



Reinforcement-learning-based parameter adaptation method for particle swarm optimization

Shiyuan Yin¹ · Min Jin¹ · Huaxiang Lu^{1,2,3,4,5} · Guoliang Gong¹ · Wenyu Mao¹ · Gang Chen¹ · Wenchang Li¹

Received: 1 August 2022 / Accepted: 17 February 2023 / Published online: 25 March 2023
© The Author(s) 2023

Abstract

Particle swarm optimization (PSO) is a well-known optimization algorithm that shows good performances in solving different optimization problems. However, the PSO usually suffers from slow convergence. In this article, a reinforcement-learning-based parameter adaptation method (RLAM) is developed to enhance the PSO convergence by designing a network to control the coefficients of the PSO. Moreover, based on the RLAM, a new reinforcement-learning-based PSO (RLPSO) algorithm is designed. To investigate the performance of the RLAM and RLPSO, experiments on 28 CEC 2013 benchmark functions were carried out to compare with other adaptation methods and PSO variants. The reported computational results showed that the proposed RLAM is efficient and effective and that the proposed RLPSO is superior to several state-of-the-art PSO variants.

Keywords Particle swarm optimization · Reinforcement learning · CEC 2013 benchmark

Introduction

In recent years, the research community has witnessed an explosion of literature in the area of swarm and evolutionary computation [1]. Hundreds of novel optimization algorithms have been reported over the years and applied successfully in many applications, e.g., system reliability optimization [2], DNA sequence compression [3], systems of boundary value problems [4], solving mathematical equations [5,6], object-level video advertising [7], and wireless networks [8]. Particle swarm optimization (PSO), which was first proposed in 1995 by Kennedy and Eberhart [9], is one of the most famous swarm-based optimization algorithms. Due to its simplicity and high performance, a multitude of enhancements have been presented on PSO during the last few decades, which can be simply categorized into three types: parameter selection, topology, and hybridization with other algorithms [10].

When solving different optimization problems, appropriate parameters need to be configured for PSO and its variants. The performance of PSO heavily depends on the parameter settings, which shows the importance of setting the parameter values. However, it is very difficult for humans to conduct a laborious preprocessing phase for the detection of promising parameter values and design the strategy to monitor the running state of PSO and then adjust the running parameters [11].

✉ Guoliang Gong
gongmianjie@semi.ac.cn

Shiyuan Yin
yinshiyuan@semi.ac.cn

Min Jin
jinmin08@semi.ac.cn

Huaxiang Lu
luhx@semi.ac.cn

Wenyu Mao
maowenyu@semi.ac.cn

Gang Chen
chengang08@semi.ac.cn

Wenchang Li
liwc@semi.ac.cn

- ¹ High Speed Circuit and Neural Network Laboratory, Institute of Semiconductors, CAS, A 35 Qinghua East Road, Beijing, Beijing 100083, China
- ² University of Chinese Academy of Sciences, Beijing, China
- ³ Semiconductor Neural Network Intelligent Perception and Computing Technology Beijing Key Laboratory, Beijing, China
- ⁴ Collage of Microelectronics, University of Chinese Academy of Sciences, Beijing, China
- ⁵ Materials and Optoelectronics Research Center, University of Chinese Academy of Sciences, Beijing, China

The current parameter setting algorithms are mainly divided into two categories: parameter tuning and parameter control. The classifications are shown in Fig. 1. Parameter tuning is relatively simple to implement, but because all its designs are determined before the optimization algorithm runs, it loses some adaptability to the problem when running. In contrast, parameter control will monitor the operation of the optimization algorithm to control the operation of the algorithm. Furthermore, the types of parameter control have been clearly defined, distinguishing deterministic, adaptive, or self-adaptive methods. Only adaptive methods are informed as they receive feedback from the EA run and assign values based on that feedback [12].

Thus, this paper focuses on adaptive methods.

Common parameter adaptation methods include control based on history, control based on test experiments, control based on fuzzy logic, and control based on reinforcement learning. In the existing adaptive methods, almost all the control rules need to be learned when the algorithm is running. Although fuzzy logic has pre-configured rules, it also needs to be manually configured item by item, which makes the current adaptation algorithms inefficient. In addition, these adaptation algorithms are usually designed for a certain optimization algorithm and cannot be applied to various optimization algorithms. However, in the image domain [13] and natural language processing domain [14], the pre-training models are very mature, which greatly improves the performance of subsequent tasks. This inspired us to use reinforcement learning to improve the particle swarm performance through pre-training. Using the adaptation method based on RL will bring the following benefits: 1. There is no need to manually design rules and tune parameters, which greatly reduces the burden on users. Although the parameters in RL are also very important, experiments have shown that using the same set of parameters achieves good results for different algorithms and test functions. Therefore, in the actual use process, it is not necessary to adjust the parameters in RL. 2. By learning from past experience, the application of the algorithm is more extensive, and the effect is better.

In this paper, we propose a reinforcement-learning-based parameter adaptation method (RLAM) by embedding a deep

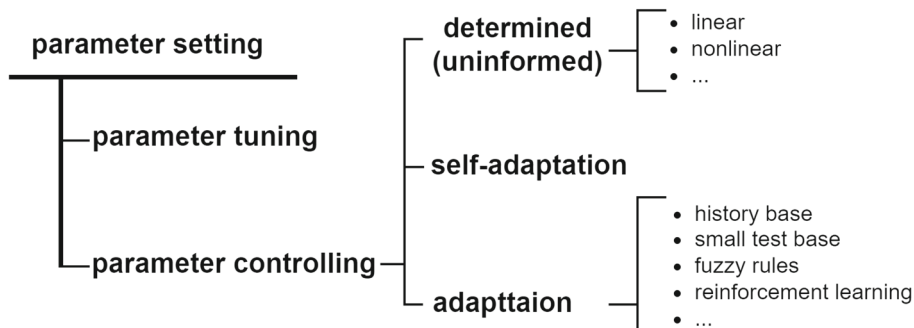
deterministic policy gradient (DDPG) [15] into the process of PSO. In the proposed method, there are two neural networks: the actor network and the action-value network. The actor network is trained to help the particles in the PSO choose their best parameters according to their states. The action-value network is trained to evaluate the performance of the actor network and provide gradients for the training of the actor network. The input of the actor network includes three parts: the percentage of iterations, the percentage of no-improvement iterations, and the diversity of the swarm. All the particles will be divided into several groups, and each group has their own action generated by the actor network. Typically, the action controls w , $c1$, and $c2$ in the PSO, but it has the ability to control any parameters if needed. A reward function is needed to train the two networks. The design is very simple and targeted at encouraging the PSO to obtain a better solution in every iteration. The whole process of training and utilizing the two networks is described in “Deep deterministic policy gradient”.

To evaluate the performance of the method proposed in this paper, three sets of experiments were designed: 1. Six particle swarm algorithm variants were selected, and their performances combined with the RLAM were compared with their original performances. 2. The RLAM was combined with the original PSO and compared with other adaptation methods. 3. A new reinforcement-learning-based PSO (RLPSO) was designed based on this method, and its performance was compared with five particle swarm variants and several advanced algorithms proposed in recent years. These experiments verified the effectiveness of the RLAM.

Main contributions of this paper

1. The reinforcement learning algorithm (DDPG algorithm) is introduced into the adaptation process, which greatly improves the parameter adaptation ability.
2. The concept of pre-training is introduced into the adaptation process so that the particle swarm algorithm can not only adapt the parameters based the current situation but also through past experience, which improves the intelligence level of the algorithm.

Fig. 1 Adaptation schemes



- Based on the above improvements, a adaptive particle swarm optimization algorithm based on reinforcement learning (RLPSO) is proposed, which greatly improves the performance of the PSO.

Structure of paper

The rest of this paper is organized as follows. The definitions of the PSO and DDPG are presented in “Background information”. Related work on parameter setting is introduced in “Related work”. The implementation of the proposed RLAM and RLPSO algorithms is described in “Proposed Algorithm”. Experimental studies are presented in “Experiments”. Conclusions summarizing the contributions of this paper are presented in “Conclusions”.

Background information

Particle swarm optimization

Particle swarm optimization is the representation of swarm behaviors in some ecological systems, such as birds flying and bees foraging [16]. In the classic PSO, the movement of a particle is influenced by its own previous best position and the best position of the best particle in the swarm. To describe the state of the particles, the velocity V_i and position X_i of the i^{th} particle are defined as follows:

$$V_i = (v_i^1, v_i^2, \dots, v_i^D), i = 1, 2, \dots, N$$

$$X_i = (x_i^1, x_i^2, \dots, x_i^D), i = 1, 2, \dots, N$$

where D represents the dimension of the search space, and N represents the number of particles. As the search progresses, the two movement vectors are updated as follows:

$$V_i(t + 1) = wV_i(t) + c1 * r1 (pBest_i - X_i(t)) + c2 * r2 (gBest - X_i(t))$$

$$X_i(t + 1) = X_i(t) + V_i(t + 1)$$

where w is the inertia weight, $c1$ is the cognitive acceleration coefficient, $c2$ is the social acceleration coefficient, $r1$ and $r2$ are uniformly distributed random numbers within $[0, 1]$, $V_i(t)$ denotes the velocity of the i^{th} particle in the t^{th} generation, $pBest_i$ is the personal best position for the i^{th} particle, and $gBest$ is the best position in the swarm.

Reinforcement learning and deep deterministic policy gradient

This subsection introduces the DDPG and RL briefly. Further details can be found elsewhere [15].

Reinforcement learning

Reinforcement learning (RL) is a kind of machine learning. Its purpose is to guide the agent to perform optimal actions in the environment to maximize the cumulative reward. The agent will interact with the environment in discrete timesteps. At each timestep t , the agent receives an observation $s_t \in S$, takes an action $a_t \in A$, and receives a scalar reward $r_t(s_t, a_t)$. S is the state space, and A is the action space. The agent’s behavior is controlled by a policy $\pi : S \rightarrow A$, which maps each observation to an action.

Deep deterministic policy gradient

In the DDPG, there are four neural networks designed to obtain the best policy π : the actor network $\mu(s_t|\theta^\mu)$, target actor network $\mu'(s_t|\theta^{\mu'})$, action-value network $Q(s_t, a_t|\theta^Q)$, and target action-value network $Q'(s_t, a_t|\theta^{Q'})$. μ and μ' are used to choose the action based on the state, Q and Q' are used to evaluate the action chosen by the actor network. θ^μ , $\theta^{\mu'}$, θ^Q , and $\theta^{Q'}$ are the neural network weights of these neural networks. Initially, $\theta^{\mu'}$ is a copy of θ^μ and $\theta^{Q'}$ is a copy of θ^Q . During training, the weights of these target networks are updated as follows:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'} \tag{1}$$

where $\tau \ll 1$.

To train the action-value network, we need to minimize the loss function:

$$L(\theta^Q) = (r(s_t, a_t) + \gamma Q'(s_{t+1}, a_{t+1} | \theta^{Q'}) - Q(s_t, a_t | \theta^Q))^2 \tag{2}$$

Then, the action-value network is used to train the actor network with the policy gradient

$$\nabla_{\theta^\mu} J = \nabla_a Q(s, a | \theta^Q) \Big|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s | \theta^\mu) \Big|_{s=s_t} \tag{3}$$

The data flow of the training process is shown in Fig. 2. In this figure, critic is the action-value network.

Comprehensive learning particle swarm optimizer (CLPSO)

In PSO, all particles are attracted by their own historical optima and the optima of all particles. The global optimum

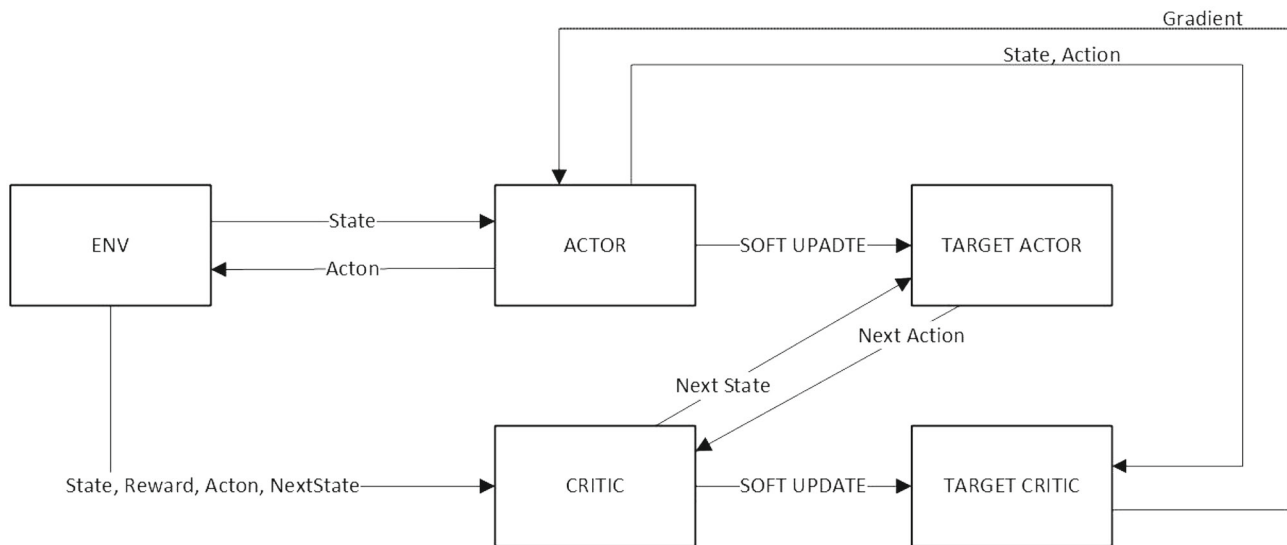


Fig. 2 Training process of deep deterministic policy gradient (DDPG)

may be inside a local minimum, which will cause the particle swarm to prematurely converge to the wrong location. To solve this problem, we introduce the mechanism of the comprehensive learning particle swarm optimizer (CLPSO) in RLPSO, which allows particles to randomly learn the dimensions of different particles. Therefore, this section will introduce the CLPSO.

More details on the CLPSO can be found elsewhere [17]. The velocity updating equation used by the CLPSO is as follows:

$$v_i^d(t+1) = wv_i^d(t) + c * r * (pbest_{f_i(d)}^d - x_i^d(t)), \quad (4)$$

where $f_i(d) = [f_i(1), f_i(2), \dots, f_i(D)]$ defines which particle's pbest the i^{th} particle should follow. The d^{th} dimension of the i^{th} particle should follow the $f_i(d)^{th}$ particle's pbest in the d^{th} dimension. r is a random number between 0 and 1. w and c are coefficients.

To determine $f_i(d)$, every particle has its own learning parameter Pc_i . The Pc_i value for each particle is calculated by the following equation:

$$Pc_i = a + b * \frac{\left(\exp\left(\frac{10(i-1)}{ps-1}\right) - 1 \right)}{\exp(10) - 1}, \quad (5)$$

where ps is the population size, $a = 0.05$, and $b = 0.45$. When a particle updates its velocity for one dimension, a random value in $[0, 1]$ is generated and compared with Pc_i . If the random value is larger than Pc_i , the particle of this dimension will follow its own pbest. Otherwise, it will follow another particle's pbest for that dimension. CLPSO employs a tournament selection to choose a target particle. Furthermore, to avoid wasting function evaluations in the wrong direction, CLPSO defines a certain number of evaluations as

the refreshing gap m . During the period in which a particle follows a target particle, the number of times the particle ceases to improve is recorded as $flag_{clpso}$. If $flag_{clpso}$ is bigger than m , the particle will obtain its new target particle by employing a tournament selection again.

Related work

When solving different optimization problems, appropriate parameters need to be configured for PSO and its variants. The performance of PSO heavily depends on the parameter setting, which shows the importance of setting the parameter values. The classifications are shown in Fig. 1. We will briefly review them next.

Parameter tuning

There are many parameters that need to be configured in the particle swarm algorithm, and these parameters are related to the quality of the final optimization result. Therefore, the parameters can be tested in groups, the parameter groups can be tested on all the test problems, and the parameter group with the best effect can be selected.

Since the end of last century, a number of automatic parameter tuning approaches have been put forward, such as Design of experiments [18], F-Race [19], ParamILS [20], REVAC [21], and SPO [22]. SMAC [23] and TPE [24] are the most commonly used methods in this field.

Determined parameter control

Determined parameter control refers to the adaptive methods whose parameters are determined before the actual optimiza-

tion problem is solved, which can be mainly divided into two categories based on the optimization progress of the optimization problem: linear parameter variation and nonlinear parameter variation.

Linear strategies

Initially, all the parameters in the particle swarm algorithm were set to fixed values [9], but researchers have found that this method is not efficient, and it is difficult to balance the relationship between exploration and exploitation. Therefore, linear strategies have been proposed. A linear strategy means that algorithms pre-specify a linear expression during the running process and determine some of the current running parameters based on the running state.

In the paper [25], the researchers proposed a linearly reduced w parameter, which improved the fine-tune capability of the PSO in the later stage. The value of w decreased linearly from an initial value w_{max} to w_{min} . The specific formula for this change is as follows:

$$\omega(t) = \frac{t_{max} - t}{t_{max}} (\omega_{max} - \omega_{min}) + \omega_{min}$$

Then, in the paper [26], the researchers proposed a linear variation operation for three parameters. In addition to the linear decrease of w , $c1$ was increased linearly and $c2$ was decreased linearly. These parameters were preset with their maximum and minimum values. This method makes the PSO algorithm more inclined to search near the optimal value that it found in the early stage, which improves the particle’s exploration ability and improves its fine-tuning ability in the later stage, allowing it to quickly converge to the global optimal value. The formulas for the changes of $c1$ and $c2$ are shown as follows, and the change of w is consistent with the above formula:

$$\begin{aligned} c1 &= (c1_f - c1_i) \frac{iter}{MAXITR} + c1_i \\ c2 &= (c2_f - c2_i) \frac{iter}{MAXITR} + c2_i \end{aligned} \tag{6}$$

In the paper [27], the researchers constructed a linearly growing w , which could also achieve a better performance in some of the problems given in the paper. The equation is as follows:

$$\omega(t) = 0.5 \times \frac{t}{t_{max}} + 0.4$$

Considering that both the linear growth and decline of w have advantages in some problems, in the paper [28], the researchers constructed a w that linearly increased first and then linearly decreased. The specific changes are expressed

as follows:

$$\omega(t) = \begin{cases} 1 \times \frac{t}{t_{max}} + 0.4, & 0 \leq \frac{t}{t_{max}} \leq 0.5 \\ -1 \times \frac{t}{t_{max}} + 1.4, & 0.5 < \frac{t}{t_{max}} \leq 1 \end{cases}$$

In addition to the above-mentioned linear adjustment according to the running progress of the algorithm, there were also studies in which the parameters were linearly adjusted based on some other parameters.

For example, in the paper [29], the researchers adjusted the w value based on the average distance between particles. The specific adjustment method is as follows:

$$\omega(t) = \frac{\omega_{max} - \omega_{min}}{D_{max} - D_{min}} \times D(t) + \frac{D_{max}\omega_{min} - D_{min}\omega_{max}}{D_{max} - D_{min}}$$

where $D(t)$ denotes the average distance between particles, and ω_{max} and ω_{min} are predefined.

Nonlinear strategies

Inspired by the various linear strategies, many researchers have turned their attention to nonlinear strategies. These methods make parameter changes more flexible and closer to the needs of the problem.

In the paper [30], the author proposed the use of the E index to update w , which improves the convergence speed of the algorithm on some problems. In the paper [31], the author used a quadratic function to update the parameters. The test results showed that this method outperformed the linear transformation algorithm in most continuous optimization problems. In the paper [32], the author combined the sigmoid function into a linear transformation, allowing the algorithm to quickly converge in the search process. In the papers [33] and [34], the authors applied a chaotic model (logistic map) to the parameter transformation, which gave the algorithm a stronger search ability. Similarly, in the paper [35], the author set the parameters to random numbers and also obtained good results.

Adaptation parameter controlling

Adaptation parameter controlling are techniques for the dynamic adaptation of the algorithm’s parameters during its execution. They are typically based on performance feedback from the algorithm on the considered problem instance.

History based

The approaches proposed in some papers divide the whole running process into many small processes, using different parameters or strategies in each small process and judging whether the parameters or strategies are good enough based

on the performance of the algorithm at the end of the small stage. The strategies or parameters that are good enough will be chosen more in subsequent runs. In the paper [36], the author built a parameter memory. In each run, all the running particles were assigned different parameter groups, and the parameters used by some particles with better performances after the small-stage run was completed were saved in the parameter memory. The parameters chosen by subsequent particles tended to be close to the average of the parameter memory. In the paper [37], the author designed five particle swarm operation strategies based on some excellent particle swarm variant algorithms in the past and kept a record of the success rate for each strategy. In the initial state, all the success rates were set to 50%, and then in each small process, a strategy was randomly selected for execution based on the weight of the past success rate. Based on the number of particles promoted, the memory of the success rate was updated. In the paper [38], the author improved the original EPSO, updated the designed strategy, divided the particle swarm into multiple subgroups, and evaluated the strategy separately, which further improved the performance of the algorithm.

Small test period based

This approach divides the entire optimization process into multiple sub-processes and divides each sub-process into two parts. One part is used to test the performance of the parameters or strategies, and the number of evaluations is generally less than 10%. The other part is the normal optimization process. In the paper [39], the author divided the parameters into many parameter groups based on the grid. In each performance test process, the population was divided into multiple subgroups, and the parameter groups around the grid of the previous process were tested. After running several iterations, the parameter group with the best effect was selected as the parameter group to be used. In the paper [40], the author divided the test sub-process into two processes. In the first process, the adjacent points of the current selection point were evaluated, and the probability of success was calculated. Then, in the second process, the direction with the highest probability of success was selected as the exploration direction, multiple steps in this direction were explored, and finally, the best parameter group in the second step was selected as the parameter configuration for the next operation. This method further improved the performance.

Fuzzy rules

In the paper [41], the author presented a new method for dynamic parameter adaptation in the PSO, where an improvement to the convergence and diversity of the swarm in the PSO using interval type-2 fuzzy logic was proposed. The

experimental results were compared with the original PSO, which showed that the proposed approach improved the performance of the PSO. In the paper [42], the author presented work to improve the convergence and diversity of the swarm in the PSO using type-1 fuzzy logic applied it to classification problems.

Reinforcement learning based

At present, most optimization algorithms based on reinforcement learning are based on Q-learning for adaptive control parameters and strategies. In the paper [43], the author combined a variety of topological structures, using the diversity of the particle swarm and the topological structure of the previous step as the state, and selected the topological structure with the largest Q value from the Q table as the topology to be used in the next optimization step. In the paper [44], the author used the Q-learning algorithm. Enter the state with how far from the optimal value and the ranking of the particle evaluation value among all particles. Different parameter groups were used as adaptation targets. The reward was determined based on whether the entire optimization problem grew, the current optimization progress, and the currently selected action. In the paper [45], the author used the Q-learning algorithm. The strategy selected in the previous step was the state input. The output actions controlled different strategies, such as jumping or fine-tuning. Finally, the reward was determined based on whether the whole optimization problem grew. In the paper [46], the author used the Q-learning algorithm. The particle position was taken as the state input. The output action controlled the prediction speeds of different particle strategies. Finally, the reward was determined based on the increase or decrease of the particle evaluation value. In the paper [47], the author used the Q-learning algorithm with the serial number of each particle, the particle position as the state input, and the output action to control two different strategies, and finally, the reward was determined according to whether the entire optimization problem grew. In the paper [48], the author adopted the algorithm of the policy gradient, which took the particle position pbest as the state input and the output action controls $c1$ and $c2$ and finally determined the reward based on the growth rate of the entire optimization problem.

A summary of these papers is shown in Table 1, which describes the method used in the paper, the state input, the action output, the reward input, and whether each particle was controlled separately. In this table, Q represents Q-Learning and PG represents Policy Gradient.

Section summary and motivation

First, current algorithms are not smart enough. If a human solves an optimization problem, he optimizes based on

Table 1 Reinforcement-learning-based adaptive methods

| Paper | Input (adaptation source) | Output (adaptation target) | Reward | Single particle | Algorithm |
|-------|-----------------------------|----------------------------|------------------|-----------------|-----------|
| [43] | Topological structure | Next topological structure | Diversity, gbest | No | Q |
| [44] | Distance with the best/rank | $c1, c2, w$ | gbest | Yes | Q |
| [45] | Previous strategy | Strategy | gbest | Yes | Q |
| [46] | Particle position | Velocity | Evaluate fitness | Yes | Q |
| [47] | Particle index/position | Strategy | gbest | Yes | Q |
| [48] | Position, pbest | $c1, c2$ | gbest | Yes | PG |

past experience and the current operating state. The current parameter tuning methods do not do this. They all adjust the policy during operation without using past experience. For the same type of problem, such as path planning, the algorithm proposed in this article can be trained in advance through various test problems. After that, the algorithm will remember the experience of this training. As a result, the convergence accuracy on new problems will be improved. The existing parameter adaptive method cannot be trained in advance and can only collect experience while operating.

Second, the current reinforcement learning methods introduced in parameter tuning methods all suffer from some performance issues. The currently used reinforcement learning method is mainly Q-learning, and another researcher used Policy Gradient.

Since Q-learning can only choose from a limited number of actions (usually four actions in past papers), this leads to the necessity of artificially mapping a high-dimensional continuous action space with a limited number of actions. This results in a huge drop in the variety of actions.

Policy Gradient (PG) can be used in continuous action space, but because its training process requires action probability, it must output a probability distribution and then determine the action by sampling. This approach makes the output actions random and reduces the performance of the reinforcement learning network.

DDPG absorbs the benefits of PG's support for continuous motion. At the same time, an evaluation network is designed, which makes the action probability no longer needed in the training process and solves the problem of performance degradation caused by randomness.

Third, how to avoid the introduction of new parameters that require manual configuration and reduce the difficulty of algorithm configuration is also a question worthy of consideration.

Proposed Algorithm

In this section, an efficient parameter adaptation method is proposed. A variant of the particle swarm algorithm based on the above algorithm will be introduced later.

Parameter adaptation based on deep deterministic policy gradient (DDPG)

In this section, we first introduce the input and output of the actor network applied to the particle swarm algorithm. Next, we describe how the reward function scores based on changes in state. Then, the approach used to train the particle swarm algorithm is introduced. The process to run it with the trained network will be described later. Finally, the proposed new particle swarm algorithm is introduced.

State of particle swarm optimization (PSO, input of actor network)

The running state designed in this paper is divided into three parts: the current iteration progress, the current particle diversity, and the current duration of the particle no longer growing. In order for the neural network to work optimally, all operating states will eventually be mapped to the interval of $[-1, 1]$ and finally input to the neural network.

The iteration input is considered as a percentage of the iterations during the execution of the PSO, and the values for this input are in the range of 0 to 1. At the start of the algorithm, the iteration is considered to be at 0%, and the percentage is increased until it reaches 100% at the end of the execution. The values are obtained as follows:

$$Iteration = Fe_num / Fe_max \quad (7)$$

where Fe_num is the number of function evaluations that have been performed, and Fe_max is the maximum number of function evaluation executions set when the algorithm is run.

The diversity input is defined by the following equation, in which the degree of the dispersion of the swarm is calculated:

$$Diversity(t) = \frac{1}{D} \sum_{i=1}^D \sqrt{\sum_{j=1}^N [x_{ij}(t) - \bar{x}_j(t)]} \quad (8)$$

where $x_{ij}(t)$ is the j^{th} position of the i^{th} particle in iteration t , $\bar{x}_j(t)$ is the j^{th} average position of all the particles

in iteration t . With lower diversity, the particles are closer together, and with higher diversity, the particles are more separated. The diversity equation can be taken as the average of the Euclidean distances between each particle and the best particle.

The stagnant growth duration input is used to indicate whether the current particle swarm is running efficiently. It is defined as follows:

$$Iteration_{no-improvement} = (Fe_num - Fe_num_{last-improve}) / Fe_max \quad (9)$$

where $Fe_num_{last-improve}$ represents the number of evaluations at the last global optimal update.

Since the state distribution space is very different when optimizing different objective functions, unified normalization will cause some states to be too large or too small. To solve this problem and make it easier for the network to obtain information from the state, the state needs to be transformed. We believe that the transformation needs to meet the following requirements:

1. Make small changes significant.
2. The range needs to be mapped to the range of -1 to 1 .
3. Information cannot be lost.

Based on the above requirements and inspired by previous work [49], we designed the following transformations.

The input is originally x , which can represent the *Iteration*, *Diversity*, or *Iteration_{no-improvement}*. The output is as follows:

$$state_i = \sin(x * 2^i) \quad (10)$$

where $state_i$ is the i^{th} parameter value newly generated from x . In this paper, i takes values of 0, 1, 2, 3, and 4. An x will eventually generate five new parameters.

For example, if *Iteration* = x_1 , *Diversity* = x_2 and *Iteration_{no-improvement}* = x_3 , the new generated parameters will be as follows:

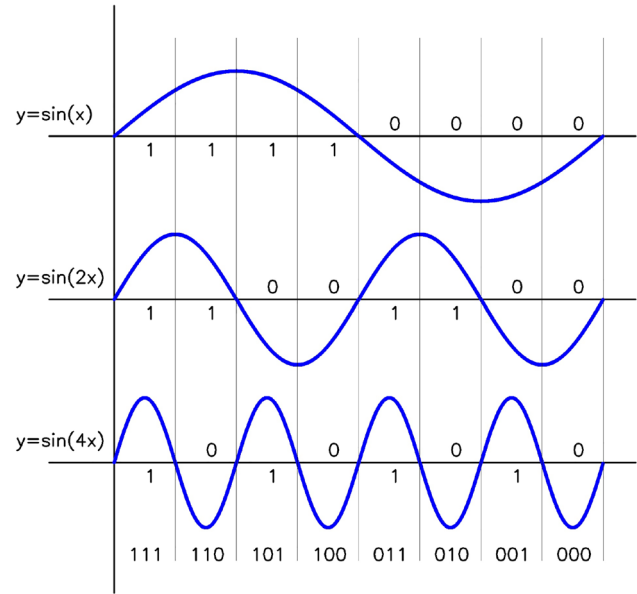


Fig. 3 Example of transform function

$$\begin{aligned} & \sin(x_1 * 2^0), \sin(x_1 * 2^1), \sin(x_1 * 2^2), \\ & \sin(x_1 * 2^3), \sin(x_1 * 2^4), \\ & \sin(x_2 * 2^0), \sin(x_2 * 2^1), \sin(x_2 * 2^2), \\ & \sin(x_2 * 2^3), \sin(x_3 * 2^4), \\ & \sin(x_3 * 2^0), \sin(x_3 * 2^1), \sin(x_3 * 2^2), \\ & \sin(x_3 * 2^3), \sin(x_3 * 2^4), \end{aligned}$$

The reason for this operation is that some of the above parameters are very small, and their changes are even smaller, which will cause the actor network to fail to capture their changes and fail to perform effective action output.

Part of the function image is shown in Fig. 3. The case where y is greater than 0 is regarded as 1, and the case where y is less than 0 is regarded as zero. These numbers are combined as shown in the lower part of the figure. These values are equivalent to the range of 7 to 0 in binary form. That is, the above transformation is similar to binarizing the original value. However, using binary values would waste computer memory when using floats. Instead, we use their float continuous counterparts—sinusoidal functions. In this way, the above requirements are met.

Action of PSO (output of actor network)

The actions designed in this paper are used to control the operating parameters in the PSO, such as w , c_1 , and c_2 . The parameters to be controlled can be set as required. The action output by the actor network does not control the above parameters of each particle separately but divides the particles into

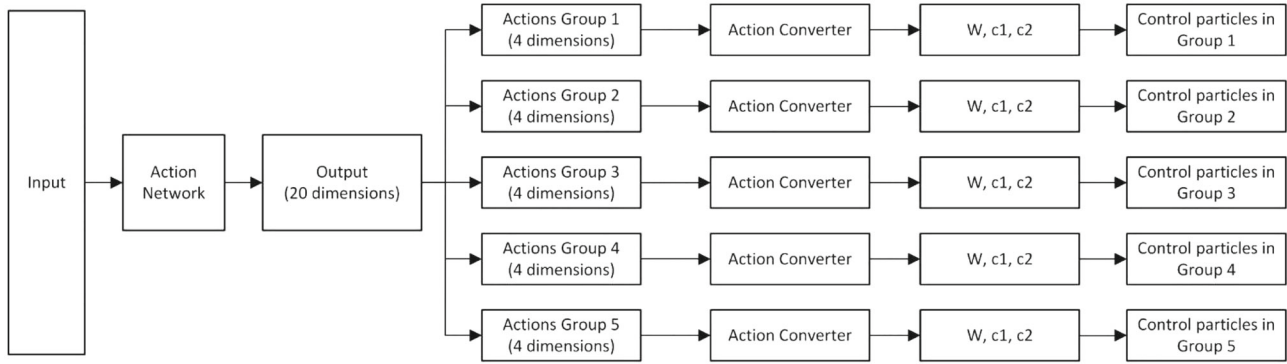


Fig. 4 Example of output of actor network

five subgroups, and it generates five sets of parameters to control the different groups.

We determine the corresponding subgroup based on the index of the particle. For example, if the index of the particle is 7, then the corresponding subgroup will be $7\%5 = 2$, i.e., subgroup 2. The indices of the subgroups are 0, 1, 2, 3, and 4. This method ensures that the numbers of individuals in every subgroup are similar.

For the traditional PSO algorithm, the obtained action vector a_t is 20-dimensional, divided into five groups, each of which is aimed at a subgroup. For a subgroup, the action vector is four-dimensional: $a[0]$ to $a[3]$. The w , $c1$, and $c2$ required for each round of the optimization algorithm are generated based on $a[0]$ to $a[3]$ as follows:

$$\begin{aligned}
 w &= a[0] * 0.8 + 0.1 \\
 scale &= 1/(a[1] + a[2] + 0.00001) * a[3] * 8 \\
 c1 &= scale * a[1] \\
 c2 &= scale * a[2]
 \end{aligned}
 \tag{11}$$

where $scale$ is a parameter that helps normalize $c1$ and $c2$, which is optional.

An example can be seen in Fig. 4. In this figure, the output represents action vector a_t , the action converter represents Eq. 11, and Action Group i (i can be 1 2 3 4 5) represents $a[0]$ to $a[3]$.

For some PSO variants, since the parameters were studied when developing the algorithm, the original parameter set is already quite good. To take advantage of the performance of the original parameter set, the new w , $c1$, $c2$ are configured according to the following formulas:

$$\begin{aligned}
 w &= a[0] * 0.5 + w_{origin} \\
 c1 &= a[1] * 0.5 + c1_{origin} \\
 c2 &= a[2] * 0.5 + c2_{origin}
 \end{aligned}
 \tag{12}$$

where w_{origin} , $c1_{origin}$, and $c2_{origin}$ represent the original parameters of the algorithm. In all subsequent experiments,

each algorithm will choose one of the action parameter configuration methods for experimentation. If there are more parameters in the algorithm that need to be configured, the number of output parameters can be increased and configured like $c1$ or w .

Reward function

A reward function is used to calculate the reward after an action is executed. Its target is to encourage the PSO to obtain a better $gBest$. Therefore, the reward function is designed as follows:

$$r(t) = \begin{cases} 1, & gBest(t+1) < gBest(t) \\ -1, & gBest(t+1) = gBest(t) \end{cases}
 \tag{13}$$

where $gBest(t)$ is the best solution in the t^{th} iteration.

Training

This section introduces the training process for the traditional PSO algorithm. Other particle swarm variants use basically the same process.

In the training process, each iteration of the particle swarm algorithm is equivalent to the agent performing an action in the environment, that is, an epoch of training. The particle swarm completes the optimization of an objective function, which is equivalent to the interaction between the agent and the environment, that is, an episode of training.

The training process is as follows. First, the actor network and the action-value network are randomly initialized, and the target actor network and the target action-value network are copied. A replay buffer R is then initialized in the second step to save the running state, actions, and rewards. The third step initializes the environment, including the initialization of the PSO and the initialization of the objective function to be optimized. The fourth step is to obtain the running state parameters from the particle swarm. The fifth step is to input

the running state s_t into the actor network, obtain the action a_t , and add a certain amount of random noise (the noise is used to help the network explore the policy space and will be removed after training). The random noise is normally distributed, with 0 as the mean and 0.5 as the variance. The formula is as follows:

$$a_t = Actor(s_t) + \mathcal{N}(0, 0.5). \quad (14)$$

The sixth step converts the resulting actions into the required w , $c1$, and $c2$ using Eq. 11. The seventh step is to perform an iteration of the PSO using the above parameters to obtain a new reward r_{t+1} and a new state s_{t+1} . The eighth step saves the state, actions, and rewards to buffer R. The ninth step is to randomly select a batch of experiences from buffer R. The weights of the action-value network are updated by minimizing the loss function (Eq. 3). The weights of the actor network are updated (Eq. 2). The tenth step is to update the weights of the target action-value network and the target actor network according to Eq. 3. If the particle swarm has not finished iterating yet, then t is increased by 1, and the process returns to step 4. If the training is not completed, the process returns to step 3. The pseudocode is presented in Algorithm 1.

If the objective function used in training directly adopts the function to be optimized later, the effect will be better. If training is performed with a set of test functions, it can work on all the objective functions, but not as well as the first method. After training, a trained action network model is obtained for subsequent runs.

Algorithm 1 Training algorithm.

- 1: Randomly initialize θ^Q and θ^μ in the actor network $\mu(s|\theta^\mu)$ and action-value network $Q(s, a|\theta^Q)$, respectively.
 - 2: Initialize the target networks Q' and μ' , and copy the weight values from Q and μ
 - 3: Initialize the playback buffer R
 - 4: **for** $episode = 1 : EpisodeMax$ **do**
 - 5: Initialization environment (PSO and evaluate function)
 - 6: **for** $t=1, Tmax$ **do**
 - 7: Obtain observation s_t from environment
 - 8: Choose actions based on s_t and network μ and explore noise {Eq. 14}
 - 9: Perform action a_t in the environment and observe reward r_t and new state s_{t+1}
 - 10: Save (s_t, a_t, r_t, s_{t+1}) to the buffer R
 - 11: Update action-value network by minimizing the loss function {Eq. 2}
 - 12: Update the actor network through the sampled action policy gradient {Eq. 3}
 - 13: Update the weights of the target network function {Eq. 1}
 - 14: **end for**
 - 15: **end for**
-

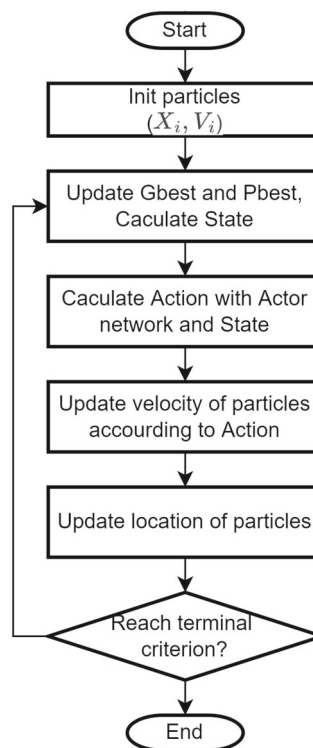


Fig. 5 Running process of particle swarm optimization (PSO) with reinforcement-learning-based parameter adaptation method (RLAM)

Running

In the running process, compared to the training process, many steps are removed, and the overall process is very simple. The operation process of the traditional particle swarm algorithm is introduced below. The implementation process of other particle swarm variant algorithms is basically the same as this process.

The flow chart of the PSO algorithm combined with RLAM is shown in Fig. 5. In addition to the original process of PSO, the new content is that before the particle velocity is updated, the running state of the particle swarm will be calculated, and a new parameter group will be generated to guide the particle update.

Network structure

This section introduces the network structures of the actor network and the action-value network in detail. Since the action-value network is only used in the pre-training process and the actor network needs to be used in the actual operation, to prevent excessive computation during the optimization process, a smaller actor network and a larger action-value network are designed. The schematic diagrams of the two networks are shown in Figs. 6 and 7.

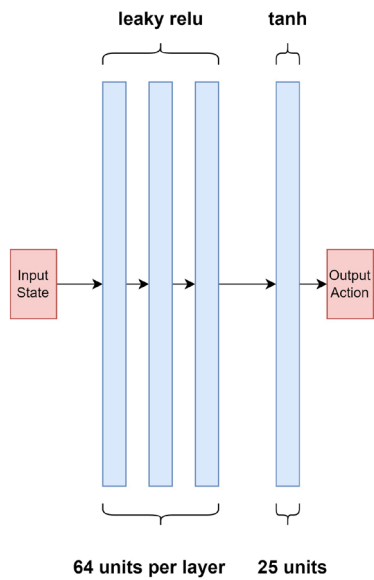


Fig. 6 Network structure of actor network

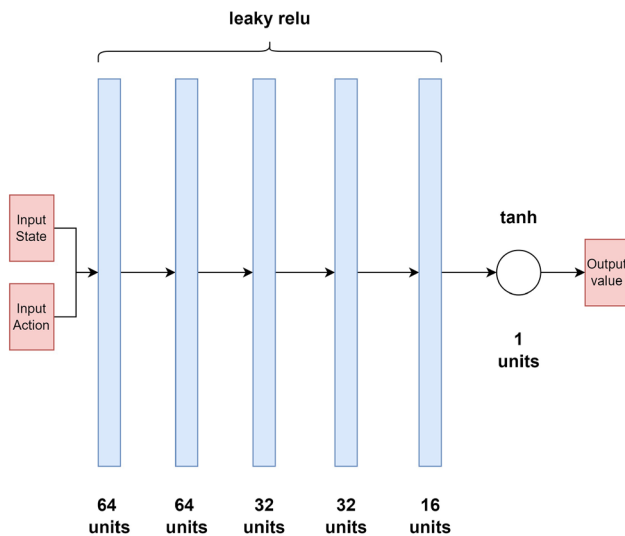


Fig. 7 Network structure of action-value network

The action-value network adopts a design with a six-layer, fully connected network, and the activation function between the networks is a Leaky ReLU. The actor network adopts a design with a four-layer fully connected network, and the activation function between the networks is also a Leaky ReLU. In the last layer, to map the action to the required range of -1 to 1 , a layer with the tanh activation function is added.

Reinforcement-learning-based PSO (RLPSO)

To better reflect the parameter adaptation ability of the RLAM, a new RLPSO algorithm based on the RLAM is

designed. The speed update equation of this algorithm is as follows:

$$v_i^d(t + 1) = w * v_i^d(t) + c1 * r1 * (pBest_{f_i(d)}^d - x_i^d(t)) + c2 * r2 * (gBest^d - x_i^d(t)) + c3 * r3 * (pBest_i^d - x_i^d(t)) \tag{15}$$

where $f_i(d)$ and the reason it is used in RLPSO were introduced in ‘‘Comprehensive learning particle swarm optimizer (CLPSO)’’, pBest is the particle’s own best experience, and gBest is the best experience in this swarm. w , $c1$, $c2$, $c3$, and $c_{mutation}$ are coefficients generated by the actor network based on the current running state. w , $c1$, $c2$, and $c3$ have been introduced in ‘‘Action of PSO (output of actor network)’’. $c_{mutation}$ is an additional parameter that is generated in the same way as w . $r1$, $r2$, and $r3$ are all uniformly distributed random numbers between 0 and 1.

To prevent particles from being trapped in local optima, there is a mutation stage after the velocity updating. During this stage, first, a random number $r4$ between 0 and 1 is generated, and then $r4$ is compared with $c_{mutation} * 0.01 * flag_{clps0}$. If $r4$ is less than it, the mutation is performed, and the particle position is reinitialized in the solution space.

At the end of one period, particles move according to their velocities, and then particles’ fitness and historical best experience are updated. The pseudocode is presented in Algorithm 2.

Algorithm 2 RLPSO algorithm.

- 1: Initialize the particle swarm and parameters
- 2: **while** computational budget is not exhausted **do**
- 3: Calculate s_t {Eq. 7, 8, 9, 10}
- 4: Calculate a_t with actor network μ (generated in ‘‘Training’’) and s_t
- 5: **for** $k = 1:n$ **do**
- 6: Convert actions into operating parameters (w , $c1$, $c2$, $c3$, and $c_{mutation}$) {Eq. 11, ‘‘Reinforcement-learning-based PSO (RLPSO)’’}
- 7: Calculate the new speed {Eq. 15}
- 8: **if** $randomvalue < c_{mutation} * 0.01 * flag_{clps0}$ **then**
- 9: Reinitialize position
- 10: **else**
- 11: Update position $X_i(t + 1) = X_i(t) + V_i(t)$
- 12: **end if**
- 13: Calculate the evaluation values of all particles
- 14: Update the parameters in particle operation
- 15: **end for**
- 16: **end while**

Experiments

To verify the performance of the proposed algorithm, three sets of experiments were conducted. In the first experiment, the RLAM was fused with various PSO variants, and the

Table 2 All test functions and their optimal values in CEC 2013

| No. | Function name | Best value |
|-----|--|------------|
| 1 | Sphere function | −1400 |
| 2 | Rotated high conditioned elliptic function | −1300 |
| 3 | Rotated bent cigar function | −1200 |
| 4 | Rotated discus function | −1100 |
| 5 | Different powers function | −1000 |
| 6 | Rotated rosenbrock's function | −900 |
| 7 | Rotated Schaffer's F7 function | −800 |
| 8 | Rotated Ackley's function | −700 |
| 9 | Rotated weierstrass function | −600 |
| 10 | Rotated Griewank's function | −500 |
| 11 | Rastrigin's function | −400 |
| 12 | Rotated Rastrigin's function | −300 |
| 13 | Non-continuous rotated Rastrigin's function | −200 |
| 14 | Schwefel's function | −100 |
| 15 | Rotated Schwefel's function | 100 |
| 16 | Rotated Katsuura function | 200 |
| 17 | Lunacek Bi_Rastrigin function | 300 |
| 18 | Rotated Lunacek Bi_Rastrigin function | 400 |
| 19 | Expanded Griewank's plus Rosenbrock's function | 500 |
| 20 | Expanded Scaffer's F6 function | 600 |
| 21 | Composition function 1 ($N = 5$, Rotated) | 700 |
| 22 | Composition function 2 ($N = 3$, Unrotated) | 800 |
| 23 | Composition function 3 ($N = 3$, Rotated) | 900 |
| 24 | Composition function 4 ($N = 3$, Rotated) | 1000 |
| 25 | Composition function 5 ($N = 3$, Rotated) | 1100 |
| 26 | Composition function 6 ($N = 5$, Rotated) | 1200 |
| 27 | Composition function 7 ($N = 5$, Rotated) | 1300 |
| 28 | Composition function 8 ($N = 5$, Rotated) | 1400 |

obtained results were compared with the original PSO variants. The second experiment compared the performance of the RLAM with other adaptation methods based on the classical particle swarm algorithm. Finally, the RLPSO designed based on the RLAM was compared with other current state-of-the-art algorithms in the third experiment.

In the experiments discussed below, to test the algorithm performance more comprehensively, we selected the CEC 2013 test set [50] for testing. The test set included 28 test functions that could simulate a wide variety of optimization problems. The test functions of CEC 2013 are shown in Table 2.

The domain of all the functions was -100 to 100 . Due to the relatively large overall testing volume, in all runs, the end condition for all algorithms was to complete 10,000 evaluations.

For all algorithms that use RLAM, if not specified, the test and training sets are the functions being tested. For example, if there are multiple functions to be tested, they will be

trained separately based on these functions and then tested separately.

The proposed algorithm was implemented using Python 3.9 on the 64-bit Ubuntu 16.04.7 LTS operating system. Experiments were conducted on a server with an Intel Xeon Silver 4116 2.1-GHz CPU and 128 GB of RAM. The source code used in experiments can be downloaded from this link: <https://github.com/Firesuiry/RLAM-OPENSOURCE>.

Improvement of PSO variants after combining with reinforcement-learning-based parameter adaptation method (RLAM)

In this experiment, we compared the performances of several PSO variants incorporating the RLAM with the original version.

- Comprehensive learning PSO (CLPSO) [17].
- Fitness-distance-ratio-based PSO (FDR-PSO) [51].

Table 3 Improvement of PSO variants after combining with RLAM

| Type | No. | CLPSO | FDRPSO | HPSO-TVAC | LIPS | PSO | SHPSO |
|-------------|-------|-------|--------|-----------|--------|--------|-------|
| Unimodal | 1 | 26.0% | 46.2% | 78.32% | 3.19% | 97.62% | 44.7% |
| | 2 | 37.6% | 38.8% | 14.26% | 28.35% | 35.37% | 26.8% |
| | 3 | 98.7% | 97.3% | 18.47% | 94.85% | 55.70% | 32.2% |
| | 4 | 7.8% | 22.8% | −1.01% | −1.09% | 7.06% | 5.5% |
| | 5 | 23.9% | 54.9% | 38.45% | 42.49% | 84.27% | 52.5% |
| Mean | | 38.8% | 52.0% | 29.7% | 33.6% | 56.0% | 32.4% |
| Multimodal | 6 | 37.4% | 49.6% | −8.33% | 27.94% | 22.82% | 29.3% |
| | 7 | 75.0% | 27.8% | 3.67% | 93.30% | 13.61% | 6.2% |
| | 8 | 0.3% | 0.2% | 0.17% | 0.12% | 0.22% | 0.1% |
| | 9 | 5.5% | 3.5% | −4.42% | 9.17% | 11.59% | 10.1% |
| | 10 | 15.5% | 34.0% | −4.43% | 15.59% | 41.57% | 52.2% |
| | 11 | 25.0% | 28.5% | 4.27% | 4.52% | 26.94% | 7.8% |
| | 12 | 21.7% | 20.8% | 12.58% | 2.73% | 32.11% | −6.4% |
| | 13 | 18.3% | 20.7% | 8.57% | 5.77% | 13.62% | 3.0% |
| | 14 | 6.5% | −3.9% | 24.55% | 8.28% | 32.44% | 4.9% |
| | 15 | 1.4% | 5.5% | 26.54% | 3.99% | 28.49% | 0.9% |
| | 16 | 6.0% | 8.3% | 7.06% | 7.90% | 19.88% | −8.0% |
| | 17 | 18.8% | 29.7% | 19.61% | −1.97% | 37.90% | 16.3% |
| | 18 | 26.1% | 43.2% | 15.95% | 0.92% | 19.59% | 11.2% |
| 19 | 52.9% | 87.4% | 31.36% | −24.77% | 95.46% | 45.7% | |
| 20 | 0.0% | 3.5% | 4.08% | 0.00% | 6.32% | 0.8% | |
| Mean | | 20.7% | 23.9% | 9.4% | 10.2% | 26.8% | 11.6% |
| Composition | 21 | 14.3% | 37.4% | 12.33% | 0.74% | 16.82% | 18.0% |
| | 22 | 5.5% | 5.0% | 17.34% | 7.54% | 24.48% | 13.5% |
| | 23 | 2.9% | 5.2% | 14.87% | 5.68% | 27.27% | 0.2% |
| | 24 | 1.5% | 1.2% | 1.60% | 19.45% | 3.26% | 1.0% |
| | 25 | 3.2% | 2.6% | 0.34% | 6.86% | 1.57% | −1.7% |
| | 26 | 5.5% | 6.7% | 15.05% | 17.34% | 24.05% | 23.4% |
| | 27 | 2.2% | 3.7% | 2.66% | 9.70% | 6.45% | 2.2% |
| | 28 | 10.1% | 20.7% | 9.99% | 3.99% | 30.77% | 13.6% |
| Mean | | 5.7% | 10.3% | 9.3% | 8.9% | 16.8% | 8.8% |
| All Mean | | 19.6% | 25.0% | 13.0% | 14.0% | 29.2% | 14.5% |

- Self-organizing hierarchical PSO with time-varying acceleration coefficients (HPSO-TVAC) [26].
- Distance-based locally informed PSO (LIPS) [52].
- Static heterogeneous swarm optimization (SHPSO) [53].
- Particle swarm optimization [54].

The combination of the PSO and the RLAM was as described above. For each problem, each algorithm was run 50 times, the test dimension was 30, and the final result is shown in Table 3.

In Table 3, the improvement was calculated as follows:

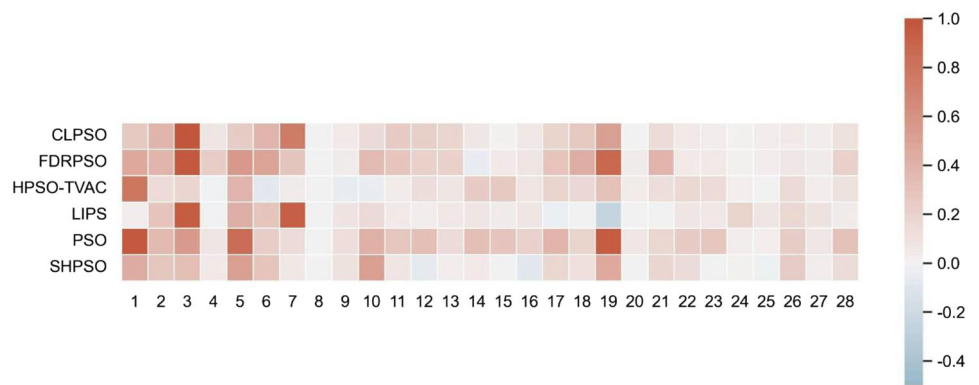
$$\begin{aligned}
 &improve \\
 &= (gBest_{origin} - gBest_{train}) / (gBest_{origin} \\
 &\quad - benchmark_{best}) \quad (16)
 \end{aligned}$$

where *improve* is the percentage of improvement, which is the data shown in the table, $gBest_{origin}$ is the optimal value obtained by the original version of the PSO variant, and $gBest_{train}$ is the optimal value obtained by the RLAM version of the PSO variant. Each row represents the result for a different evaluation function. $benchmark_{best}$ represents the optimal value of the current test function.

To show the improvement more intuitively, Fig. 8 shows the heatmap of the improvement of various PSO algorithms after adding the RLAM.

From Table 3 and Fig. 8, we can see that almost all the algorithms had significant improvements after incorporating the RLAM on all the test functions. The average improvement ranged from 13% to 29.2%, and the overall average improvement was 19.2%.

Fig. 8 Heatmap of improvement



For the unimodal functions (F01–F05), PSO combined with RLAM obtained a high average increase. For the multimodal functions (F06–F20), the average improvement was significantly lower than that of the unimodal functions. The improvement was the lowest among the composition functions (F21–F28). In general, the magnitude of the improvement was inversely related to the complexity of the function.

We also notice that the improvement varied quite a lot, even within the same class of test functions. To further study the influencing factors of the improvement, we tested several hypotheses and carried out experimental verification.

First, we believe that the original performance of the algorithm will affect the improvement. If the algorithm itself is already very good, then its improvement will be small. In contrast, if the algorithm itself is poor, it is more likely to see a larger improvement.

In addition, we believe that the sensitivity of the algorithm to the parameters will affect the improvement. If the algorithm is not sensitive to the parameters, no matter how you adjust it, there will be no change.

Finally, we believe that the probability of gBest updating at each iteration during the algorithm run affects the magnitude of the improvement. At each iteration in the training process, the reward is 1 if gBest is updated and -1 otherwise. If there are few rewards in the algorithm run, the network may not learn anything during the learning process.

We designed the following features to reflect the sensitivity of the algorithm to parameters and the update probability of gBest: For each algorithm and test function combination, we traversed all parameters that could be adjusted by training (such as w , $c1$, and $c2$), with 11 preset values from 0, 0.1 to 0.9, 1. For each preset parameter, the algorithm ran 10 times on the test function with fixed parameters to obtain the average optimal value. The standard deviation of these 11 average optimal values was calculated. If it was larger, the algorithm was more sensitive to parameters. In the above experimental process, the percentage of iterations with updated gBest in the run of each algorithm and test function combination to

the total number of iterations was recorded. For example, if the algorithm ran 10 steps in one run and gBest improved in three steps, then the probability of gBest improvement was 0.3.

Finally, we calculated the Spearman correlation coefficient between the optimal value obtained by the original algorithm, the parameter sensitivity of the algorithm and the update probability of gBest and the improvement with RLAM respectively. The results are shown in Table 6.

It can be seen that with the confidence level of $\alpha = 0.05$, the algorithm sensitivity, the average best update probability, and the original optimal value were positively correlated with the improvement.

To further investigate why the partial improvement was less than 0, we studied the parameters of the worst performing LIPS in function 19 as a function of the diversity, and the results are shown in the Fig. 9. The parameters of LIPS converged to the edge of the parameter definition domain as the diversity changed, indicating that the network had not learned useful knowledge, and the parameters of the PSO in function 1 changed significantly with the diversity. Based on this, we believe that the reason that the improvement was less than 0 was that the reinforcement learning failed to learn enough from experience in the past runs, which may have been caused by the performance limitations of the algorithm itself. In addition, the gBest update probability of LIPS on function 19 was 9%, and it ranked 119th in all 168 sets of data, which was lower than the overall average (32%). This also partially caused the reinforcement learning to fail to effectively learn from experience.

To provide a more comprehensive statistical analysis, non-parametric statistical tests were carried out. Table 4 presents the results of the pairwise comparison on a 30D problem, showing the number of cases in which the improvement was positive and negative as well as the p-values from the Wilcoxon test [43,55]. We normalized the results on every function to be in $[0, 1]$ based on the best and worst results obtained by all the algorithms [56]. As shown in the table, out of all six tested PSO methods, there were two algorithms

Table 4 Pairwise comparison between PSO and PSO with RLAM on 30D problems

| | CLPSO | FDRPSO | HPSO-TVAC | LIPS | PSO | SHPSO |
|-----------------|----------|----------|-----------|----------|----------|----------|
| Improve | 28 | 27 | 24 | 25 | 28 | 25 |
| Decrease | 0 | 1 | 4 | 3 | 0 | 3 |
| <i>P</i> -value | 3.79E−06 | 8.07E−06 | 1.19E−04 | 1.19E−04 | 3.79E−06 | 9.86E−05 |

Table 5 Pairwise comparison between PSO and PSO with RLAM on 100D problems

| | CLPSO | FDRPSO | HPSO-TVAC | LIPS | PSO | SHPSO |
|-----------------|----------|----------|-----------|----------|----------|----------|
| Improve | 25 | 26 | 24 | 19 | 19 | 23 |
| Decrease | 3 | 2 | 4 | 9 | 9 | 5 |
| <i>P</i> -value | 3.79E−06 | 8.80E−06 | 8.98E−06 | 2.69E−04 | 1.65E−03 | 3.05E−02 |

for which the performance improved for all test functions after being combined with the RLAM. The performance of the worst of the other algorithms decreased in four of the 28 test functions. Overall, of the 168 combinations of test algorithms and test functions, the performance decreased for a total of 11 cases, and it improved in the other cases. Thus, the proportion of cases for which the RLAM was effective was 93.5%. To further test the effectiveness of the algorithm, we also ran the above experiments in 100 dimensions, and the results are shown in Table 5. In general, according to the results of the Wilcoxon test, all PSO variants with RLAM were better than their origin versions, with a level of significance $\alpha = 0.05$.

To verify the generalization performance of RLAM, we used different training and test sets to test RLAM on the classic PSO algorithm. For different test functions, the functions in CEC2013 other than the function to be tested were used as the training set during training. For example, if the test set was function F1, then the training sets were F2, F3... F28. If the test set was function F2, then the training sets were F1, F3,...,F28. A total of 28 training processes were performed,

Table 6 Spearman correlation of optimal value, parameter sensitivity and gBest update probability with improvement

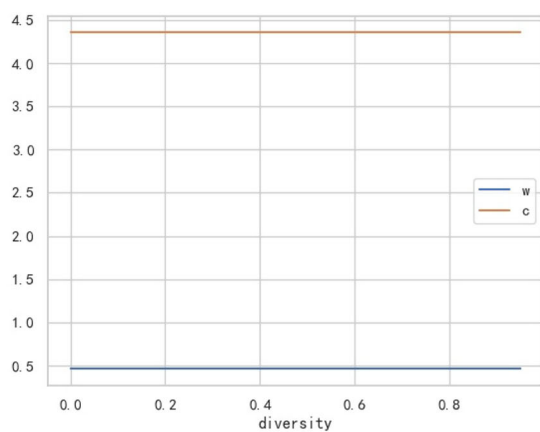
| | Correlation | <i>p</i> -value |
|-----------------------------|-------------|-----------------|
| Origin optimal value | 0.183 | 1.73E−2 |
| Update probability of gBest | 0.294 | 1.06E−4 |
| Parameter sensitivity | 0.465 | 2.07E−10 |

Table 7 Experimental results when the training and test sets were different

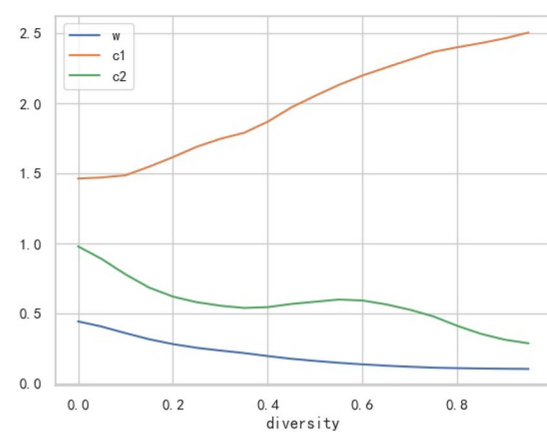
| Average improvement | Improve | Decrease | <i>p</i> -value |
|---------------------|---------|----------|-----------------|
| 22.3% | 27 | 1 | 4.23E−06 |

and tests were carried out on 28 CEC2013 test functions. This experiment was performed on 30D problems. The other conditions and calculations were the same as in the previous experiments.

As can be seen in Table 7, the average improvement rate was 22.3%, and 27 of the 28 test functions were improved.



(a) LIPS on F19



(b) PSO on F1

Fig. 9 Parameter changes with diversity

Table 8 Convergence accuracy comparison between RLAM and other adaptation methods

| No. | FT1PSO | FT2PSO | SuccessHistoryPSO | QLPSO | PSOtrain |
|-------|---------------------|--------------|--------------------|---------------------|------------------|
| 1 | 3.47E+02(+) | 6.92E+02(+) | 3.16E+02(+) | 2.78E+02(+) | -1.33E+03 |
| 2 | 3.38E+07(+) | 3.57E+07(+) | 5.10E+07(+) | 2.68E+07(=) | 2.05E+07 |
| 3 | 1.85E+10(+) | 2.56E+10(+) | 2.57E+10(+) | 2.05E+10(+) | 8.66E+09 |
| 4 | 6.76E+04(=) | 7.02E+04(=) | 6.92E+04(=) | 5.57E+04(=) | 5.92E+04 |
| 5 | -4.27E+02 | -1.70E+02(+) | 1.17E+02(+) | -8.42E+02(-) | -8.20E+02 |
| 6 | -6.67E+02 | -6.75E+02(+) | -6.23E+02(+) | -7.82E+02(=) | -8.16E+02 |
| 7 | -6.77E+02(=) | -6.50E+02(=) | -6.40E+02(+) | -6.76E+02(=) | -6.70E+02 |
| 8 | -6.79E+02(=) | -6.79E+02(+) | -6.79E+02(=) | -6.79E+02(+) | -6.79E+02 |
| 9 | -5.72E+02(+) | -5.72E+02(+) | -5.71E+02(+) | -5.71E+02(+) | -5.75E+02 |
| 10 | -2.31E+01(+) | -2.11E+01(+) | 4.84E+01(+) | -2.18E+02(+) | -3.22E+02 |
| 11 | -2.23E+02(+) | -2.13E+02(+) | -2.21E+02(+) | -2.82E+02(+) | -2.83E+02 |
| 12 | -1.01E+02(+) | -8.81E+01(+) | -4.00E+01(+) | -1.30E+02(=) | -1.40E+02 |
| 13 | 1.03E+02(+) | 1.09E+02(+) | 1.21E+02(+) | 3.37E+01(+) | 1.71E+01 |
| 14 | 3.44E+03(+) | 3.56E+03(+) | 3.26E+03(+) | 5.34E+03(+) | 2.79E+03 |
| 15 | 5.48E+03(+) | 4.84E+03(=) | 4.93E+03(=) | 7.11E+03(+) | 4.26E+03 |
| 16 | 2.03E+02(=) | 2.03E+02(=) | 2.03E+02(=) | 2.03E+02(+) | 2.03E+02 |
| 17 | 4.96E+02(+) | 5.21E+02(+) | 5.41E+02(+) | 4.94E+02(+) | 4.61E+02 |
| 18 | 6.49E+02(=) | 6.75E+02(+) | 6.52E+02(=) | 6.47E+02(-) | 6.72E+02 |
| 19 | 2.23E+03(+) | 8.66E+02(+) | 1.25E+03(+) | 9.46E+02(+) | 5.15E+02 |
| 20 | 6.14E+02(+) | 6.14E+02(+) | 6.14E+02(+) | 6.14E+02(=) | 6.14E+02 |
| 21 | 1.74E+03(+) | 1.52E+03(+) | 1.96E+03(+) | 1.22E+03(+) | 9.96E+02 |
| 22 | 5.23E+03(+) | 5.12E+03(+) | 5.49E+03(+) | 5.63E+03(+) | 4.37E+03 |
| 23 | 6.21E+03(=) | 6.19E+03(=) | 5.80E+03(-) | 8.29E+03(=) | 6.30E+03 |
| 24 | 1.29E+03(+) | 1.28E+03(+) | 1.30E+03(+) | 1.28E+03(+) | 1.28E+03 |
| 25 | 1.41E+03(+) | 1.41E+03(+) | 1.41E+03(+) | 1.39E+03(=) | 1.40E+03 |
| 26 | 1.45E+03(=) | 1.54E+03(=) | 1.52E+03(=) | 1.44E+03(=) | 1.44E+03 |
| 27 | 2.32E+03(+) | 2.34E+03(+) | 2.35E+03(+) | 2.38E+03(+) | 2.28E+03 |
| 28 | 3.52E+03(+) | 3.51E+03(+) | 3.53E+03(+) | 2.93E+03(+) | 2.16E+03 |
| +/-/= | 21/0/7 | 22/0/6 | 21/1/6 | 17/2/9 | |

The bold data indicates the optimal result

The effect was slightly worse than that for the previous data, but with the confidence level of significance $\alpha = 0.05$, the results for the classic PSO with RLAM using different training and test sets was better than those of the corresponding original versions. The original PSO algorithm was improved, which showed that RLAM could still achieve significant results when the test and training sets were different.

Thus, we can conclude that the effect of RLAM was highly significant.

Comparison with other adaptation methods

To test the pros and cons of the RLAM with other adaptation methods, the RLAM was compared with four other adaptation methods. The other four algorithms were as follows:

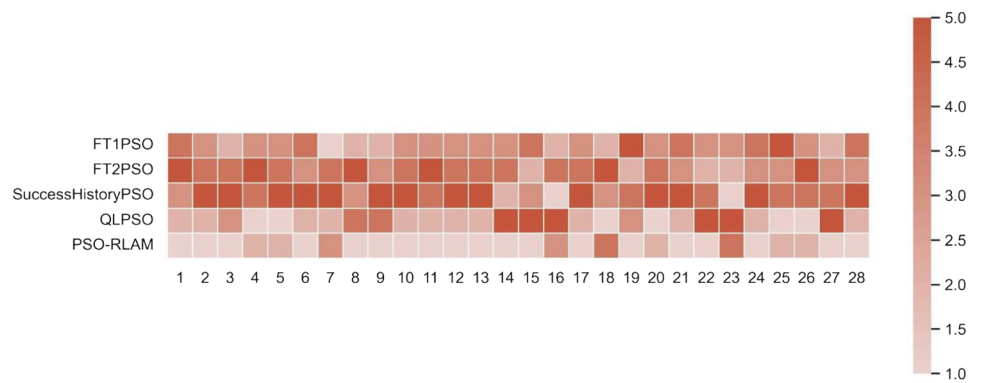
- an adaptation algorithm based on type-1 fuzzy logic (TF1PSO) [57].

- an adaptation algorithm based on type-2 fuzzy logic (TF2PSO) [41].
- an adaptation algorithm based on the success rate history (SuccessHistoryPSO) [58].
- an adaptation algorithm based on Q-learning (QLPSO) [11].

In the result, the PSO with the RLAM was labeled as PSOtrain. Wilcoxon's signed rank test (+, -, = in Table 8) at the 0.05 significance level was employed to compare the performances of the algorithms.

The best results obtained by the PSO with the RLAM and the PSO with other adaptation methods are listed in Table 8, and the best results are shown in bold. Some of the results were the same for several algorithms. Only one of the cases marked in bold was due to insufficient numerical resolution, and finer results would show differences between these algorithms. The fuzzy-logic-based adaptation performed the

Fig. 10 Comparison between RLAM and other adaptation methods



worst. The type-1-fuzzy-logic-based adaptation was the best for F7, whereas the type-2-fuzzy-logic-based adaptation was not the best in any cases. The adaptation based on successful history was the best for F16 and F23, while the adaptation based on reinforcement learning performed the best in the test. Of these, the adaptation based on Q-learning was the best for F4, F5, F18, F20, F25, and F26, and the RLAM was the best for a total of 19 other functions.

Figure 11 describes the ranking of these algorithms for multiple comparisons. From Fig. 10, we can see that the RLAM had the lightest overall color and the highest average ranking on the entire function test set. The adaptation based on Q-learning also worked well, but it performed the worst for six test functions: F14, F15, F16, F22, F23, and F27. The effects of the other algorithms were not much different. Of these, the adaptation algorithm based on type-1 fuzzy logic ranked better overall, with almost no worst results, indicating that its robustness was better.

Table 9,10 shows the pairwise comparisons between the PSOs with other adaptation methods on 30D and 100D problems. According to the Sign test in [55], it is clear that the PSO with the RLAM was significantly better than the FT1PSO, FT2PSO, success-history-PSO, and QLPSO with a level of significance $\alpha = 0.05$. Overall, the RLAM exhibited excellent performances on all the test functions compared to other adaptation methods.

Comparison of RLPSO with other state-of-the-art PSO variants

Since most of the current state-of-the-art particle swarm algorithms combine many other optimization methods, adaptation parameters alone cannot provide a sufficient comparison. To test the strength of the RLAM applied in particle swarm optimization, here we compare the performance of the RLPSO based on the RLAM and some other particle swarm optimization algorithms. The algorithms compared include some of the widely used particle swarm algorithms: CLPSO [17], FDR-PSO [51], LIPS [52], PSO [9], and SHPSO [53], as well as several state-of-the-art variants of the PSO, including the EPSO [37], AWPSO [59], and PPPSO [60]. All the algorithms were executed 50 times for each problem and the results shown are the averages. Wilcoxon’s signed rank test (+, −, = in Table 11) at the 0.05 significance level was employed to compare the performances of the algorithms.

The best results obtained by the RLPSO and other PSO variants are listed in Table 11, and the best results are also shown in bold. The RLPSO was the best in 18 of the 28 test functions, far exceeding several other PSO algorithms. The SHPSO was the best for two functions, the AWPSO was the best for five functions, the PPPSO was the best for two functions, and the EPSO was the best for one function.

Table 9 Pairwise comparison between PSO with RLAM and other adaptation methods on 30D problems

| PSO-TRAIN VS. | FT1PSO | FT2PSO | SuccessHistoryPso | QLPSO |
|-----------------|-------------|-------------|-------------------|-------------|
| WIN | 24 | 27 | 25 | 21 |
| LOSS | 4 | 1 | 3 | 7 |
| <i>p</i> -value | 6.74804E−05 | 4.21632E−06 | 1.45882E−05 | 0.001319008 |

Table 10 Pairwise comparison between PSO with RLAM and other adaptation methods on 100D problems

| PSO-TRAIN VS. | FT1PSO | FT2PSO | SuccessHistoryPso | QLPSO |
|-----------------|-------------|------------|-------------------|------------|
| WIN | 27 | 26 | 24 | 26 |
| LOSS | 1 | 2 | 4 | 2 |
| <i>p</i> -value | 4.71565E−06 | 1.2305E−05 | 2.79026E−05 | 9.9777E−06 |

Figure 11 shows the specific ranking of each algorithm on different test problems. The RLPSO ranked very high on almost all problems, reflecting the excellent performance of the RLPSO. To show the stability of the proposed algorithm, convergence curves are shown in Fig. 12. Tables 12 and 13 show pairwise comparisons of the RLPSO results. According to the Sign test in [55], it is clear that RLPSO was significantly better than other PSOs with a level of significance of $\alpha = 0.05$.

In general, the RLPSO exhibited outstanding performance compared to many other particle swarm algorithms, which showed that the RLAM method can provide an excellent particle swarm variant algorithm after a certain amount of design, and the results highlighted the power of the RLAM.

Conclusions

In this paper, a reinforcement-learning-based parameter adaptation method (RLAM) and an RLAM-based RLPSO are proposed. In the RLAM, through each generation, the optimal coefficients are generated by the actor network, where the actor network is trained before running. It can be trained using the target function or using test functions. A combination of iterations, non-improving iterations, and diversity is used to reflect the state. The reward is calculated based on the change of the best result after an update. In the RLPSO, in addition to the RLAM, the CLPSO and mutation mechanisms are also added to the algorithm.

Furthermore, comprehensive experiments were carried out to compare the new algorithm with other adaptation methods, investigate the effects of the RLAM with different PSO

variants, and compare the RLPSO with other state-of-the-art PSO variants. The proposed method was incorporated into multiple PSO variants and tested on the CEC 2013 test set, and almost all the PSO variants had improved final optimization accuracies. This result showed that the RLAM was beneficial and harmless in almost all the problems and all the optimization algorithms considered. It solves the problem that manual parameter adjustment is too cumbersome and burdensome.

The algorithm proposed in this paper was also compared with other adaptation methods, including adaptive particle swarm optimization based on Q-learning, adaptive particle swarm optimization based on fuzzy logic, and adaptive particle swarm optimization based on the success rate history. Based on the final results, the adaptive algorithm proposed in this paper significantly outperformed other adaptive algorithms.

Since most of the current state-of-the-art particle swarm algorithms combine many other methods, adaptation parameters alone cannot make a good comparison. Therefore, the RLPSO based on the RLAM was designed, which combined the CLPSO and some mutation and grouping operations. The final algorithm was compared with a variety of top particle swarm algorithms on the CEC 2013 test set, and the proposed algorithm was at the leading level.

In the future, based on the RLAM, the selection of states, the control of actions, and the applicability of other optimization algorithms will be further studied to further improve the optimization performance. We will focus on how to use this algorithm on binary optimization problems.

Table 11 Comparison between RLPSO and other PSO variants

| No. | CLPSO | FDRPSO | LIPS | PSO | SHPSO | AWPSO | PPPSO | EPSO | RLPSO |
|-------|--------------|--------------|--------------|--------------|--------------------|--------------------|--------------------|--------------|------------|
| 1 | 5.19E+04(+) | 2.75E+04(+) | 5.06E+04(+) | -8.12E+02(+) | -1.18E+03(+) | 7.39E+02(+) | -2.16E+02(+) | -1.37E+03(+) | -1.40E+03 |
| 2 | 8.98E+08(+) | 3.46E+08(+) | 1.49E+09(+) | 3.10E+07(+) | 6.42E+07(+) | 2.59E+07(+) | 3.33E+07(+) | 3.57E+07(+) | 1.72E+07 |
| 3 | 9.09E+16(+) | 2.75E+12(+) | 7.56E+18(+) | 1.47E+10(+) | 4.47E+10(+) | 9.83E+09(=) | 3.05E+10(+) | 1.34E+10(+) | 1.14E+10 |
| 4 | 1.03E+05(+) | 1.21E+05(+) | 6.71E+04(+) | 6.06E+04(=) | 5.88E+04(=) | 6.18E+04(=) | 6.18E+04(=) | 5.74E+04(=) | 5.71E+04 |
| 5 | 2.03E+04(+) | 8.23E+03(+) | 3.01E+04(+) | -3.13E+02(+) | -7.76E+02(+) | -2.85E+02(+) | -2.64E+02(+) | -9.54E+02(=) | -9.55E+02 |
| 6 | 8.86E+03(+) | 9.39E+02(+) | 1.32E+04(+) | -8.02E+02(=) | -7.37E+02(+) | -7.88E+02(+) | -7.73E+02(+) | -7.59E+02(+) | -8.07E+02 |
| 7 | 6.78E+04(+) | -4.79E+02(+) | 4.53E+06(+) | -6.52E+02(+) | -6.03E+02(+) | -6.10E+02(+) | -6.29E+02(+) | -6.83E+02(=) | -6.79E+02 |
| 8 | -6.79E+02(+) | -6.79E+02(+) | -6.79E+02(+) | -6.79E+02(+) | -6.79E+02(=) | -6.79E+02(=) | -6.79E+02(+) | -6.79E+02(+) | -6.79E+02 |
| 9 | -5.58E+02(+) | -5.58E+02(+) | -5.53E+02(+) | -5.73E+02(+) | -5.62E+02(+) | -5.75E+02(=) | -5.73E+02(+) | -5.67E+02(+) | -5.75E+02 |
| 10 | 6.87E+03(+) | 2.81E+03(+) | 8.24E+03(+) | -3.58E+02(+) | -5.43E+01(+) | -1.11E+02(+) | -2.14E+02(+) | -4.00E+02(+) | -4.38E+02] |
| 11 | 5.24E+02(+) | 2.26E+02(+) | 4.78E+02(+) | -2.58E+02(+) | -2.59E+02(+) | -2.95E+02(=) | -2.60E+02(+) | -1.45E+02(+) | -2.88E+02 |
| 12 | 6.04E+02(+) | 2.72E+02(+) | 5.76E+02(+) | -8.19E+01(+) | -3.41E+01(+) | -7.22E+01(+) | -1.06E+02(+) | -9.19E+00(+) | -1.40E+02 |
| 13 | 7.30E+02(+) | 3.58E+02(+) | 6.81E+02(+) | 4.64E+01(=) | 1.16E+02(+) | 5.73E+01(=) | 2.58E+01(-) | 1.13E+02(+) | 5.62E+01 |
| 14 | 7.19E+03(+) | 7.45E+03(+) | 8.80E+03(+) | 4.19E+03(+) | 3.86E+03(+) | 4.47E+03(+) | 4.43E+03(+) | 7.10E+03(+) | 2.76E+03 |
| 15 | 8.03E+03(+) | 8.54E+03(+) | 8.48E+03(+) | 6.72E+03(+) | 6.21E+03(+) | 7.48E+03(+) | 7.27E+03(+) | 7.98E+03(+) | 5.00E+03 |
| 16 | 2.04E+02(+) | 2.04E+02(+) | 2.04E+02(+) | 2.03E+02(+) | 2.02E+02(-) | 2.04E+02(+) | 2.03E+02(+) | 2.04E+02(+) | 2.03E+02 |
| 17 | 1.68E+03(+) | 1.60E+03(+) | 1.22E+03(+) | 5.42E+02(+) | 5.94E+02(+) | 5.51E+02(+) | 5.41E+02(+) | 6.20E+02(+) | 4.78E+02 |
| 18 | 2.00E+03(+) | 1.88E+03(+) | 1.33E+03(+) | 7.01E+02(+) | 7.57E+02(+) | 7.04E+02(+) | 6.95E+02(+) | 7.33E+02(+) | 6.79E+02 |
| 19 | 1.54E+06(+) | 9.37E+04(+) | 5.14E+05(+) | 9.59E+02(+) | 5.88E+02(+) | 1.37E+03(+) | 5.45E+02(+) | 5.36E+02(+) | 5.23E+02 |
| 20 | 6.15E+02(+) | 6.15E+02(+) | 6.15E+02(+) | 6.14E+02(=) | 6.14E+02(+) | 6.14E+02(+) | 6.14E+02(=) | 6.15E+02(+) | 6.14E+02 |
| 21 | 4.58E+03(+) | 4.27E+03(+) | 3.25E+03(+) | 1.10E+03(+) | 1.38E+03(+) | 1.21E+03(+) | 1.19E+03(+) | 1.11E+03(+) | 9.90E+02 |
| 22 | 9.00E+03(+) | 9.75E+03(+) | 1.04E+04(+) | 5.38E+03(+) | 5.42E+03(+) | 5.14E+03(+) | 4.92E+03(+) | 8.46E+03(+) | 4.07E+03 |
| 23 | 9.48E+03(+) | 9.48E+03(+) | 1.02E+04(+) | 7.95E+03(+) | 7.28E+03(=) | 8.56E+03(+) | 8.18E+03(+) | 9.46E+03(+) | 7.64E+03 |
| 24 | 1.37E+03(+) | 1.30E+03(+) | 1.54E+03(+) | 1.28E+03(=) | 1.30E+03(+) | 1.28E+03(=) | 1.28E+03(+) | 1.30E+03(+) | 1.28E+03 |
| 25 | 1.49E+03(+) | 1.41E+03(+) | 1.54E+03(+) | 1.40E+03(=) | 1.40E+03(=) | 1.39E+03(-) | 1.39E+03(-) | 1.42E+03(+) | 1.40E+03 |
| 26 | 1.51E+03(+) | 1.53E+03(+) | 1.64E+03(+) | 1.52E+03(+) | 1.49E+03(+) | 1.53E+03(+) | 1.56E+03(+) | 1.51E+03(+) | 1.44E+03 |
| 27 | 2.86E+03(+) | 2.62E+03(+) | 3.16E+03(+) | 2.31E+03(+) | 2.51E+03(+) | 2.21E+03(-) | 2.28E+03(=) | 2.45E+03(+) | 2.27E+03 |
| 28 | 7.79E+03(+) | 5.56E+03(+) | 7.89E+03(+) | 2.61E+03(+) | 3.04E+03(+) | 2.94E+03(+) | 2.42E+03(+) | 3.68E+03(+) | 2.20E+03 |
| +/-/= | 28/0/0 | 28/0/0 | 28/0/0 | 22/0/6 | 23/1/4 | 19/3/6 | 23/2/3 | 25/0/3 | |

The bold data indicates the optimal result

Fig. 11 Comparison between RLPSO and other PSO algorithms

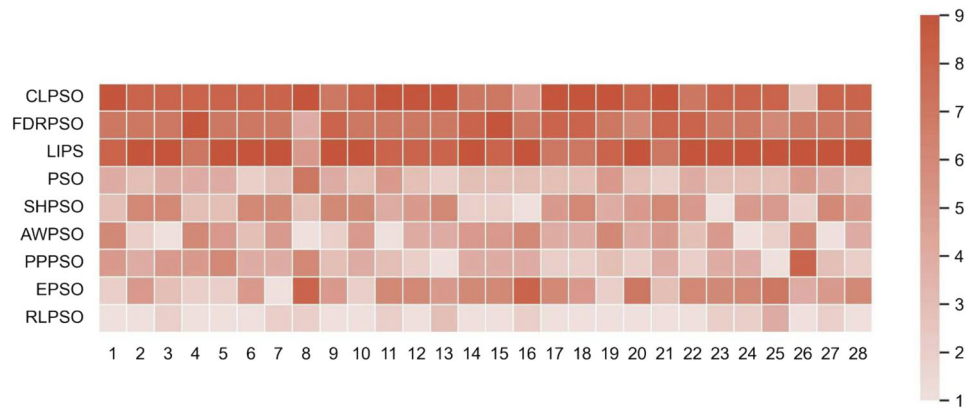


Fig. 12 Convergence curves between RLPSO and other PSO variants

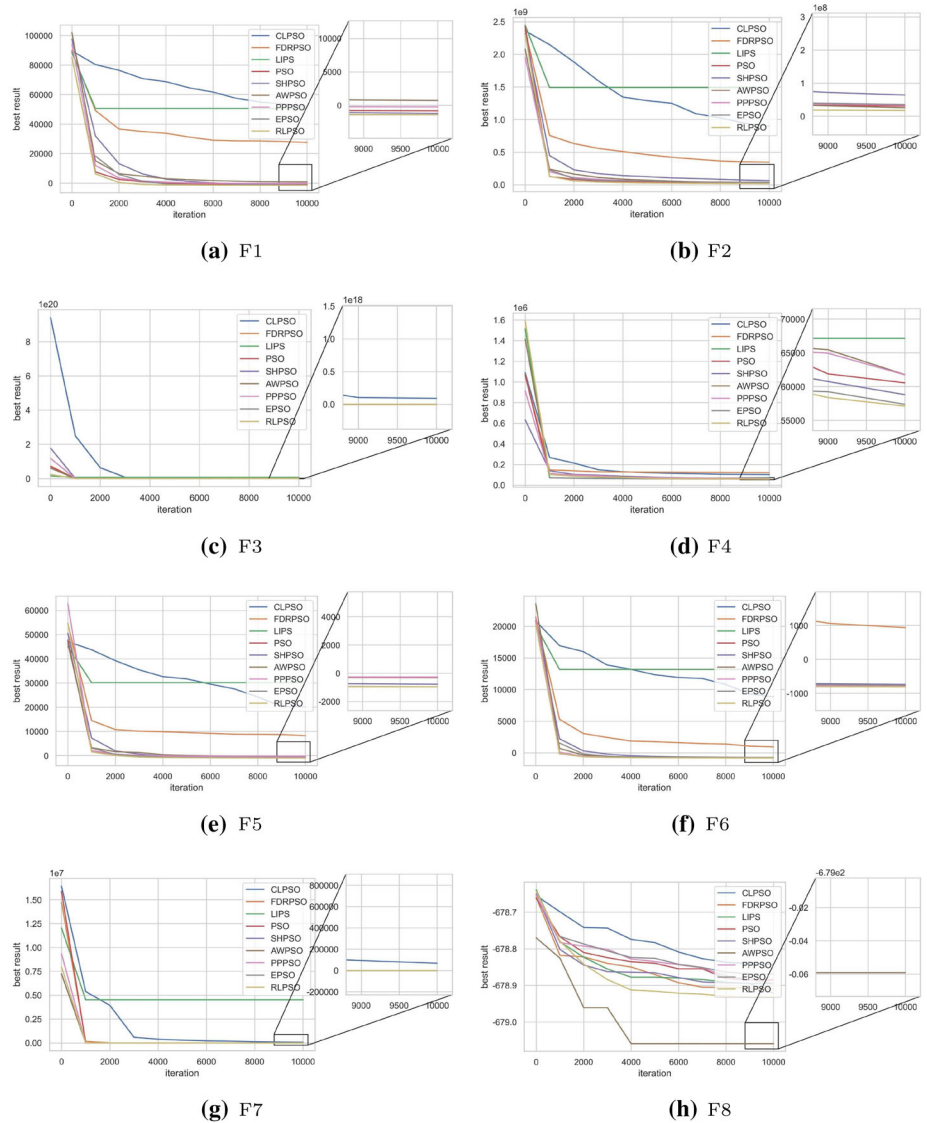


Fig. 12 continued

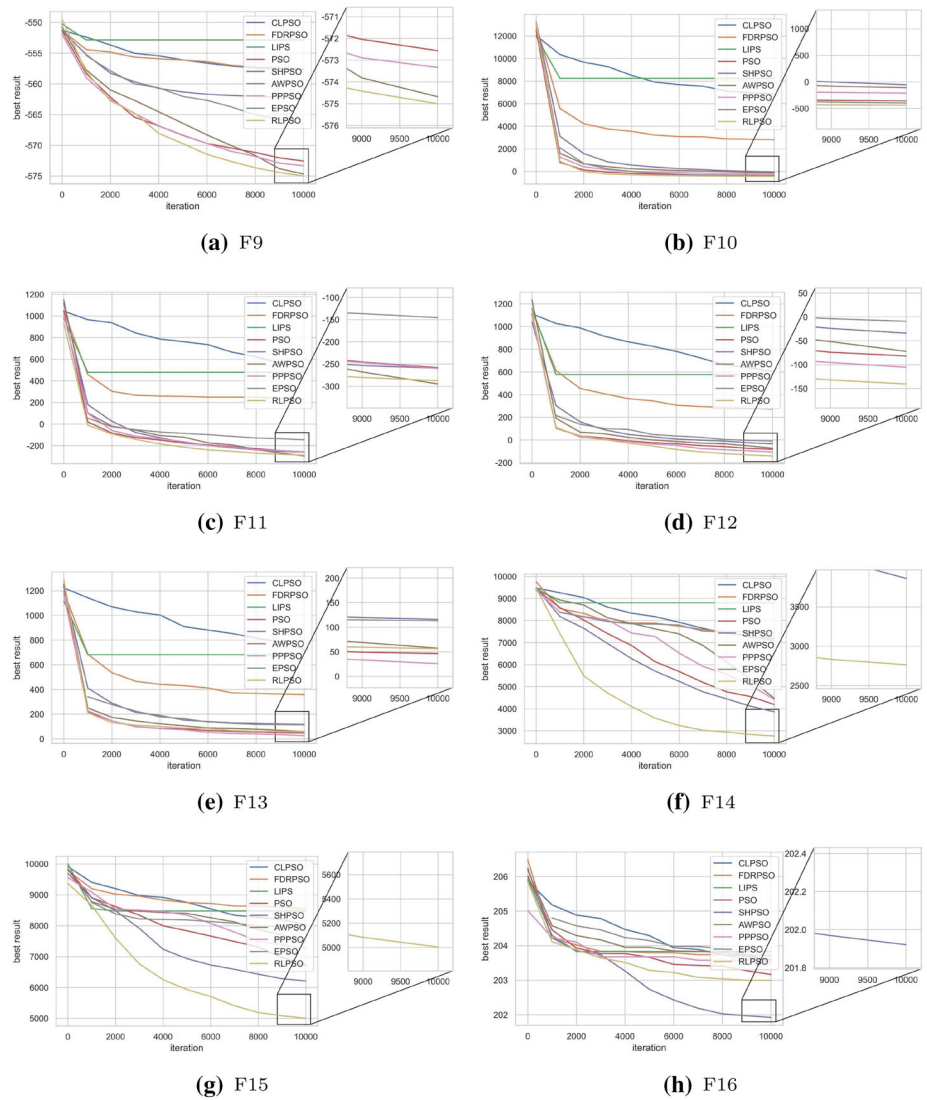


Fig. 12 continued

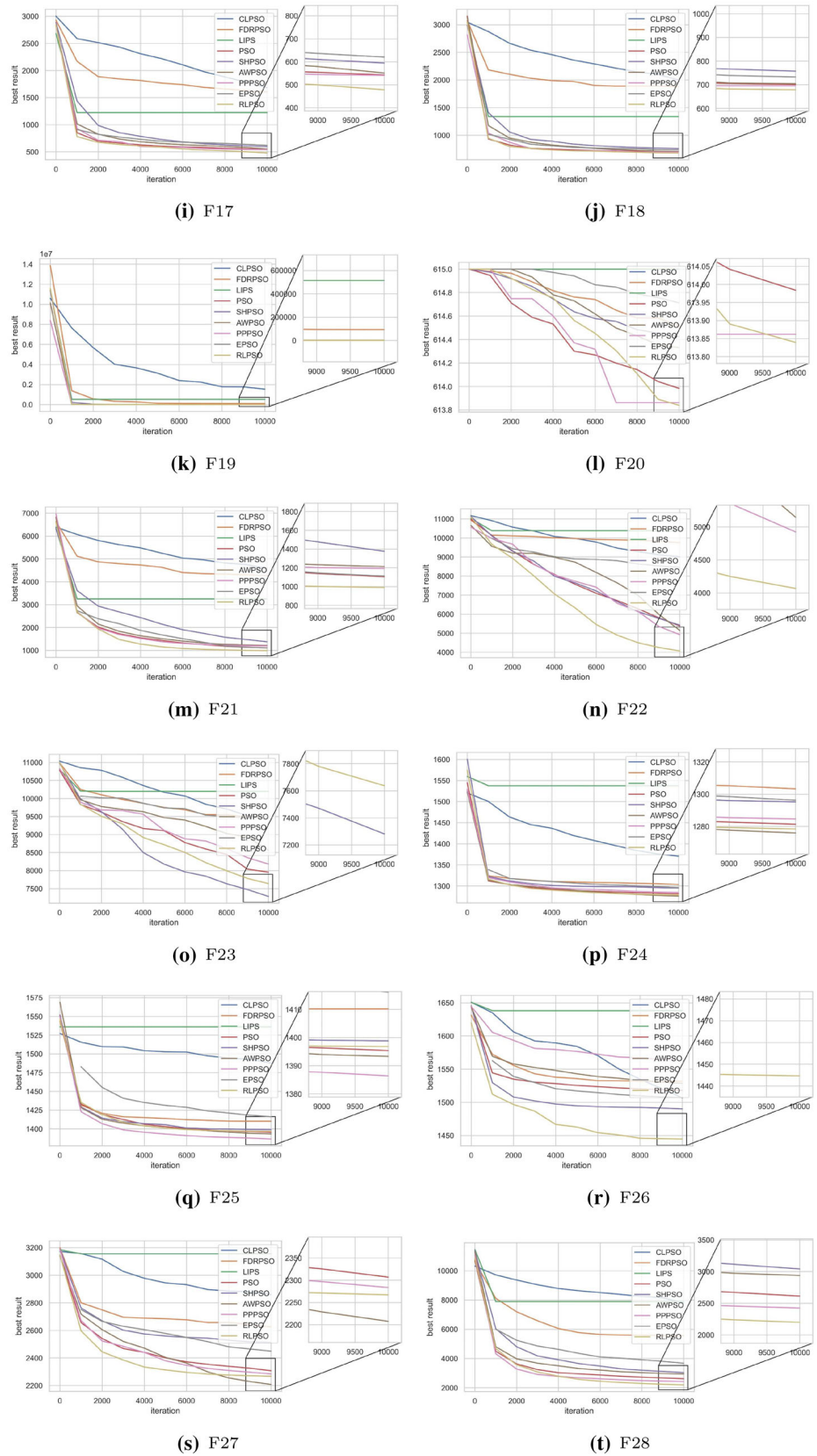


Table 12 Pairwise comparison between RLPSO and state-of-the-art PSO algorithms on 30D problems

| RLPSO VS | CLPSO | FDRPSO | LIPS | PSO | SHPSO | AWPSO | PPPSO | EPSO |
|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| WIN | 28 | 28 | 28 | 26 | 26 | 22 | 26 | 27 |
| LOSS | 0 | 0 | 0 | 2 | 2 | 6 | 2 | 1 |
| <i>p</i> -value | 3.69479E-06 | 3.78688E-06 | 3.35953E-06 | 2.28059E-05 | 0.000320816 | 0.002853853 | 0.000118906 | 4.71565E-06 |

Table 13 Pairwise comparison between RLPSO and state-of-the-art PSO algorithms on 100D problems

| RLPSO VS | CLPSO | FDRPSO | LIPS | PSO | SHPSO | AWPSO | PPPSO | EPSO |
|-----------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| WIN | 27 | 26 | 26 | 21 | 22 | 23 | 22 | 21 |
| LOSS | 1 | 2 | 2 | 7 | 6 | 5 | 6 | 7 |
| <i>p</i> -value | 3.73519E-06 | 5.85220E-06 | 5.84000E-06 | 1.81045E-03 | 1.78735E-02 | 1.32380E-03 | 2.11109E-03 | 2.69031E-04 |

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there are no conflicts of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Del Ser J, Osaba E, Molina D, Yang X-S, Salcedo-Sanz S, Camacho D, Das S, Suganthan PN, Coello Coello CA, Herrera F (2019) Bio-inspired computation: Where we stand and what's next. *Swarm Evol Comput* 48:220–250. <https://doi.org/10.1016/j.swevo.2019.04.008>
- Xu Y, Pi D (2019) A hybrid enhanced bat algorithm for the generalized redundancy allocation problem. *Swarm Evol Comput* 50:100562. <https://doi.org/10.1016/j.swevo.2019.100562>
- Zhu Z, Zhou J, Ji Z, Shi Y-H (2011) Dna sequence compression using adaptive particle swarm optimization-based memetic algorithm. *IEEE Trans Evol Comput* 15(5):643–658. <https://doi.org/10.1109/TEVC.2011.2160399>
- Arqub OA, Abo-Hammour Z (2014) Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. *Inform Sci* 279:396–415
- Abu Arqub O (2017) Adaptation of reproducing kernel algorithm for solving fuzzy fredholm-volterra integrodifferential equations. *Neural Comput Appl* 28(7):1591–1610
- Abu Arqub O, Maayah B (2018) Solutions of bagley-torvik and painlevé equations of fractional order using iterative reproducing kernel algorithm with error estimates. *Neural Comput Appl* 29(5):1465–1479
- Zhang H, Cao X, Ho JK, Chow TW (2016) Object-level video advertising: an optimization framework. *IEEE Trans Ind Inform* 13(2):520–531
- Chen L, Xu Y, Xu F, Hu Q, Tang Z (2022) Balancing the trade-off between cost and reliability for wireless sensor networks: a multi-objective optimized deployment method. *Appl Intell*. <https://doi.org/10.1007/s10489-022-03875-9>
- Kennedy J, Eberhart R (1995) Particle swarm optimization. In: *Proceedings of ICNN'95-international Conference on Neural Networks*, vol 4. IEEE, pp 942–1948
- Xu Y, Pi D (2020) A reinforcement learning-based communication topology in particle swarm optimization. *Neural Comput Appl* 32(14):10007–10032. <https://doi.org/10.1007/s00521-019-04527-9>
- Liu Y, Lu H, Cheng S, Shi Y (2019) An Adaptive Online Parameter Control Algorithm for Particle Swarm Optimization Based on Reinforcement Learning. In: *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, pp 815–822. <https://doi.org/10.1109/CEC.2019.8790035>
- Karafotias G, Hoogendoorn M, Eiben AE (2015) Parameter control in evolutionary Algorithms: trends and challenges. *IEEE Trans Evol Comput* 19(2):167–187. <https://doi.org/10.1109/TEVC.2014.2308294>
- Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B (2021) Swin transformer: Hierarchical vision transformer using shifted windows. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp 10012–10022
- Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference 1(Mlm)*, pp 4171–4186 [arXiv:1810.04805](https://arxiv.org/abs/1810.04805)
- Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. [arXiv:1509.02971](https://arxiv.org/abs/1509.02971)
- Wang F, Zhang H, Li K, Lin Z, Yang J, Shen X-L (2018) A hybrid particle swarm optimization algorithm using adaptive learning strategy. *Inform Sci* 436–437:162–177. <https://doi.org/10.1016/j.ins.2018.01.027>
- Liang JJ, Qin AK, Suganthan PN, Baskar S (2006) Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans Evol Comput* 10(3):281–295. <https://doi.org/10.1109/TEVC.2005.857610>
- Bartz-Beielstein T, Preuss M (2007) Experimental research in evolutionary computation. In: *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, pp 3001–3020
- Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. *Genetic and evolutionary computation conference*
- Hutter F, Hoos HH, Leyton-Brown K, Stuetzle T (2009) Paramils: an automatic algorithm configuration framework. *J Artificial Intell Res* 36:267–306
- Nannen V, Eiben AE (2006) A method for parameter calibration and relevance estimation in evolutionary algorithms. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. GECCO '06*. Association for Computing Machinery, New York, NY, USA, pp 183–190. <https://doi.org/10.1145/1143997.1144029>
- Bartz-Beielstein T, Lasarczyk CWG, Preuss M (2005) Sequential parameter optimization. In: *2005 IEEE Congress on Evolutionary Computation*, vol 1, pp 773–7801. <https://doi.org/10.1109/CEC.2005.1554761>
- Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello CAC (ed) *Learning and intelligent optimization*. Springer, Berlin, Heidelberg, pp 507–523
- Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, pp 1–9
- Eberhart RC, Shi Y (2000) Comparing inertia weights and constriction factors in particle swarm optimization. In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, vol 1. IEEE, pp 84–88
- Ratnaweera A, Halgamuge SK, Watson HC (2004) Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Trans Evol Comput* 8(3):240–255. <https://doi.org/10.1109/TEVC.2004.826071>
- Zheng Y-L, Ma L-H, Zhang L-Y, Qian J-X (2003) On the Convergence Analysis and Parameter Selection in Particle Swarm Optimization, vol 3, pp 1802–1807. cited By 218

28. Cui H-M, Zhu Q-B (2007) Convergence analysis and parameter selection in particle swarm optimization. *Comput Eng Appl* 43(23):89–91 (cited By 25)
29. Yu H-J, Zhang L-P, Chen D-Z, Hu S-X (2005) Adaptive particle swarm optimization algorithm based on feedback mechanism. *Zhejiang Daxue Xuebao (Gongxue Ban)/Journal of Zhejiang University (Engineering Science)* 39(9):1286–1291 (cited By 24)
30. Chen G, Huang X, Jia J, Min Z (2006) Natural exponential inertia weight strategy in particle swarm optimization. In: 2006 6th World Congress on Intelligent Control and Automation, vol 1. IEEE pp 3672–3675
31. Guimin C, Jianyuan J, Qi H (2006) Study on the strategy of decreasing inertia weight in particle swarm optimization algorithm. *J Xi'an Jiaotong Univ* 40(1):53–56
32. Malik RF, Rahman TA, Hashim SZM, Ngah R (2007) New particle swarm optimizer with sigmoid increasing inertia weight. *Int J Comput Sci Security* 1(2):35–44
33. Feng Y, Yao Y-M, Wang A-X (2007) Comparing with chaotic inertia weights in particle swarm optimization. In: 2007 International Conference on Machine Learning and Cybernetics, vol 1. IEEE, pp 329–333
34. Chen K, Zhou F, Liu A (2018) Chaotic dynamic weight particle swarm optimization for numerical function optimization. *Knowl-Based Syst* 139:23–40
35. Eberhart RC, Shi Y (2001) Tracking and optimizing dynamic systems with particle swarms. In: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546), vol 1. IEEE, pp 94–100
36. Tanabe R, Fukunaga A (2013) Success-history based parameter adaptation for differential evolution. In: 2013 IEEE Congress on Evolutionary Computation. IEEE, pp 71–78
37. Lynn N, Suganthan PN (2017) Ensemble particle swarm optimizer. *Appl Soft Comput* 55:533–548. <https://doi.org/10.1016/j.asoc.2017.02.007>
38. Liu Z, Nishi T (2020) Multipopulation ensemble particle swarm optimizer for engineering design problems. *Mathematical Problems in Engineering* 2020
39. Tatsis VA, Parsopoulos KE (2017) Grid-based parameter adaptation in particle swarm optimization. In: 12th Metaheuristics International Conference (MIC 2017), pp 217–226
40. Tatsis VA, Parsopoulos KE (2019) Dynamic parameter adaptation in metaheuristics using gradient approximation and line search. *Appl Soft Comput* 74:368–384
41. Olivas F, Valdez F, Castillo O, Melin P (2016) Dynamic parameter adaptation in particle swarm optimization using interval type-2 fuzzy logic. *Soft Comput* 20(3):1057–1070. <https://doi.org/10.1007/s00500-014-1567-3>
42. Melin P, Olivas F, Castillo O, Valdez F, Soria J, Valdez M (2013) Optimal design of fuzzy classification systems using pso with dynamic parameter adaptation through fuzzy logic. *Expert Syst Appl* 40(8):3196–3206. <https://doi.org/10.1016/j.eswa.2012.12.033>
43. Xu Y, Pi D (2020) A reinforcement learning-based communication topology in particle swarm optimization. *Neural Comput Appl* 32(14):10007–10032
44. Liu Y, Lu H, Cheng S, Shi Y (2019) An adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning. In: 2019 IEEE Congress on Evolutionary Computation (CEC). IEEE, pp 815–822
45. Samma H, Lim CP, Saleh JM (2016) A new reinforcement learning-based memetic particle swarm optimizer. *Appl Soft Comput* 43:276–297
46. Lu L, Zheng H, Jie J, Zhang M, Dai R (2021) Reinforcement learning-based particle swarm optimization for sewage treatment control. *Complex Intell Syst* 7(5):2199–2210
47. Hsieh Y-Z, Su M-C (2016) A q-learning-based swarm optimization algorithm for economic dispatch problem. *Neural Comput Appl* 27(8):2333–2350
48. Wu D, Wang GG (2022) Employing reinforcement learning to enhance particle swarm optimization methods. *Eng Opt* 54(2):329–348
49. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. *Advances in neural information processing systems* 30
50. Liang J, Qu B, Suganthan P, Hernández-Díaz AG (2013) Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212(34):281–295
51. Peram T, Veeramachaneni K, Mohan CK (2003) Fitness-distance-ratio based particle swarm optimization. In: 2003 IEEE Swarm Intelligence Symposium, SIS 2003 - Proceedings (2), pp 174–181. <https://doi.org/10.1109/SIS.2003.1202264>
52. Qu BY, Suganthan PN, Das S (2013) A distance-based locally informed particle swarm model for multimodal optimization. *IEEE Trans Evol Comput* 17(3):387–402. <https://doi.org/10.1109/TEVC.2012.2203138>
53. Engelbrecht AP (2010) Heterogeneous particle swarm optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6234 LNCS, pp 191–202. https://doi.org/10.1007/978-3-642-15461-4_17
54. Shi Y, Eberhart R (1998) A modified particle swarm optimizer. In: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360). IEEE, pp 69–73
55. Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput* 1(1):3–18. <https://doi.org/10.1016/j.swevo.2011.02.002>
56. García-Martínez C, Lozano M (2010) Evaluating a local genetic algorithm as context-independent local search operator for metaheuristics. *Soft Comput* 14(10):1117–1139. <https://doi.org/10.1007/s00500-009-0506-1>
57. Melin P, Olivas F, Castillo O, Valdez F, Soria J, Valdez M (2013) Optimal design of fuzzy classification systems using PSO with dynamic parameter adaptation through fuzzy logic. *Expert Syst Appl* 40(8):3196–3206. <https://doi.org/10.1016/j.eswa.2012.12.033>
58. Tanabe R, Fukunaga A (2013) Success-history based parameter adaptation for Differential Evolution. In: 2013 IEEE Congress on Evolutionary Computation, CEC 2013 (3), pp 71–78. <https://doi.org/10.1109/CEC.2013.6557555>
59. Liu W, Wang Z, Yuan Y, Zeng N, Hone K, Liu X (2021) A novel sigmoid-function-based adaptive weighted particle Swarm optimizer. *IEEE Trans Cybernet* 51(2):1085–1093. <https://doi.org/10.1109/TCYB.2019.2925015>
60. Zhang H, Yuan M, Liang Y, Liao Q (2018) A novel particle swarm optimization based on prey-predator relationship. *Appl Soft Comput* 7 J 68:202–218. <https://doi.org/10.1016/j.asoc.2018.04.008>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.