\cos x_1 + 10,\ -5 \le x_1 \le 10,\ 0 \le x_2 \le 15$$
$$x_{\min} = (-3.142, 12.275), (3.142, 2.275), (9.425, 2.425), \min(f_{15}) = 0.398$$

(16) f_{16}: Goldstein–Price function

$$f_{16}(x) = [1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$\times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$
$$-2 \le x_i \le 2,\ \min(f_{16}) = f_{16}(0, -1) = 3$$

To evaluate the performance of the RLPSO algorithm, comparisons with other algorithms were conducted under the same test parameters. The problem dimension, population size, maximum number of iterations, and number of independent runs were set uniformly as 30, 40, 5000, and 30 [14]. For the RLPSO algorithm, we randomly generated a Q table of ($D \times ps$) size for each particle of the population, and each value in the Q table was randomly generated as an integer between −40 and 0. Then, we compared the performance

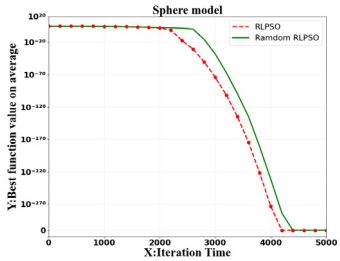
of the PSO, CLPSO, ELPSO, and RLPSO algorithms using experimental data from an H.D. Shao article [14]. Table 2 presents the parameter settings for all comparison algorithms, as obtained from their respective studies.

Table 2. Parameter settings for PSO algorithms.

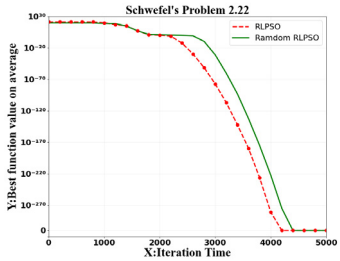
Algorithm	Parameter Settings
PSO [1]	$\omega = 0.729$, $c_1 = c_2 = 1.494$
CLPSO [13]	$\omega = 0.9$–0.4, $c = 1.494$, $m = 7$
ELPSO [14]	$\omega = 0.729$, $c_1 = 1.49445$, $c_2 = 1.494$, $m = 7$, $B_m = 4$
RLPSO	$\omega = 0.9$–0.4, $c = 1.49445$, $m = 10$, $dim_{up} = 30$, global reward = 10, local reward = 2, penalty = -1, $\alpha = 0.1$, $\gamma = 0.95$, $\varepsilon_0 = 0.6$, $des = 0.001$

4.1. An Analysis of the Role of Q-Learning

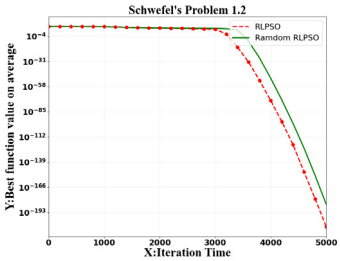
To explore the role that Q-learning plays in the RLPSO algorithm, we set some parameters, such as $\varepsilon_0 = 1$ and $des = 0$, while keeping other parameters constant, to evaluate performance. Under these conditions, the RLPSO algorithm randomly selects learned particles without considering the Q table. We denote this configuration of the RLPSO algorithm as "Random RLPSO". The performance of both "Random RLPSO" and "RLPSO" on 16 benchmark function problems is depicted in Figure 6, while corresponding experimental results are presented in Table 3.



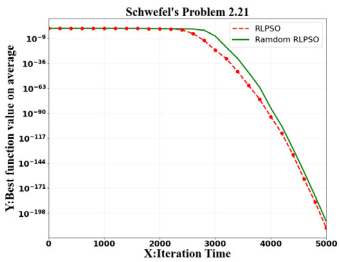
(**a**) f_1: sphere model



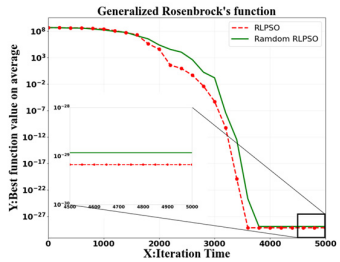
(**b**) f_2: Schwefel's problem 2



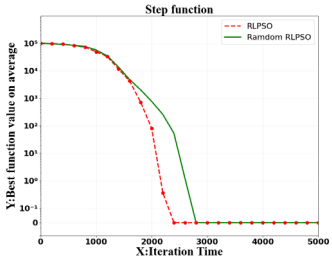
(**c**) f_3: Schwefel's problem



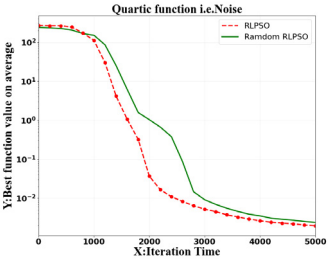
(**d**) f_4: Schwefel's problem 2



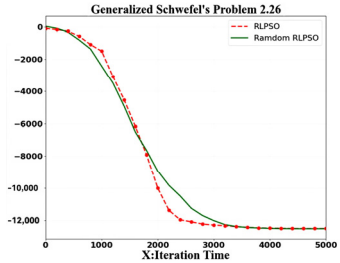
(**e**) f_5: generalized Rosenbrock's function



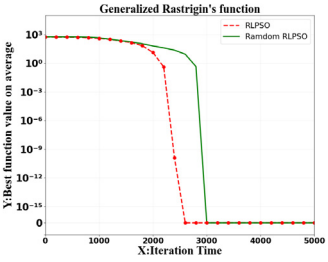
(**f**) f_6: step function



(**g**) f_7: quartic function i.e.,noise

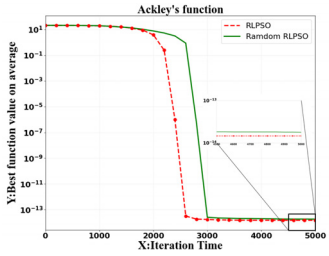


(**h**) f_8: generalized Schwefel's problem 2.26

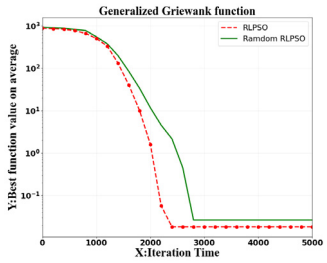


(**i**) f_9: generalized Rastrigin's function

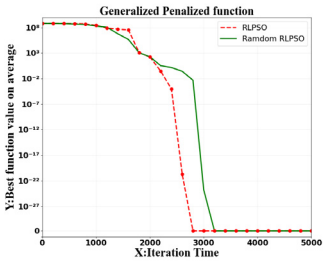
Figure 6. *Cont.*



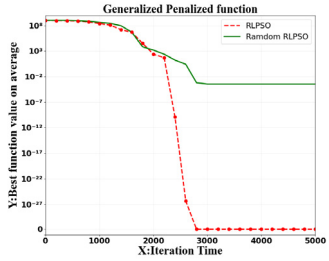
(**j**) f_{10}: Ackley's function



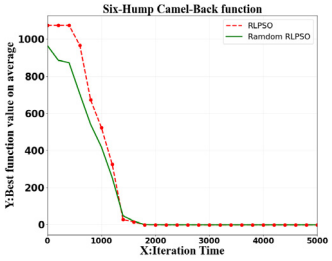
(**k**) f_{11}: generalized griewank function



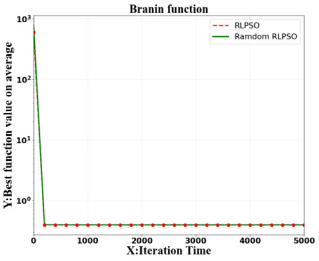
(**l**) f_{12}: generalized penalized function



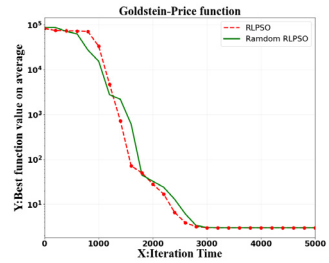
(**m**) f_{13}: generalized penalized function



(**n**) f_{14}: six-hump camel-back function



(**o**) f_{15}: Branin function



(**p**) f_{16}: Goldstein–Price function

Figure 6. A comparison of Random RLPSO and RLPSO on the convergence of f_1–f_{16}.

Figure 6 illustrates that "RLPSO" achieves earlier convergence compared to "Random RLPSO" across almost all 16 benchmark functions. Additionally, according to the experimental results in Table 3, "RLPSO" outperforms "Random RLPSO" in terms of convergence for eight benchmark functions (f_3–f_5, f_7, f_{10}–f_{13}). As for the remaining eight benchmark functions (f_1–f_2, f_6, f_8–f_9, f_{14}–f_{16}), both "RLPSO" and "Random RLPSO" converge to optimal values, but "RLPSO" does so sooner.

The above analysis demonstrates that Q-learning can enhance the RLPSO algorithm's fitness value by accelerating convergence at appropriate intervals, thereby achieving a better balance between convergence speed and diversity. Strategic learning proves to be more efficient than random learning, as evidenced by experimental results. Initially, random learning with a certain probability of ε_k allows for comprehensive exploration and utilization of population diversity. As ε_k decreases, particles gradually learn from superior particles based on the Q table, further expediting convergence. The adjustment of parameters ε_0 and *des* enables control over the timing of convergence acceleration. With a maximum of 5000 iterations, we aim for enhanced convergence speed after 2000 iterations (about 40% of the total) by setting parameters to $\varepsilon_0 = 0.6$ and *des* = 0.001. Consequently, ε_k becomes very small ($\varepsilon_k < 0.078$) after 2000 iterations.

4.2. Parameter Setting of RLPSO

The configurations of global reward, local reward, and penalty are crucial in determining the performance of the RLPSO algorithm. This section outlines an experimental approach for configuring these parameters.

We evaluated the performance of five sets of parameter configurations across benchmark functions. Table 4 presents these parameter sets and compares their performance

across 16 benchmark functions. Table 4 also includes the mean and standard deviation of the optimal solutions, with the best results among the five algorithms highlighted in bold.

Table 3. Comparisons of values extracted by the PSO, CLPSO, ELPSO, and RLPSO algorithms in 30 trials.

Function		PSO	CLPSO	ELPSO	RLPSO	Random RLPSO	*t*-Test *s* Value
f_1	Mean	6.75×10^{-96}	4.46×10^{-14}	4.8×10^{-93}	**0**	**0**	1
	Std	3.5×10^{-100}	1.73×10^{-14}	1.26×10^{-93}	**0**	**0**	
f_2	Mean	5.53×10^{-16}	3.79×10^{-12}	1.41×10^{-12}	**0**	**0**	1
	Std	1.72×10^{-20}	2.19×10^{-12}	2.51×10^{-12}	**0**	**0**	
f_3	Mean	6.98×10^{-9}	4.68×10^{-3}	8.45×10^{-12}	$\mathbf{3.77 \times 10^{-209}}$	4.71×10^{-185}	1
	Std	1.25×10^{-10}	3.83×10^{-3}	6.58×10^{-12}	$\mathbf{2.02 \times 10^{-208}}$	1.93×10^{-184}	
f_4	Mean	1.52×10^{-6}	2.6	2.71×10^{-7}	$\mathbf{1.91 \times 10^{-214}}$	2.2×10^{-207}	1
	Std	1.84×10^{-7}	2.4	2.81×10^{-7}	$\mathbf{7.2 \times 10^{-214}}$	7.15×10^{-207}	
f_5	Mean	1.01×10^{1}	2.1×10^{1}	9.82	$\mathbf{6.67 \times 10^{-30}}$	1.17×10^{-29}	1
	Std	1.66	2.98	1.66	$\mathbf{1.49 \times 10^{-29}}$	1.37×10^{-29}	
f_6	Mean	**0**	**0**	**0**	**0**	**0**	0
	Std	**0**	**0**	**0**	**0**	**0**	
f_7	Mean	7.49×10^{-3}	5.78×10^{-3}	3.9×10^{-3}	$\mathbf{1.97 \times 10^{-3}}$	2.4×10^{-3}	1
	Std	9.4×10^{-4}	2.34×10^{-3}	1.42×10^{-3}	$\mathbf{8.1 \times 10^{-4}}$	1.09×10^{-3}	
f_8	Mean	-8.44×10^{3}	-9.54×10^{3}	-1.22×10^{4}	$\mathbf{-1.25 \times 10^{4}}$	$\mathbf{-1.25 \times 10^{4}}$	1
	Std	5.68×10^{2}	2.15×10^{2}	3.29×10^{2}	$\mathbf{9.01 \times 10^{1}}$	$\mathbf{8.5 \times 10^{1}}$	
f_9	Mean	4.69×10^{1}	4.85×10^{-10}	**0**	**0**	**0**	0
	Std	1.59×10^{1}	3.63×10^{-10}	**0**	**0**	**0**	
f_{10}	Mean	1.21	**0**	**0**	1.47×10^{-14}	1.8×10^{-14}	1
	Std	8.6×10^{-1}	**0**	**0**	4.1×10^{-15}	5.2×10^{-15}	
f_{11}	Mean	2.88×10^{-2}	3.14×10^{-10}	**0**	1.82×10^{-2}	2.62×10^{-2}	1
	Std	3.18×10^{-2}	4.64×10^{-10}	**0**	2.36×10^{-2}	3.01×10^{-2}	
f_{12}	Mean	3.52	1.12×10^{-11}	3.02×10^{-17}	**0**	**0**	1
	Std	5.2×10^{-1}	1.12×10^{-10}	1.35×10^{-18}	**0**	**0**	
f_{13}	Mean	8.46×10^{1}	1.07×10^{-11}	2.88×10^{-17}	**0**	3.66×10^{-4}	1
	Std	2.35×10^{-1}	1.7×10^{-27}	2.06×10^{-22}	**0**	1.97×10^{-3}	
f_{14}	Mean	**−1.0316285**	**−1.0316285**	**−1.0316285**	**−1.0316285**	**−1.0316285**	0
	Std	**0**	**0**	**0**	**0**	$\mathbf{4.44 \times 10^{-16}}$	
f_{15}	Mean	0.398903	0.398903	0.398874	**0.397887**	**0.397887**	1
	Std	0	0	0	**0**	**0**	
f_{16}	Mean	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**	0
	Std	**0**	**0**	**0**	**0**	$\mathbf{1.22 \times 10^{-15}}$	

Note: a *t*-test s value of 1 indicates statistically significant differences in performances between the RLPSO algorithm and the PSO, CLPSO, and ELPSO algorithms at a 95% confidence level, while a *t*-test s value of 0 suggests no statistically significant differences. The best results among the five algorithms highlighted in bold.

The results indicate that the RLPSO algorithm performs best when Global Reward = 10, Local Reward = 2 and Penalty = −1. In this scenario, the RLPSO algorithm outperforms others in 13 benchmark functions, expect for f_3, f_{10}, and f_{11}. When Global Reward = 10 and Penalty = −1, if *gbest* is updated, the particle receives Global Reward and follows the same learning strategy at least 10 times. Similarly, if *pbest* is updated, the particle receives Local Reward and follows the same learning strategy at least two times. If *gbest* and *pbest* remain unchanged for an extended period, the particle adjusts its learning strategy and begins to learn from other particles' *pbests*. The RLPSO algorithm balances convergence speed and particle diversity through reward and penalty. Therefore, setting Global Reward = 10, Local Reward = 2, and Penalty = −1 achieves a desirable balance between convergence speed and particle diversity. Additionally, we set $\alpha = 0.1$ and $\gamma = 0.95$ following the conventional Q-learning parameter settings of many cases.

Table 4. Comparisons of 5 groups of parameter settings.

Function		Global Reward = 10 Local Reward = 5 Penalty = −1	Global Reward = 10 Local Reward = 5 Penalty = −2	Global Reward = 10 Local Reward = 2 Penalty = −1	Global Reward = 10 Local Reward = 2 Penalty = −2	Global Reward = 10 Local Reward = 1 Penalty = −2
f_1	Mean	**0**	**0**	**0**	**0**	**0**
	Std	**0**	**0**	**0**	**0**	**0**
f_2	Mean	**0**	**0**	**0**	**0**	**0**
	Std	**0**	**0**	**0**	**0**	**0**
f_3	Mean	8.62×10^{-197}	$\mathbf{4.3 \times 10^{-218}}$	3.77×10^{-209}	2.6×10^{-201}	1.15×10^{-194}
	Std	4.64×10^{-196}	$\mathbf{2.31 \times 10^{-217}}$	2.02×10^{-208}	1.4×10^{-200}	5.52×10^{-194}
f_4	Mean	1.3×10^{-204}	2.99×10^{-209}	$\mathbf{1.91 \times 10^{-214}}$	3.39×10^{-210}	5.13×10^{-206}
	Std	6.99×10^{-204}	8.97×10^{-209}	$\mathbf{7.2 \times 10^{-214}}$	1.82×10^{-209}	2.53×10^{-205}
f_5	Mean	1.53×10^{-29}	2.25×10^{-29}	$\mathbf{6.67 \times 10^{-30}}$	1.05×10^{-29}	1.01×10^{-29}
	Std	2.28×10^{-29}	7.23×10^{-29}	$\mathbf{1.49 \times 10^{-29}}$	2.09×10^{-29}	1.57×10^{-29}
f_6	Mean	**0**	**0**	**0**	**0**	**0**
	Std	**0**	**0**	**0**	**0**	**0**
f_7	Mean	2.47×10^{-3}	2.21×10^{-3}	$\mathbf{1.97 \times 10^{-3}}$	2.43×10^{-3}	2.51×10^{-3}
	Std	9.26×10^{-4}	1.43×10^{-3}	$\mathbf{8.1 \times 10^{-4}}$	1.06×10^{-3}	9.22×10^{-4}
f_8	Mean	$\mathbf{-1.25 \times 10^{4}}$	$\mathbf{-1.25 \times 10^{4}}$	$\mathbf{-1.25 \times 10^{4}}$	$\mathbf{-1.25 \times 10^{4}}$	$\mathbf{-1.25 \times 10^{4}}$
	Std	$\mathbf{6.23 \times 10^{1}}$	$\mathbf{7.95 \times 10^{1}}$	$\mathbf{9.01 \times 10^{1}}$	$\mathbf{6.23 \times 10^{1}}$	$\mathbf{6.23 \times 10^{1}}$
f_9	Mean	**0**	**0**	**0**	3.32×10^{-2}	**0**
	Std	**0**	**0**	**0**	1.79×10^{-1}	**0**
f_{10}	Mean	1.44×10^{-14}	1.43×10^{-14}	1.47×10^{-14}	1.41×10^{-14}	$\mathbf{1.38 \times 10^{-14}}$
	Std	3.99×10^{-15}	2.95×10^{-15}	4.1×10^{-15}	2.76×10^{-15}	$\mathbf{3.86 \times 10^{-15}}$
f_{11}	Mean	2.12×10^{-2}	1.81×10^{-2}	1.82×10^{-2}	$\mathbf{1.52 \times 10^{-2}}$	2.29×10^{-2}
	Std	2.4×10^{-2}	2.08×10^{-2}	2.36×10^{-2}	$\mathbf{1.91 \times 10^{-2}}$	2.56×10^{-2}
f_{12}	Mean	**0**	**0**	**0**	**0**	**0**
	Std	**0**	**0**	**0**	**0**	**0**
f_{13}	Mean	**0**	**0**	**0**	**0**	**0**
	Std	**0**	**0**	**0**	**0**	**0**
f_{14}	Mean	**−1.0316285**	**−1.0316285**	**−1.0316285**	**−1.0316285**	**−1.0316285**
	Std	**0**	$\mathbf{9.57 \times 10^{-16}}$	**0**	**0**	$\mathbf{9.62 \times 10^{-15}}$
f_{15}	Mean	**0.397887**	**0.397887**	**0.397887**	**0.397887**	**0.397887**
	Std	**0**	$\mathbf{3.19 \times 10^{-16}}$	**0**	**0**	$\mathbf{4.66 \times 10^{-14}}$
f_{16}	Mean	**3.00**	**3.00**	**3.00**	**3.00**	**3.00**
	Std	**0**	**0**	**0**	**0**	**0**

Note: the best results among the five algorithms highlighted in bold.

4.3. Experimental Results and Analysis

In this section, we compare the RLPSO algorithm and the Random RLPSO algorithm with classical PSO, CLPSO, and ELPSO algorithms. Table 3 presents a comparison of the above five algorithms in 16 benchmark functions. The table displays the mean and standard deviation of the optimal solutions for the PSO, CLPSO, ELPSO, Random RLPSO, and RLPSO algorithms, with the best results among the five algorithms highlighted in bold.

The results of the *t*-test are presented in the last column of Table 3. At a 95% confidence level, when $s = 1$, this indicates that the performance differences between the RLPSO algorithm and the PSO, CLPSO, and ELPSO algorithms are statistically significant. Conversely, when $s = 0$, it suggests no statistically significant differences. Among the comparisons for the 16 benchmark functions in Table 3, 12 exhibited statistically significant disparities.

As shown in Table 3, the RLPSO algorithm demonstrates superior performance in solving unimodal function problems (f_1–f_5, f_7) compared to all other algorithms. When tackling multimodal function problems (f_8–f_{16}), RLPSO outperforms in seven cases (f_8, f_9, f_{12}– f_{16}). Overall, the RLPSO algorithm emerges as the best solution in 14 out of 16 function problems.

Despite the RLPSO algorithm's average best solution value being lower than that of the CLPSO and ELPSO algorithms in the multimodal function problem f_{11}, RLPSO successfully identifies 12 global optimal solutions of f_{11} across 30 runs. To provide a comprehensive assessment of the RLPSO algorithm, Figure 7 illustrates the frequency with which RLPSO finds the global optimum in 30 runs across 16 benchmark functions.

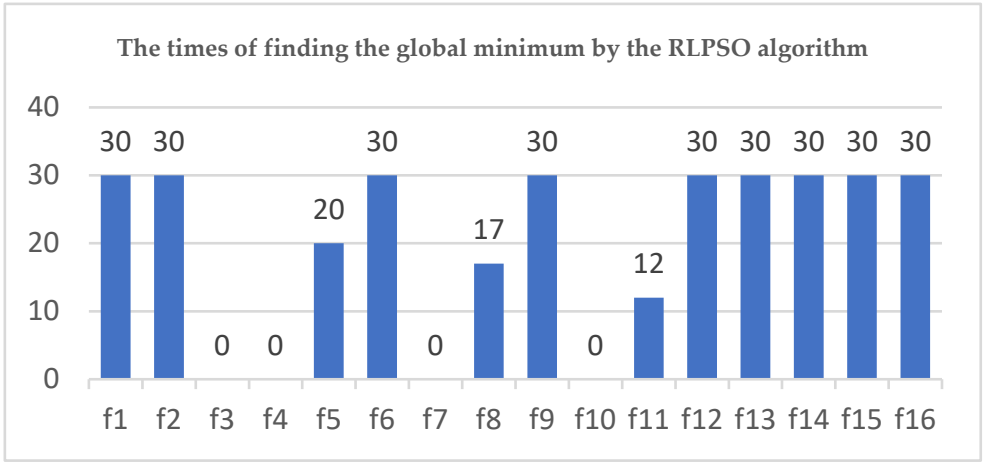


Figure 7. The times of finding the global minimum by the RLPSO algorithm.

Figure 7 shows that when the RLPSO algorithm is applied to each benchmark function problem 30 times, it successfully identifies 12 global minima for 16 benchmark function problems. In the case of the remaining 4 benchmark function problems (f_3–f_4, f_7, f_{10}) where the global minimum is not found, the *gbest*s discovered by the RLPSO algorithm are very close to the global optimum. For instance, the average value of the optimal solution for f_3 is 3.77×10^{-209}, which is close to 0, the global optimal solution of f_3. Similarly, the average value of the optimal solution for f_4 is 1.91×10^{-214}, also close to 0, the global optimal solution of f_4. The same applies to f_7 and f_{10}. Based on the above analysis, we can conclude that the RLPSO algorithm has the capability to identify the global optimum when run multiple times within a certain error range.

Furthermore, it is evident that various PSO algorithms exhibit different performances across different benchmark functions. At times, they may successfully locate the global optimum, while in other instances, they may become trapped in local optima or converge too slowly. The experimental results confirm that the RLPSO algorithm manages to obtain nearly all global optima within a certain margin of error. Consequently, the RLPSO algorithm demonstrates greater stability compared to other PSO algorithms when addressing diverse problem sets.

To further validate the performance of the RLPSO algorithm, we selected 13 test functions from the CEC2017 benchmark set as the benchmark functions and compared the RLPSO algorithm with four PSO algorithms and two evolutionary algorithms on the test set. To evaluate the RLPSO algorithm's performance, we compared it with other algorithms using identical test parameters. All algorithms were configured with the same settings: a population size of 50, a maximum iteration count of 1000, a particle dimension of 30, and each algorithm was executed 50 times [24]. The selected test functions are listed in Table 5, algorithm parameters are provided in Table 6, and the computational results are presented in Table 7.

The data from Table 7 clearly indicate that the RLPSO algorithm outperforms the other six algorithms (CLPSO, HPSO, THSPSO, TCSPSO, BOA, and OSA algorithms) in the testing of the 13 benchmark functions of CEC2017, achieving the minimum value on 11 of these benchmarks. This demonstrates the superior performance of the RLPSO algorithm, suggesting its capability to find solutions closer to the global optimum in most scenarios. These results imply that the RLPSO algorithm could be an effective and reliable choice for addressing complex optimization problems.

4.4. Particle Swarm Diversity Analysis

We conducted Principal Component Analysis (PCA) on functions f_{17} to f_{29} over 1000 iterations, tracking the positions of 50 particles at the 200th, 400th, 600th, and 800th iterations. To evaluate particle diversity during the iterative process, we utilized PCA

to reduce the 30-dimensional particle position data to 3 dimensions. This analysis of the principal components offered valuable insights into particle behavior and enhanced swarm optimization. The findings are illustrated in Figures 8–20.

Table 5. CEC2017 benchmark functions.

NO.	Function	D	Range	f_{opt}
f_{17}	Shifted and Rotated Zakharov	30	[−100,100]	300
f_{18}	Shifted and Rotated Rastrigin	30	[−100,100]	500
f_{19}	Shifted and Rotated Lunacek Bi-Rastrigin	30	[−100,100]	700
f_{20}	Shifted and Rotated Non-Continuous Rastrigin	30	[−100,100]	800
f_{21}	Shifted and Rotated Schwefel	30	[−100,100]	1000
f_{22}	Hybrid Function 2 (N = 3)	30	[−100,100]	1200
f_{23}	Hybrid Function 4 (N = 4)	30	[−100,100]	1400
f_{24}	Hybrid Function 5 (N = 4)	30	[−100,100]	1500
f_{25}	Hybrid Function 6 (N = 4)	30	[−100,100]	1600
f_{26}	Composition Function 1 (N = 3)	30	[−100,100]	2100
f_{27}	Composition Function 2 (N = 3)	30	[−100,100]	2200
f_{28}	Composition Function 5 (N = 5)	30	[−100,100]	2500
f_{29}	Composition Function 6 (N = 5)	30	[−100,100]	2600

Table 6. Some variants of PSO and other compared evolutionary algorithms.

Algorithm	Years	Parameter Information
CLPSO [13]	2006	ω = 0.9~0.4, c = 1.49445, m = 7.
HPSO [29]	2014	ω = 0.9~0.4, $c_1 = c_2 = 2$, c_3 = randn [0,1].
THSPSO [30]	2019	$c_1 = c_2 = c_3 = 2$, ω = 0.9
TCSPSO [24]	2019	ω = 0.9~0.4, $c_1 = c_2 = 2$
BOA [31]	2018	Sensormodality c = 0~1, Powerexponent: 0.1~0.3, Switchprobability: p = 0~8
OSA [32]	2018	β = 1.9~0

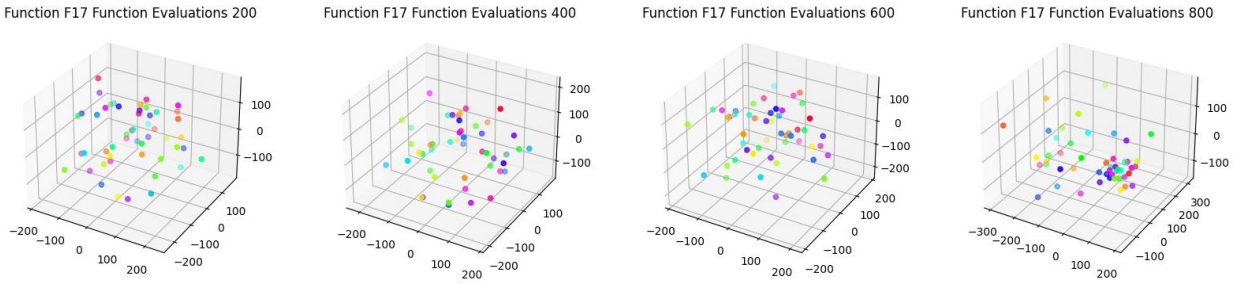


Figure 8. PCA was applied to the positions of 50 particles in f_{17} of CEC2017 for Shifted and Rotated Zakharov. Note: The colored dots are the positions of the particles.

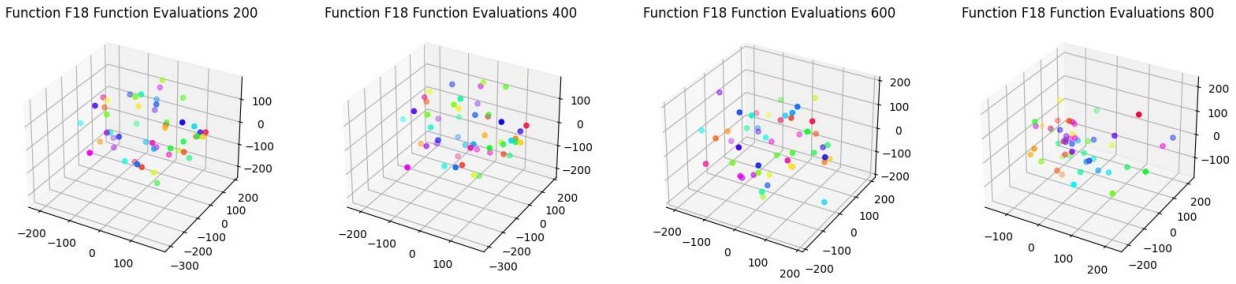


Figure 9. PCA was applied to the positions of 50 particles in f_{18} of CEC2017 for Shifted and Rotated Rastrigin. Note: The colored dots are the positions of the particles.

Table 7. Comparisons of values extracted by the CLPSO, HPSO, THSPSO, TCSPSO, BOA, and OSA algorithms in 50 trials.

Function		CLPSO	HPSO	THSPSO	TCSPSO	BOA	OSA	RLPSO	*t*-Test *s* Value
f_{17}	Mean	9.17×10^4	4.57×10^4	8.07×10^4	2.2×10^4	8.23×10^4	9.16×10^4	$\mathbf{3.13 \times 10^2}$	1
	Std	1.46×10^4	1.75×10^4	4.45×10^3	4.72×10^3	7.23×10^3	2.79×10^3	$\mathbf{1.75 \times 10^1}$	
f_{18}	Mean	6.57×10^2	6.12×10^2	8.99×10^2	6.02×10^2	8.99×10^2	9.38×10^2	$\mathbf{5 \times 10^2}$	1
	Std	1.05×10^1	5.04×10^1	3.8×10^1	2.54×10^1	2.59×10^1	2.35×10^1	$\mathbf{7.36 \times 10^{-4}}$	
f_{19}	Mean	9.53×10^2	8.58×10^2	1.38×10^3	$\mathbf{8.51 \times 10^2}$	1.36×10^3	1.47×10^3	9.06×10^2	1
	Std	1.64×10^1	4.07×10^1	5.56×10^1	$\mathbf{3.98 \times 10^1}$	4.04×10^1	4.32×10^1	5.96×10^1	
f_{20}	Mean	9.6×10^2	9.01×10^2	1.13×10^3	8.91×10^2	1.13×10^3	1.15×10^3	$\mathbf{8.09 \times 10^2}$	1
	Std	1.26×10^1	5.11×10^1	2.93×10^1	2.53×10^1	1.95×10^1	2.43×10^1	**5.18**	
f_{21}	Mean	6.26×10^3	8.05×10^3	8.75×10^3	4.85×10^3	8.84×10^3	9.04×10^3	$\mathbf{1.31 \times 10^3}$	1
	Std	2.88×10^2	5.99×10^2	5.95×10^2	9.05×10^2	3.11×10^2	4.36×10^2	$\mathbf{3.14 \times 10^2}$	
f_{22}	Mean	1.16×10^7	8.87×10^5	1.07×10^{10}	3.18×10^6	1.27×10^{10}	1.44×10^{10}	$\mathbf{4.18 \times 10^3}$	1
	Std	3.54×10^6	9.07×10^5	2.88×10^9	4.1×10^6	3.34×10^9	2.16×10^9	$\mathbf{3.03 \times 10^3}$	
f_{23}	Mean	6.13×10^4	6.8×10^4	1.56×10^6	6.6×10^4	1.22×10^7	2.14×10^7	$\mathbf{3.83 \times 10^3}$	1
	Std	4.41×10^4	5.24×10^4	9.77×10^5	1.02×10^5	2.09×10^7	2.03×10^7	$\mathbf{2.16 \times 10^3}$	
f_{24}	Mean	3.72×10^4	1.03×10^4	1.48×10^8	9.3×10^3	6.83×10^8	8.28×10^8	$\mathbf{1.63 \times 10^3}$	1
	Std	1.84×10^4	1×10^4	1.52×10^8	8.85×10^3	4.37×10^8	3.43×10^8	$\mathbf{1.88 \times 10^2}$	
f_{25}	Mean	2.47×10^3	2.65×10^3	5.04×10^3	2.67×10^3	7.3×10^3	6.17×10^3	$\mathbf{1.84 \times 10^3}$	1
	Std	1.6×10^2	3.51×10^2	6.82×10^2	3.1×10^2	1.39×10^3	9.21×10^2	$\mathbf{2.92 \times 10^2}$	
f_{26}	Mean	2.44×10^3	2.41×10^3	2.71×10^3	2.41×10^3	2.65×10^3	2.76×10^3	$\mathbf{2.24 \times 10^3}$	1
	Std	3.7×10^1	5.12×10^1	5.49×10^1	2.77×10^1	1.23×10^2	4.56×10^1	$\mathbf{2.71 \times 10^1}$	
f_{27}	Mean	2.94×10^3	7.04×10^3	8.2×10^3	2.74×10^3	5.44×10^3	1.01×10^4	$\mathbf{2.44 \times 10^3}$	1
	Std	1.07×10^3	3.2×10^3	1.29×10^3	1.36×10^3	1.08×10^3	6.9×10^2	$\mathbf{9.51 \times 10^1}$	
f_{28}	Mean	2.93×10^3	$\mathbf{2.89 \times 10^3}$	4.43×10^3	2.94×10^3	5.64×10^3	4.71×10^3	3.32×10^3	1
	Std	9.74	**1.45**	3.71×10^2	2.57×10^1	5.27×10^2	3.61×10^2	2.65×10^1	
f_{29}	Mean	4.81×10^3	4.99×10^3	1.01×10^4	4.7×10^3	1.14×10^4	1.14×10^4	$\mathbf{3.14 \times 10^3}$	1
	Std	4.67×10^2	5.5×10^2	6.96×10^2	1.19×10^3	9.03×10^2	8.81×10^2	**8.63**	

Note: a *t*-test s value of 1 indicates statistically significant differences in performances between the RLPSO algorithm and any other algorithm at a 95% confidence level, while a *t*-test s value of 0 suggests no statistically significant differences.

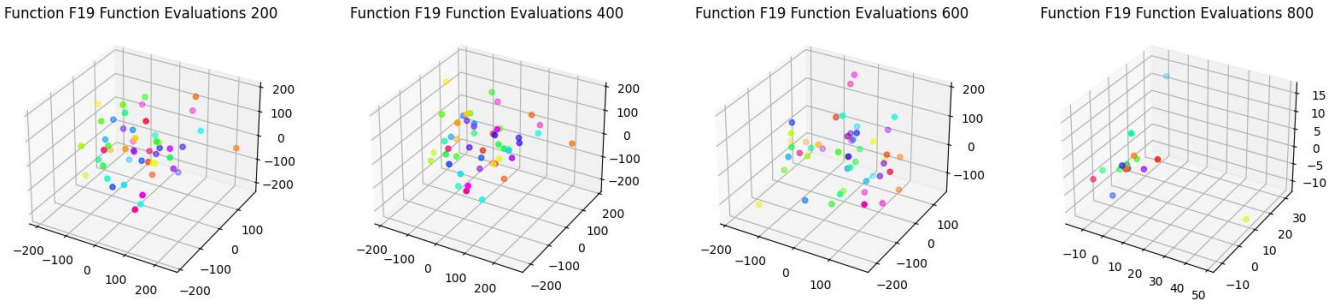


Figure 10. PCA was applied to the positions of 50 particles in f_{19} of CEC2017 for Shifted and Rotated Lunacek Bi-Rastrigin. Note: The colored dots are the positions of the particles.

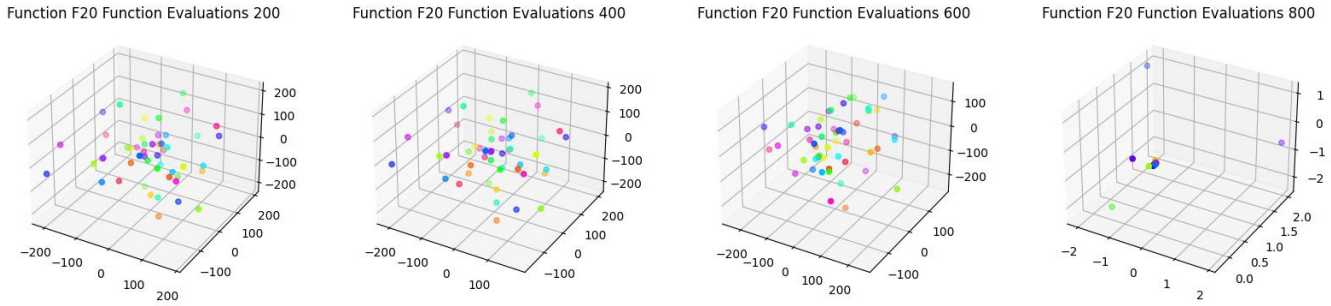


Figure 11. PCA was applied to the positions of 50 particles in f_{20} of CEC2017 for Shifted and Rotated Non-Continuous Rastrigin. Note: The colored dots are the positions of the particles.

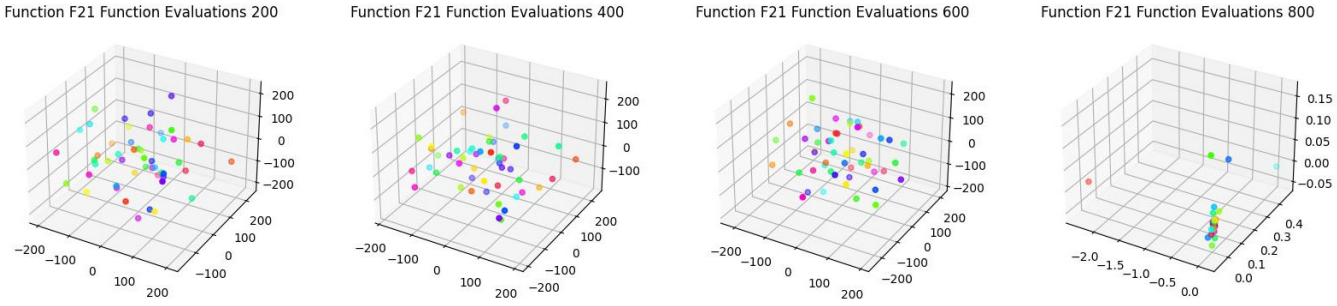


Figure 12. PCA was applied to the positions of 50 particles in f_{21} of CEC2017 for Shifted and Rotated Schwefel. Note: The colored dots are the positions of the particles.

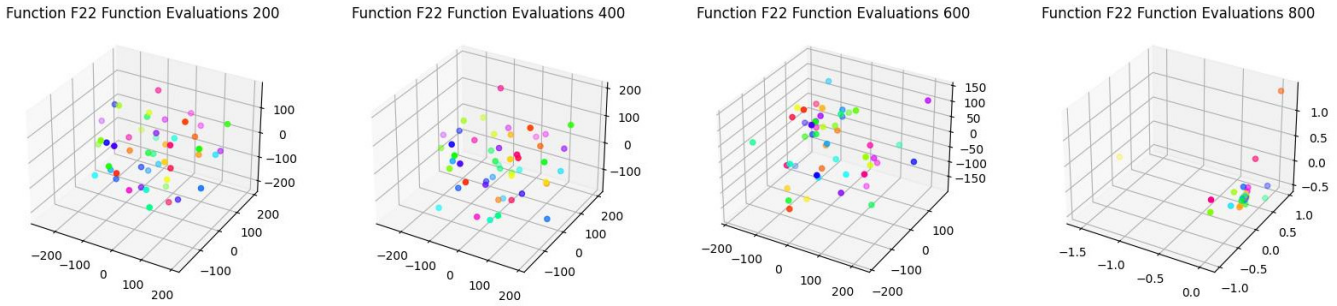


Figure 13. PCA was applied to the positions of 50 particles in f_{22} of CEC2017 for Hybrid Function 2 (N = 3). Note: The colored dots are the positions of the particles.

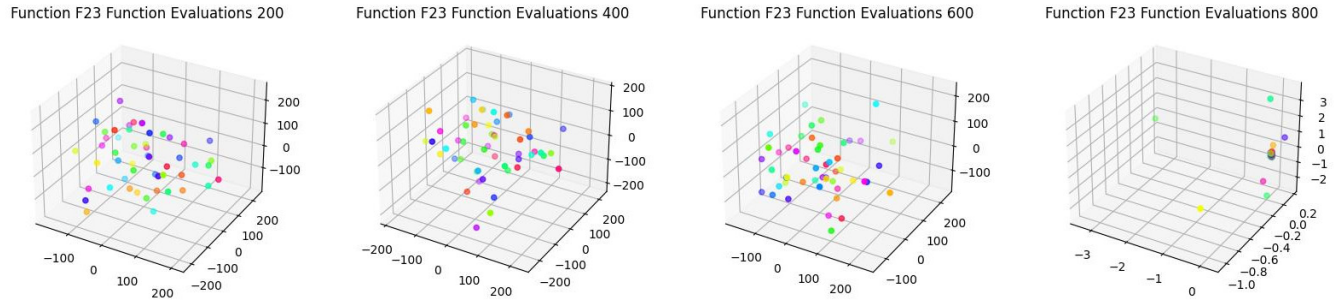


Figure 14. PCA was applied to the positions of 50 particles in f_{23} of CEC2017 for Hybrid Function 4 (N = 4). Note: The colored dots are the positions of the particles.

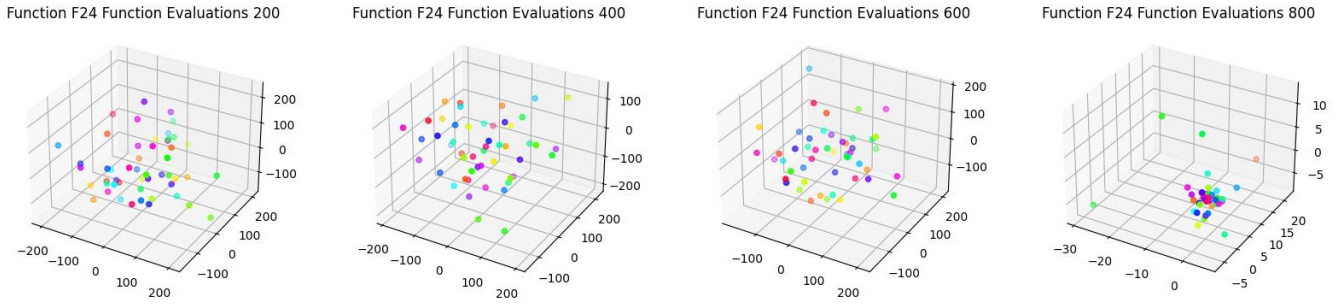


Figure 15. PCA was applied to the positions of 50 particles in f_{24} of Hybrid Function 5 (N = 4). Note: The colored dots are the positions of the particles.

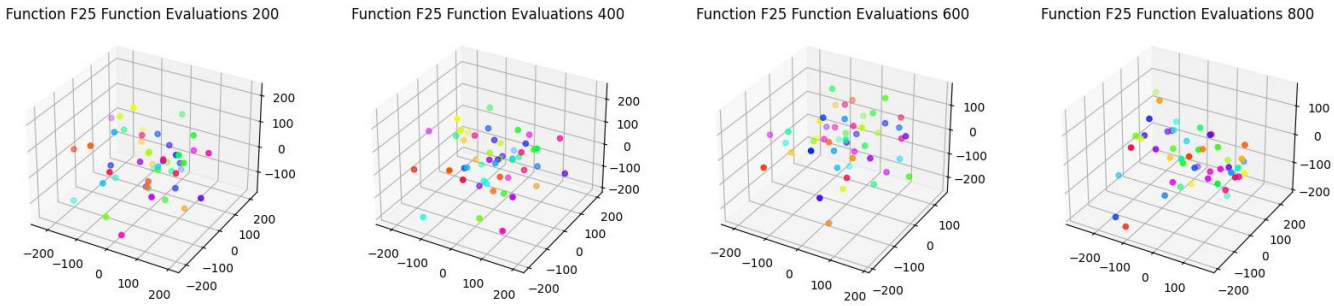


Figure 16. PCA was applied to the positions of 50 particles in f_{25} of Hybrid Function 6 (N = 4). Note: The colored dots are the positions of the particles.

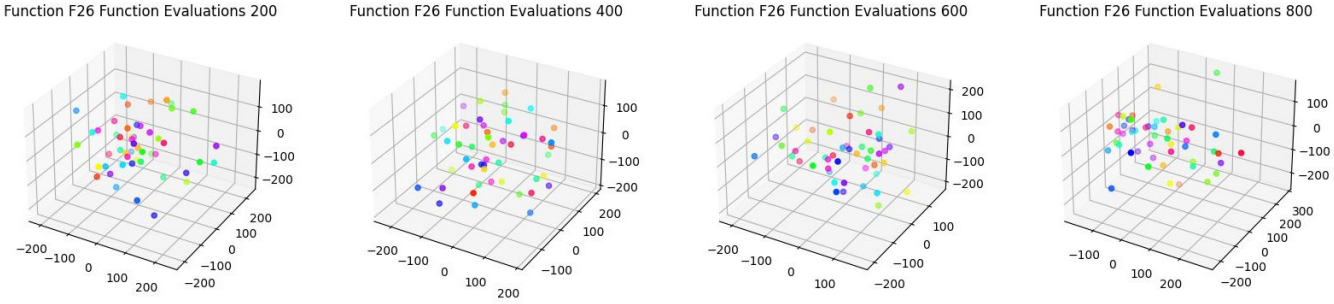


Figure 17. PCA was applied to the positions of 50 particles in f_{26} of Composition Function 1 (N = 3). Note: The colored dots are the positions of the particles.

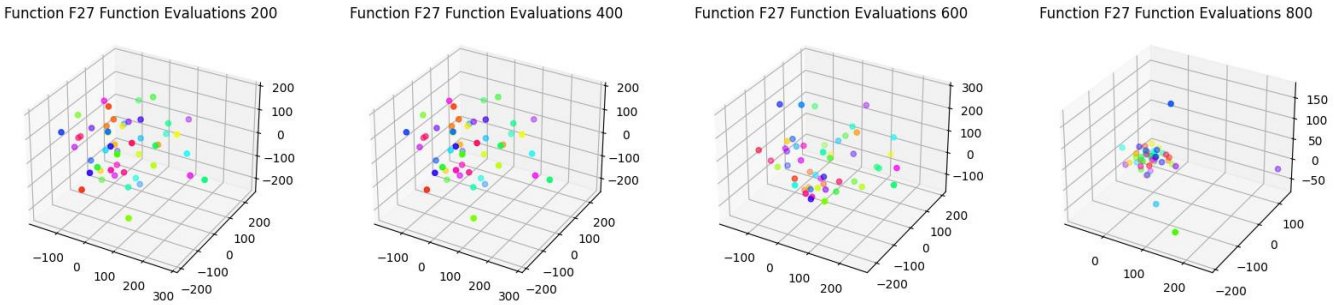


Figure 18. PCA was applied to the positions of 50 particles in f_{27} of Composition Function 2 (N = 3). Note: The colored dots are the positions of the particles.

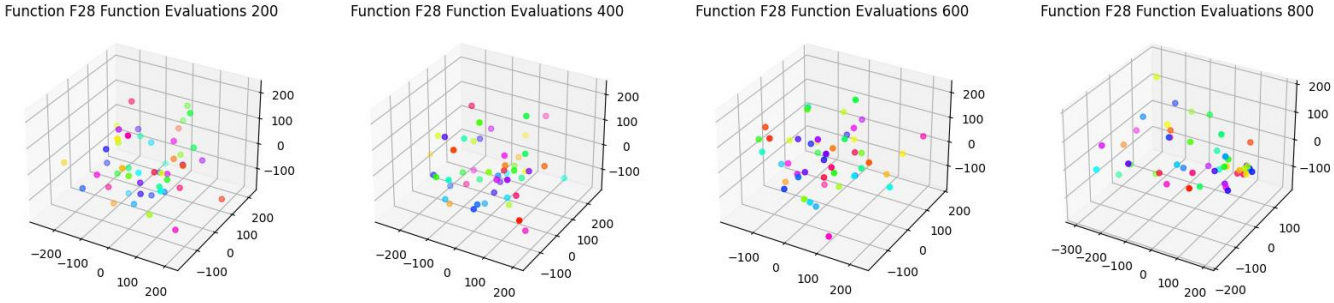


Figure 19. PCA was applied to the positions of 50 particles in f_{28} of Composition Function 5 (N = 5). Note: The colored dots are the positions of the particles.

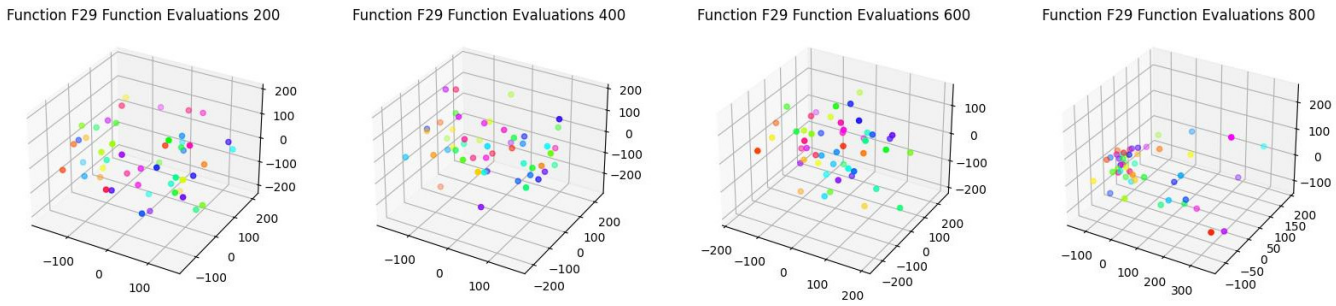


Figure 20. PCA was applied to the positions of 50 particles in f_{29} of Composition Function 6 (N = 5). Note: The colored dots are the positions of the particles.

From Figures 8–20, it is evident that the algorithm continues to improve beyond 600 function iterations and does not reach convergence. For certain functions (f_{17}, f_{18}, f_{25}, f_{26}, f_{28}, f_{29}), convergence is not achieved even after 800 function evaluations. Furthermore, the comparisons across different runs highlight the algorithm's stability for these functions.

To further analyze swarm diversity, a diversity measurement [33] is considered and defined as follows:

$$div^j = \frac{1}{N}\sum_{i=1}^{N}\left|median(x^j) - x_i^j\right| \tag{9}$$

$$div = \frac{1}{d}\sum_{j=1}^{d} div^j \tag{10}$$

N and d represent the number of particles and dimensions, respectively, x_i^j represents the j'th dimension of the i'th particle, while the median (x^j) is the median of dimension j in the whole swarm, div^j is the diversity in each dimension, the diversity of whole population (div) is calculated by averaging all div^j.

Furthermore, with the help of diversity measurement, we can calculate the percentage of exploration and exploitation during each iteration using the following equations:

$$exploration\% = \left(\frac{div}{div_{\max}}\right) \times 100\% \tag{11}$$

$$exploitation\% = 1 - exploration\% \tag{12}$$

$div_{\max}$ is defined as the maximum diversity value achieved throughout the optimization process. The *exploration%* connects the diversity in each iteration to this maximum diversity. It is inversely related to the exploitation level and is calculated as the complementary percentage to *exploitation%*.

We conducted research on the percentage of exploration and exploitation in 1000 iterations for f_{17}–f_{29}. The results are shown in Figure 21.

From Figure 21, we can observe that the *exploitation%* of the algorithm starts to increase around the 400th iteration. For f_{20} and f_{23}, when the number of iterations reaches 800, the *exploitation%* nearly approaches 100%. This observation is consistent with what is shown in Figures 11 and 14, where the particles are in a clustered state. The data in Figure 21 indicate that as the number of iterations increases, the algorithm's performance gradually improves and eventually stabilizes. This suggests that in the later stages of the algorithm's execution, the clustering of particles significantly enhances the exploitation rate, reaching nearly full exploitation.

Moreover, the analysis of the percentages of exploration and exploitation reveals that the algorithm predominantly explores in the early stages, with *exploration%* reaching nearly 100%. In the later stages, the algorithm shifts towards exploiting the accumulated information to accelerate convergence. This behavior is consistent across different problems,

highlighting the algorithm's stability and its ability to effectively balance exploration and exploitation in various scenarios.

4.5. Engineering Problem

In this section, the RLPSO algorithm was applied to solve the three-bar truss problem [33], and the maximum iteration and population size are 1000 and 30, respectively. The mathematical formulations are as follows:

$$\vec{x} = (x_1, x_2) \tag{13}$$

Objective function:

$$Min.f(x) = L * (x_2 + 2\sqrt{2}x_1) \tag{14}$$

These are subject to the following:

$$h_1(x) = \frac{x_2}{2x_2x_1 + \sqrt{2}x_1^2}P - \delta \leq 0, \tag{15}$$

$$h_2(x) = \frac{x_2 + \sqrt{2}x_1}{2x_2x_1 + \sqrt{2}x_1^2}P - \delta \leq 0, \tag{16}$$

$$h_3(x) = \frac{1}{x_1 + \sqrt{2}x_2}P - \delta \leq 0, \tag{17}$$

where $0 \leq x_1, x_2 \leq 1$, and $P = 2$, $L = 100$, and $\delta = 2$.

This engineering problem is solved by using our proposed RLPSO algorithm and compared with the methods mentioned in reference [33]. The results of the comparison are shown in Table 8.

As shown in Table 8, RLPSO performs well in solving the engineering problem. However, the complexity of the boundary conditions in this engineering problem often leads to particles not satisfying the constraints after updating, resulting in ineffective learning. Additionally, due to the problem's low dimensionality, particles lose a significant amount of their inherent information after updating, resulting in substantial changes in particle positions and the loss of valuable data. Consequently, addressing practical engineering problems with complex constraint conditions will be a major focus of our future research.

Table 8. Comparison performance of RLPSO with other algorithms for three-bar truss problem.

Algorithm	Optimal Weight
RLPSO	209.173475679612
m-DMFO	174.2761613819025
MFO	263.895979682
DEDS	263.8958434
MBA	263.8958522
Tsa	263.68
PSO-DE	263.8958433
CS	263.9716

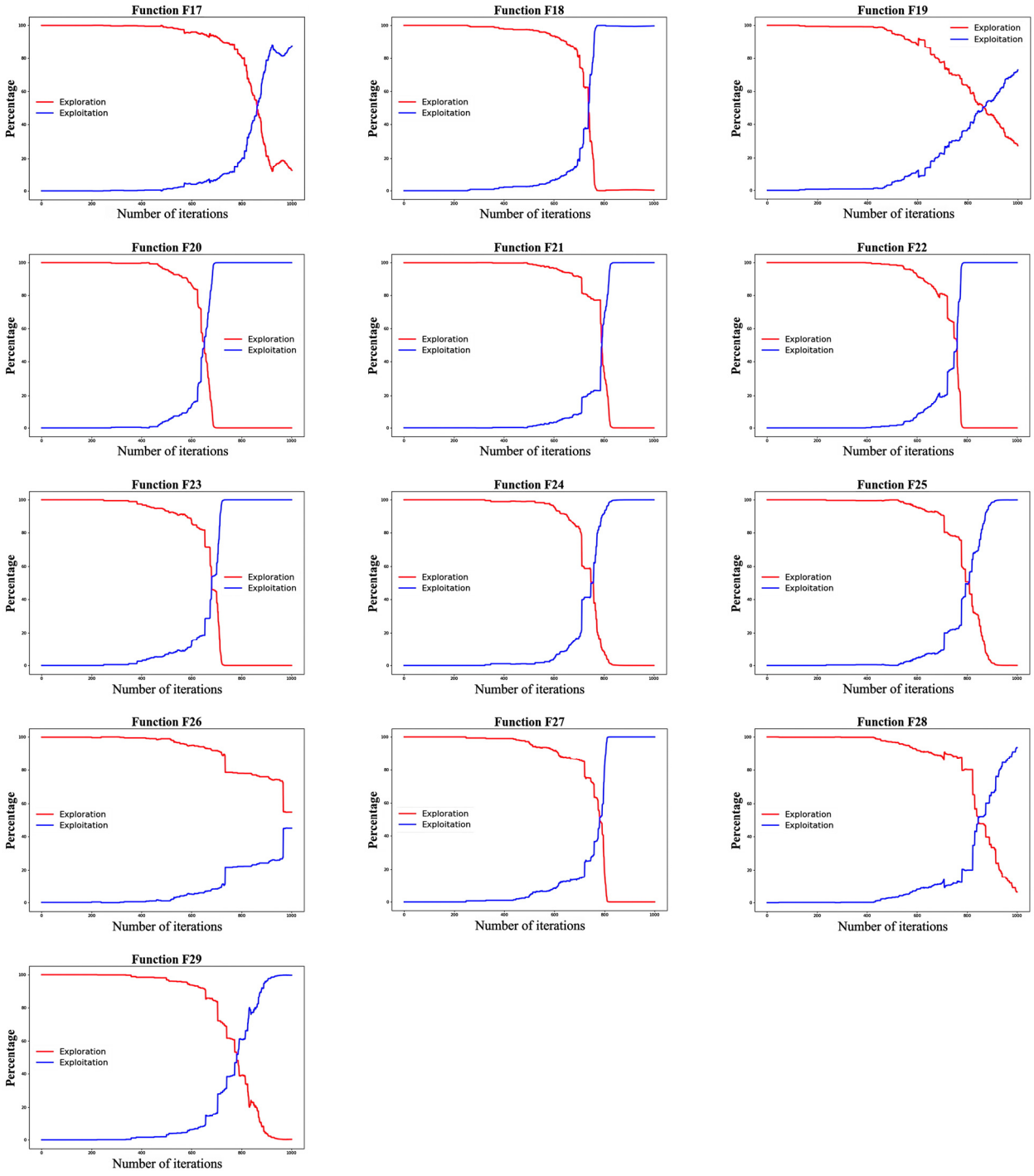


Figure 21. *exploration%* and *exploitation%* of f_{17}–f_{29}.

5. Conclusions

This study explores the application of strategic learning in optimization by proposing a Reinforcement Learning-based Particle Swarm Optimization (RLPSO) algorithm aimed at improving the performance and convergence speed of traditional PSO algorithms. The research on the RLPSO algorithm involves knowledge from multiple theoretical domains, including Particle Swarm Optimization, Reinforcement Learning, Q-learning, multi-modal optimization, and adaptive algorithms. This study is based on a deep understanding of these theories and their effective integration.

Under the same testing parameter settings, performance comparisons were conducted between the RLPSO algorithm and the PSO, CLPSO, and ELPSO algorithms on the testing of 16 benchmark functions from CEC2005. The results revealed that, compared to other algorithms, the RLPSO algorithm exhibited the fastest convergence speed. It also found the global optimum in 14 out of the 16 benchmark functions, showing significant statistical differences. In the testing of 13 benchmark functions from CEC2017, performance comparisons were made between the RLPSO algorithm and six other algorithms (CLPSO, HPSO, THSPSO, TCSPSO, BOA, and OSA). The results demonstrated that the RLPSO algorithm found the global optimum in 11 out of the 13 benchmark functions, with statistically significant differences. This indicates that the RLPSO algorithm exhibits excellent performance across multiple benchmark function problems, which finds the global optimum in almost all the cases. The introduced Q-learning mechanism in reinforcement learning plays a crucial role in enhancing algorithm performance.

This algorithm selects particles to update their velocities based on an online updated Q-table. At the beginning of each iteration, particles randomly choose particles to learn from with a certain probability, exploring the solution space as much as possible to update the Q-table. This process filters out particles worth learning from and stores the information in the Q-table. As iterations proceed, particles determine which particles they want to learn from based on the Q-table and store the learning results in the Q-table to guide the next step of learning. The algorithm continuously adjusts the learning targets and updates the learning strategy online to accelerate convergence speed at the right time, thus striking a good balance between convergence speed and diversity. In this paper, comparisons with random algorithms that do not incorporate Q-table learning reveal that the RLPSO algorithm converges faster. This indicates the crucial role of the Q-learning mechanism introduced in reinforcement learning in enhancing algorithm performance.

By combining Q-learning with particle swarm optimization, we achieve effective learning and experience sharing among particles, accelerating the algorithm's convergence speed, and obtaining better fitness values at the appropriate times. This demonstrates the effectiveness of strategic learning compared to random learning, providing strong support for further research and the application of reinforcement learning in optimization algorithms. Compared with traditional PSO algorithms and other improved versions, the RLPSO algorithm exhibits strong adaptability, fast convergence speed, strong global search capability, ease of implementation and application, and demonstrates more stable and efficient performance, making it more applicable and versatile when facing different types of optimization problems. The RLPSO algorithm has a wide range of applications, such as in engineering design, data mining, and artificial intelligence. By improving the performance and robustness of optimization algorithms, the RLPSO algorithm provides effective solutions for solving practical problems.

Additionally, in terms of performance on the CEC2005 and CEC2017 test functions, the RLPSO algorithm performs excellently in handling both multi-modal and single-modal problems, and is particularly outstanding in solving multi-modal problems, indicating its applicability in solving complex problems and further validating the superiority of the RLPSO algorithm over other PSO algorithms in solving practical problems. The diversity graphs also reveal that the algorithm maintains the diversity of the particle swarm throughout the computation, and balances the exploration and utilization of the algorithm. Through the engineering problem verification, it shows that the algorithm has a certain practical application value. This suggests that the RLPSO algorithm has broad potential in practical applications, especially in engineering optimization and data mining fields.

Despite the significant achievements of the RLPSO algorithm in optimization problems, there are limitations that need to be considered. The reinforcement learning parameters in the RLPSO algorithm need appropriate adjustments, including on learning rate, rewards, and penalties. The choice of these parameters may significantly affect the algorithm's performance and convergence speed, but determining the optimal parameter settings typically requires a large number of experiments and experiences. The updating and maintenance

of the Q-table in the RLPSO algorithm increase the computational cost, especially when dealing with high-dimensional problems or large-scale optimization tasks. Therefore, practical applications may face limitations in computational resources. Although the RLPSO algorithm demonstrates many advantages in optimization problems, further research and improvements are still needed to overcome its limitations and enhance the algorithm's performance and applicability.

Future research can focus on optimizing the parameter settings of Q-learning in the RLPSO algorithm to further enhance the algorithm's performance and robustness. Additionally, ideas from other novel PSO algorithms can be borrowed, such as the Particle Swarm Optimization algorithm with priority-based sorting [34] and the DOADAPO algorithm [35], to extend Q-learning into the multi-objective optimization domain, thereby exploring a wider problem space. These research directions will contribute to a deeper understanding of the working principles of the RLPSO algorithm and further promote the application and development of reinforcement learning-based optimization algorithms in practical problems.

In summary, as a reinforcement learning-based optimization algorithm, the RLPSO algorithm not only has significant theoretical significance but also has broad prospects in practical applications. We believe that through further research and exploration, the RLPSO algorithm will play a more important role in the field of optimization and provide effective solutions for practical problems.

Author Contributions: F.Z. and Z.C. proposed the algorithm and were responsible for writing the article. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the National Social Science Fund of China (2022-SKJJ-B-112).

Data Availability Statement: The datasets used and analyzed during the current study are available from the corresponding author on reasonable request.

Acknowledgments: We would like to thank all the authors whose articles are referenced in our study. We extend special thanks to the professors of the Sixty-third Institute of the National University of Defense Technology.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the 1995 IEEE International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; pp. 1942–1948.
2. Sabir, Z.; Raja, M.A.Z.; Umar, M.; Shoaib, M. Design of neuro-swarming-based heuristics to solve the third-order nonlinear multi-singular Emden-Fowler equation. *Eur. Phys. J. Plus* **2020**, *135*, 410. [CrossRef]
3. Che, H.; Wang, J. A Two-Timescale Duplex Neurodynamic Approach to Mixed-Integer Optimization. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 36–48. [CrossRef] [PubMed]
4. Shao, H.D.; Jiang, H.K.; Zhang, X.; Niu, M.G. Rolling bearing fault diagnosis using an optimization deep belief network. *Meas. Sci. Technol.* **2015**, *26*, 115002. [CrossRef]
5. Yan, X.A.; Jia, M.P. A novel optimized SVM classification algorithm with multi-domain feature and its application to fault diagnosis of rolling bearing. *Neurocomputing* **2018**, *313*, 47–64. [CrossRef]
6. Mohammad, Y.; Eberhart, R.; Mohammad, H.S. A Novel Flexible Inertia Weight Particle Swarm Optimization Algorithm. *PLoS ONE* **2016**, *11*, e0161558.
7. Ratnaweera, J.A.; Mousa, S.; Watson, H.C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Trans. Evol. Comput.* **2004**, *8*, 240–255. [CrossRef]
8. Liu, Y.; Lu, H.; Cheng, S.; Shi, Y. An Adaptive Online Parameter Control Algorithm for Particle Swarm Optimization Based on Reinforcement Learning. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation, Wellington, New Zealand, 10–13 June 2019; pp. 815–822.
9. Liu, W.; Wang, Z.; Yuan, Y.; Zeng, N.; Hone, K.; Liu, X. A novel sigmoid-function-based adaptive weighted particle swarm optimizer. *IEEE Trans. Cybern.* **2021**, *51*, 1085–1093. [CrossRef]
10. Mendes, R.; Kennedy, J.; Neves, J. The fully informed particle swarm: Simpler, maybe better. *IEEE Trans. Evol. Comput.* **2004**, *8*, 204–210. [CrossRef]
11. Yang, C.H.; Lin, Y.S.; Chang, L.Y.; Chang, H.W. A Particle Swarm Optimization-Based Approach with Local Search for Predicting Protein Folding. *J. Comput. Biol.* **2017**, *24*, 981–994. [CrossRef]

12. Bergh, F. A Cooperative approach to particle swarm optimization. *IEEE Trans. Evol. Comput.* **2004**, *8*, 225–239.
13. Liang, J.J.; Qin, A.K.; Suganthan, P.N.; Baskar, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans. Evol. Comput.* **2006**, *10*, 281–295. [CrossRef]
14. Huang, H.; Qin, H.; Hao, Z.F.; Lim, A. Example-based learning particle swarm optimization for continuous optimization. *Inf. Sci.* **2012**, *182*, 125–138. [CrossRef]
15. Lynn, N.; Suganthan, P.N. Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation. *Swarm Evol. Comput.* **2015**, *24*, 11–24. [CrossRef]
16. Zhang, X.M.; Lin, Q.Y. Three-learning strategy particle swarm algorithm for global optimization problems. *Inf. Sci. Int. J.* **2022**, *593*, 289–313. [CrossRef]
17. Xu, G.P.; Cui, Q.L.; Shi, X.H.; Ge, H.W.; Zhan, Z.H.; Lee, H.P.; Liang, Y.C.; Tai, R.; Wu, C.G. Particle swarm optimization based on dimensional learning strategy. *Swarm Evol. Comput.* **2019**, *45*, 33–51. [CrossRef]
18. Cai, L.; Hou, Y.; Zhao, Y.; Wang, J. Application research and improvement of particle swarm optimization algorithm. In Proceedings of the 2020 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS), Shenyang, China, 28–30 July 2020; pp. 238–241.
19. Garg, H. A hybrid PSO-GA algorithm for constrained optimization problems. *Appl. Math. Comput.* **2016**, *274*, 292–305. [CrossRef]
20. Uriarte, A.; Melin, P.; Valdez, F. An improved Particle Swarm Optimization algorithm applied to Benchmark Functions. In Proceedings of the 2016 IEEE 8th International Conference on Intelligent Systems (IS), Sofia, Bulgaria, 4–6 September 2016; pp. 128–132.
21. Wang, X.H.; Li, J.J. Hybrid particle swarm optimization with simulated annealing. In Proceedings of the 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), Shanghai, China, 26–29 August 2004; pp. 2402–2405.
22. Aydilek, I.B. A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems. *Appl. Soft Comput.* **2018**, *66*, 232–249. [CrossRef]
23. Liang, B.X.; Zhao, Y.; Li, Y. A hybrid particle swarm optimization with crisscross learning strategy. *Eng. Appl. Artif. Intell.* **2021**, *105*, 104418. [CrossRef]
24. Zhang, X.W.; Liu, H.; Zhang, T.; Wang, Q.W.; Tu, L.P. Terminal crossover and steering-based particle swarm optimization algorithm with disturbance. *Appl. Soft Comput.* **2019**, *85*, 105841. [CrossRef]
25. Babak, Z.A. No-free-lunch-theorem: A page taken from the computational intelligence for water resources planning and management. *Environ. Sci. Pollut. Res. Int.* **2023**, *30*, 57212–57218.
26. Xu, L.; Zhu, S.; Wen, N. Deep reinforcement learning and its applications in medical imaging and radiation therapy: A survey. *Phys. Med. Biol.* **2022**, *67*, 22. [CrossRef] [PubMed]
27. Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D. Thesis, Cambridge University, Cambridge, UK, 1989.
28. Suganthan, P.N.; Hansen, N.; Liang, J.J.; Deb, K.; Tiwari, S. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. In *Natural Computing*; Nanyang Technological University: Singapore, 2005; pp. 341–357.
29. Liu, H.; Zhang, Y.; Tu, L.; Wang, Y. Human Behavior-Based Particle Swarm Optimization: Stability Analysis. In Proceedings of the 2018 37th Chinese Control Conference (CCC), Wuhan, China, 25–27 July 2018; pp. 3139–3144.
30. Liu, Y.J. A hierarchical simple particle swarm optimization with mean dimensional information. *Appl. Soft Comput.* **2019**, *76*, 712–725. [CrossRef]
31. Arora, S.; Singh, S. Butterfly optimization algorithm: A novel approach for global optimization. *Soft Comput.* **2019**, *23*, 715–734. [CrossRef]
32. Jain, M.; Maurya, S.; Rani, A.; Singh, V. Owl search algorithm: A novel nature-inspired heuristic paradigm for global optimization. *J. Intell. Fuzzy Syst.* **2018**, *34*, 1573–1582. [CrossRef]
33. Sahoo, S.K.; Saha, A.K.; Nama, S.; Masdari, M. An improved moth flame optimization algorithm based on modified dynamic opposite learning strategy. *Artif. Intell. Rev.* **2022**, *56*, 2811–2869. [CrossRef]
34. Wang, Y.J.; Yang, Y.P. Particle swarm optimization with preference order ranking for multi-objective optimization. *Inf. Sci.* **2009**, *179*, 1944–1959. [CrossRef]
35. Deng, W.; Zhao, H.M.; Yang, X.H.; Xiong, J.X.; Sun, M.; Li, B. Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment. *Appl. Soft Comput.* **2017**, *59*, 288–302. [CrossRef]