Engineering Optimization

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/geno20

Employing reinforcement learning to enhance particle swarm optimization methods

Di Wu & G. Gary Wang

To cite this article: Di Wu & G. Gary Wang (2021): Employing reinforcement learning to enhance particle swarm optimization methods, Engineering Optimization, DOI: 10.1080/0305215X.2020.1867120

To link to this article: https://doi.org/10.1080/0305215X.2020.1867120

Published online: 17 Jan 2021.

ENGINEERING OPTIMIZATION
https://doi.org/10.1080/0305215X.2020.1867120

Employing reinforcement learning to enhance particle swarm optimization methods

Di Wu and G. Gary Wang

Product Design and Optimization Laboratory, Simon Fraser University, Surrey, BC, Canada

ABSTRACT

Particle swarm optimization (PSO) is a well-known optimization algorithm that shows good performance in solving different optimization problems. However, PSO usually suffers from slow convergence. In this article, a reinforcement learning strategy is developed to enhance PSO in convergence by replacing the uniformly distributed random number in the updating function with a random number generated from a selected normal distribution. In the proposed method, the mean and standard deviation of the normal distribution are estimated from the current state of each individual through a policy net. The historic behaviour of the swarm group is used to update the policy net and guide the selection of parameters of the normal distribution. The proposed method is integrated into the original PSO and a state-of-the-art PSO, called the self-adaptive dynamic multi-swarm PSO (sDMS-PSO), and tested with numerical functions and engineering problems. The test results show that the convergence rate of PSO methods can be improved with the proposed reinforcement learning strategy.

ARTICLE HISTORY
Received 18 June 2020
Accepted 16 December 2020

KEYWORDS
PSO; reinforcement learning; policy search; policy gradient; swarm-intelligence

1. Introduction

Metaheuristic optimization algorithms have been widely studied in the past decades and applied in various engineering fields due to their gradient-free property (Beheshti and Shamsuddin 2013). Among them, swarm intelligence algorithms, such as particle swarm optimization (PSO) (Kennedy and Eberhart 1995), ant colony optimization (ACO) (Dorigo, Maniezzo, and Colorni 1996), artificial bee colony (ABC) (Karaboga 2005), grey wolf optimization (GWO) (Mirjalili, Mirjalili, and Lewis 2014), and so on, show high performance in solving a variety of optimization problems. In this group of algorithms, the behaviour of social animals/insects is studied and simulated to find the global optimum. Those algorithms, however, have issues such as converging only to a local optimum and low convergence rate (Beheshti and Shamsuddin 2013). One way to alleviate such problems is to adjust the parameters in the algorithm according to different optimization problems. Other methods may include changing the topology of the algorithm or simulating different animal social behaviours. However, those modifications are all based on the experiences of researchers. No matter what topology structure is used or what kind of behaviour is simulated, those structures or behaviours are defined by the algorithm developers through mathematical functions. Parameters in such mathematical functions are usually fixed and the mathematical relations are all defined *a priori*. Such parameters and relations are assumed to be applicable to all problems, which is a very audacious presumption. The research question is then, ‘is there a way to improve the intelligence of swarm-intelligence algorithms

CONTACT G. Gary Wang gary_wang@sfu.ca

by self-learning from previous iterations or swarm behaviour?' In the artificial intelligence field, reinforcement learning is a method of learning the behaviour and determining actions based on the learned policy (Sutton and Barto 1998). In this article, the reinforcement learning method is applied in PSO to give more intelligence to each individual to select their preference in the optimization process.

Reinforcement learning has been used in multiple metaheuristic optimization algorithms including PSO (Wauters *et al.* 2013), while Q-learning is the most widely used reinforcement learning method. In Samma, Lim, and Mohamad Saleh (2016), five different operations for the particles in PSO are defined, including exploration, convergence, high-jump, low-jump and fine-tuning, and Q-learning is applied to each particle to select one operation from the five candidates based on the current state. Y. Xu and Pi (2020) employed the Q-learning method to select different communication topologies at each iteration of the PSO process. The Q-learning method was also applied in a simulated annealing (SA) algorithm to solve constrained engineering problems, where reinforcement learning was used to choose the best parameter values from a set of candidates (Samma *et al.* 2020). In Misir *et al.* (2009), learning automata (LA) were also used to determine the parameter values from six candidates in the iteration limited threshold acceptance (ILTA) method. Additionally, reinforcement learning methods were applied in specific fields, such as Cytosine phosphodiester Guanine (CpG) islands prediction (Chuang *et al.* 2011) and noisy problems (Piperagkas *et al.* 2012), where the operations in the PSO process were selected by reinforcement learning methods. One characteristic of the existing reinforcement learning enhanced metaheuristic optimization algorithms is that reinforcement learning is limited to the selection of actions from a given candidate set. The use of reinforcement learning methods in the parameter tuning of optimization algorithms has not been explored.

There are other works that used reinforcement learning to assist optimization. Li and Malik (2017) used a reinforcement learning method to learn from existing gradient-based optimization methods and employed a well-trained policy net to optimize the weights of different neural networks in classification. A continuous reinforcement learning method based on the normal distribution was applied to find the policy/updating function that has the highest convergence rate. In these works, once the policy (updating function) was trained, the policy would not change when used to optimize other problems. This method, however, has limitations in practice. First, gradient information is required in the method, which is not easy to obtain in most real-world optimization problems. Secondly, the policy is not updated according to the optimization problem, which means that the problem to be solved needs to be the same or sufficiently similar to the problems that have been used to train the policy net.

In this article, a continuous reinforcement learning strategy is proposed and applied in PSO and its modifications to improve the convergence rate of the original method. Multiple studies on the parameters in the PSO method, including the inertia weight and the acceleration coefficients, have been performed (Clerc and Kennedy 2002; Y. Shi and Eberhart 1998, 1999). Except for those parameters, the other two uniformly distributed random numbers in the updating function have never been studied before by other researchers. In PSO, these two random numbers are used to increase the uncertainty of the algorithm to help the algorithm to converge to the global optimum. The question is, 'does the uniform distribution assumption limit the performance of PSO?' This work will replace the uniform distribution with the normal distribution and report the effect of such a change.

A normal distribution with selected mean and standard deviation is used to represent the preference of each individual, which is estimated from a policy net according to the current state of each individual and the state of the swarm group. The policy net will be updated during the optimization process by promoting the positive actions and punishing the negative actions in the historical behaviour.

The rest of this article is organized as follows. Section 2 briefly introduces the PSO and reinforcement learning methods. Section 3 explains the proposed reinforcement learning strategy in the original PSO method. The experimental results of adapting reinforcement learning into the original

PSO and the self-adaptive dynamic multi-swarm PSO (sDMS-PSO) are shown in Section 4. Section 5 concludes the article.

2. Related materials

2.1. Particle swarm optimization

PSO is one of the most popular swarm intelligence metaheuristic algorithms simulating the motion of birds or insects to pursue the best solution of optimization problems (Koziel and Yang 2011). Initially, the birds or insects, called particles, are randomly distributed in the design space. At the *i*th iteration, the position of each particle is updated by Equation (1):

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \boldsymbol{v}_{i+1}, \tag{1}$$

where $\boldsymbol{x}_i$ is the current position of the particle while $\boldsymbol{x}_{i+1}$ is the newly generated position. $\boldsymbol{v}_{i+1}$ is the velocity of the particle, which is updated according to the current velocity ($\boldsymbol{v}_i$), the best personal position (*pBset*), and the best position of all particles (*gBest*), as shown in Equation (2):

$$v_{i+1} = w_i \times v_i + c_1 \times rnd \times (pBest - x_i) + c_2 \times rnd \times (gBest - x_i), \tag{2}$$

where w is the inertia weight, c_1 and c_2 are acceleration coefficients, and rnd is a uniformly distributed random number between zero and one. In practice, w is usually decreased from 0.9 to 0.4 (Y. Shi and Eberhart 1999).

Not only being applied in different fields, PSO is also a widely studied optimization algorithm. Multiple convergence analysis and stability studies have been presented in Clerc and Kennedy (2002), Kadirkamanathan, Selvarajah, and Fleming (2006) and Trelea (2003), which try to explain why PSO suffers from the issues of slow convergence and easily falling into a local optimum. There are two types of modification in PSO studies to improve the performance: topological structures and parameter studies. Different topological structures, such as ring topology (LPSO) (Kennedy and Mendes 2002) and Von Neumann topology (VPSO) (Mendes 2006) are developed to increase the convergence rate and avoid trapping into local optima. In the dynamic multi-swarm particle swarm optimizer (DMS-PSO), swarms are regrouped frequently to form a dynamic population (X. Xu *et al.* 2015). In guided adaptive search-based particle swarm optimizer (GuASPO) (Rezaei and Safavi 2020), the personal best particles are all divided into different clusters and the unique global best at a cluster is obtained as a weighted average calculated over other clusters' best particles. Tian, Zhao, and Z. Shi (2019) use three different kinds of mutation strategy (Gaussian, Cauchy and chaotic mutations) to tackle the premature convergence issue of PSO. A normal-distribution-based update mechanism is proposed (Kiran 2017). The new positions are generated from a normal distribution, where the mean and standard deviation are calculated according to the personal best and the global best. On the other hand, experiments are applied to show that both acceleration coefficients c_1 and c_2 have obvious impact on the performance of PSO. The most popular choice is to fix the values of c_1 and c_2 at two (Kennedy and Eberhart 1995). Additionally, time-varying acceleration coefficients are also studied in Ratnaweera, Halgamuge, and Watson (2004) to find the best coefficient values during the optimization process. This work focuses on the two random numbers in the PSO updating functions. The normal distribution generated from reinforcement learning methods is used to replace the uniform distribution.

2.2. Reinforcement learning and policy gradient

Reinforcement learning is to determine how agents take actions in an environment, in order to maximize the cumulative reward (Kaelbling, Littman, and Moore 1996). In the standard reinforcement learning model, an agent is connected to its environment via perception and action. At each time step, the agent observes the current state of the environment and then takes actions to a new state. The environment will provide a reward or punishment for this action. After all actions are taken, a score can be generated by summation of the reward or punishment provided for each action in the entire process, where a sequence of the actions is generated from a programmed policy. The goal of reinforcement learning is to generate the optimal policy for the agent to maximize the score of this action.

Reinforcement learning can be described as a Markov decision process (MDP) (Arulkumaran *et al.* 2017), which consists of a set of states **S**, a set of actions $\boldsymbol{A}$ and an immediate/instantaneous reward function *R*. In general, the policy π is a mapping from states $\boldsymbol{S}$ to a probability distribution over a given action set $\boldsymbol{A} = \boldsymbol{a}$

$$\pi : \boldsymbol{S} \rightarrow p(\boldsymbol{A} = \boldsymbol{a}|\boldsymbol{S}), \tag{3}$$

After a time step of length *T*, every rollout of a policy accumulates rewards from the environment resulting in the return

$$R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}, \tag{4}$$

where r is the reward or punishment caused by the current action, and γ is a discount factor between zero and one to determine the importance of immediate rewards. A low value of γ means more emphasis on the immediate reward. Hence, the goal of reinforcement learning can be formulated as follows:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E[R|\pi]. \tag{5}$$

There are two main strategies for solving reinforcement learning problems: methods based on value functions (Bellman 1952) and methods based on policy search (Deisenroth 2011). Instead of finding a general policy, value function methods tend to find the best action for a given state and then link all actions to generate the entire policy. The main drawback of this kind of method is that the number of states and actions needs to be finite and known. On the other hand, policy search methods do not need to maintain a value function but directly search for an optimal policy π^*, which can be performed in a continuous space. In this kind of method, neural networks are usually used to encode policies. In this work, the policy gradient method is chosen as one of the policy search methods (Deisenroth 2011).

In the policy gradient method, the gradient ascent method is used to maximize the expected return $E[R|\pi]$. Assuming that a neural network, called a policy net, is used to represent the policy and $\boldsymbol{\theta}$ is the weight vector in the neural network, $\boldsymbol{\theta}$ is updated at each time step according to the following equation:

$$\theta_{t+1} = \theta_t + \alpha \cdot \nabla_\theta E[R|\pi], \tag{6}$$

where α is a user-specified learning rate and $\nabla_{\boldsymbol{\theta}} E[R|\pi]$ is the policy gradient. One way to estimate the policy gradient is by using the 'likelihood-ratio' trick as shown in Equation (7) (Williams 1992):

$$\nabla_\theta E[R|\pi] = E[\nabla_\theta \log(p_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}))], \tag{7}$$

where the expectation over $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ is approximated by using a sum over the sampled trajectories$\boldsymbol{\tau} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots)$.

For example, if there are N sampled trajectories $(\boldsymbol{\tau}^{(1)}, \boldsymbol{\tau}^{(2)}, \ldots, \boldsymbol{\tau}^{(N)})$, the expectation can be estimated as

$$E[\log(p_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau}))] = \frac{1}{N}\sum_{i=1}^{N} \log(p_\theta(\boldsymbol{\tau}^{(i)})R(\boldsymbol{\tau}^{(i)})). \tag{8}$$

The policy gradient method can be extended to continuous spaces by assuming the policy follows a normal distribution,

$$\pi_w(a_t|S_t) = N(\mu_w(S_t), \sigma_w(S_t)), \tag{9}$$

where $N(\mu_w(S_t), \sigma_w(S_t))$ denotes the probability density function of a normal distribution with mean $\mu_w(S_t)$ and standard deviation $\sigma_w(S_t)$.

3. Reinforcement learning in particle swarm optimization

Compared to the uniform distribution, a normal distribution allows for a higher sample preference by changing the mean and standard deviation. To give more intelligence to the swarms, the uniformly distributed random number is replaced by a random number following a normal distribution. A reinforcement learning method is used to determine the mean and standard deviation of the normal distribution according to the particle position, the personal best position and the global best position. The original PSO is used to explain how to utilize reinforcement learning in PSO algorithms. Following a similar process, the proposed strategy can be applied to different PSO modifications.

3.1. Using reinforcement learning in the original PSO (RL-PSO)

As shown in Equation (2), two uniformly distributed random numbers are involved in the updating function to increase the exploration ability of the PSO method, but this may decrease the exploitation of the algorithm. In RL-PSO, the random numbers are generated from two normal distributions rather than the uniform distribution, as shown in Equation (10):

$$\begin{aligned} v_{i+1} &= w_i \times v_i + c_1 \times a_1 \times (pBest - x_i) + c_2 \times a_2 \times (gBest - x_i) \\ a_1 &\sim N(\mu_1, \sigma_1) \\ a_2 &\sim N(\mu_2, \sigma_2), \end{aligned} \tag{10}$$

where a_1 and a_2 are the random numbers following the normal distribution with means μ_1 and μ_2 and standard deviations σ_1 and σ_2, respectively.

At each iteration, μ and σ are updated according to the current position, personal best position and global best position to balance the exploration and exploitation by the current state and historical experience. A policy net using a neural network is constructed to realize the mapping from the current state to the values of μ and σ. The policy net is updated at each iteration by learning from the historical behaviour of the swarm group. Hence, the RL-PSO algorithm is presented as follows.

As shown in Figure 1, two policy nets (*i.e.* π_1 and π_2) are constructed to respectively determine the means and standard deviations for two normal distributions. According to the associated terms in Equation (10), the input state of π_1 includes the current position and the personal best position of this individual, while the input state of π_2 includes the global best position as well as the current position. The structure and the updating process of the policy net are introduced in Section 3.2.

In each iteration of the RL-PSO, two normal distributions, $[\mu_1, \sigma_1]$ and $[\mu_2, \sigma_2]$, are generated from the policy net with the current state of the individuals and the weight vector. Two random numbers, a_1 and a_2, are generated following the two generated normal distributions. After updating the position of each individual according to Equation (10) and calculating the response values for the individual, the weight vector policy net is updated according to the score, *i.e.* the objective function value at the new position, via the policy gradient method.

```
Algorithm: RL-PSO
Input: Population (nP), Maximum iteration (MaxIter), Initial weight vectors of the policy nets ($\boldsymbol{\theta}_1^{(1)}, \boldsymbol{\theta}_1^{(2)}$)
Randomly initialize a population of nP individuals
Calculate function values at initial points
Initialize policy nets $\pi_1$ and $\pi_2$
for i = 1 to MaxIter
        for j = 1 to nP
                $[\mu_1, \sigma_1] \leftarrow \pi_1(x_i^{(j)}, pBest^{(j)}, \boldsymbol{\theta}_i^{(1)})$
                $[\mu_2, \sigma_2] \leftarrow \pi_2(x_i^{(j)}, gBest, \boldsymbol{\theta}_i^{(2)})$
                $a_1 \sim N(\mu_1, \sigma_1)$
                $a_2 \sim N(\mu_2, \sigma_2)$
                Update $v_{i+1}^{(j)}$ via Eq. (10)
                $\boldsymbol{x}_{i+1}^{(j)} = \boldsymbol{x}_i^{(j)} + v_{i+1}^{(j)}$
                Calculate the function value at the new position
                Update policy nets $\pi_1$ and $\pi_2$
        end
        Update $pBest^{(j)}$ & $gBest$
end
Output $gBest$ and function value at $gBest$
```

Figure 1. Algorithm of reinforcement learning enhanced PSO (RL-PSO).

3.2. Policy network

In RL-PSO, two policy nets are used to learn from previous behaviours and guide the behaviour in the next iteration. Since there are two random numbers in the updating function, two policy nets are employed to generate two different normal distributions with different inputs: one using the personal best position as the input state and the other using the global best position. In this section, one policy net π_1 is used to introduce the construction and updating method. The other policy net π_2 follows the same process.

As shown in Figure 2, a multi-layer feedforward neural network with two hidden layers and four neurons in each layer is constructed to realise the mapping from the state to action. In this policy network, the position of the individual and the position of the personal best are defined as the state, while the position of the global best along with the individual position are selected as the state in the other policy net, π_2. A μ, σ pair is calculated by the neural network. Next, the random number a is generated from the normal distribution with the obtained μ and σ. The bounds of μ and σ are set to be [0, 1]. When the values of a is outside the bounds, the action a value is scaled from $[\mu - 3\sigma, \mu + 3\sigma]$ to [0, 1]. After updating the position of the individual, the response of the new point is obtained as f_2, while f_1 stands for the function value at the previous position. The new function value f_2 along with f_1 are used to update the weight vector in the neural network.

A modified policy gradient method is used to update the policy net. Instead of the cumulative reward, the instant reward obtained from the current action is considered in the proposed strategy to maximize the reduction of the function value at each iteration of the optimization process. Hence,

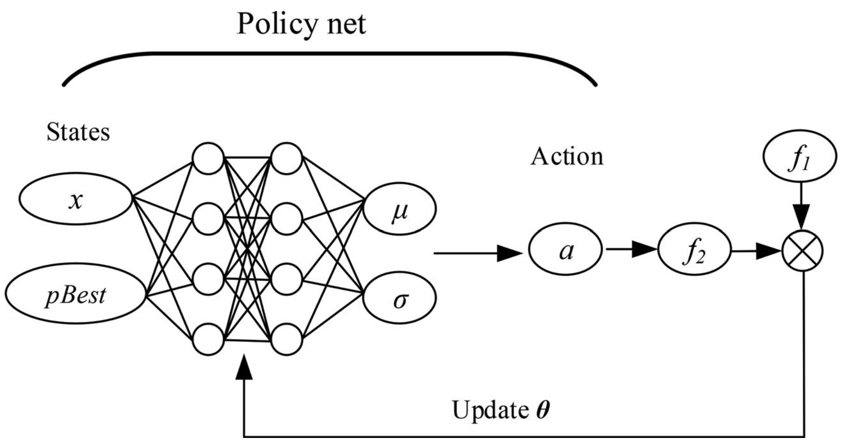

Figure 2. Policy net π_1 and its updating process.

the goal of the policy net is modified as follows:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\left[-\log(p(a))\frac{f_2 - f_1}{f_{\max} - f_{\min}}\right]. \tag{11}$$

In Equation (11), two relations between f_2 and f_1 are considered, $f_2 < f_1$ or $f_2 \geq f_1$. When $f_2 < f_1$, this means that the action taken can find a smaller function value, which means this action is positive and the probability of taking this action needs to be improved. On the other hand, if $f_2 \geq f_1$, the action is negative, the probability of which needs to be reduced. The maximum function value ($f_{\max}$) and the minimum function value ($f_{\min}$) in history are used to normalize the differences between f_2 and f_1.

The policy net will be trained after an individual updating using Equation (12):

$$\theta_{i+1} = \theta_i + \alpha \cdot g, \tag{12}$$

where $\boldsymbol{\theta}$ are the weights of the policy net, α is a pre-defined learning rate and g is the gradient of Equation (11), which is calculated as follows:

$$g = -\nabla_\theta \log(p(a))\frac{f_2 - f_1}{f_{\max} - f_{\min}}. \tag{13}$$

For a given normal distribution with μ and σ, the probability of the action a, $p(a)$, can be calculated by the normal distribution probability density function as defined by Equation (13):

$$p(a) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2\right\}. \tag{14}$$

When $a = \mu$, the probability has its largest value. Since$p(a) \leq 1$, σ should be larger than 0.398. Therefore, the boundary of σ output from the policy net is set to be [0.4,1], while the boundary of μ is set to be [0, 1].

By updating the weight vector at each iteration, the positive action (*i.e.* decreasing function value) can be promoted by increasing the probability and the negative (*i.e.* increasing function value) will be punished by reducing the probability. Therefore, each individual can learn from previous states to determine the action taken at the current state.

It should be noted that updating of the policy net does not add extra objective function evaluations in RL-PSO since the function values, f_1 and f_2, have already been computed and do not change when calculating the gradient. Additionally, the policy nets update once for each individual at one iteration, which means the policy nets update *nP* (the population size) times at one iteration in RL-PSO and update $nP \times MaxIter$ times in the entire RL-PSO process. As updating happens within the RL-PSO process and is agnostic with respect to specific optimization problems, the policy nets can adapt to any optimization problem without change.

4. Tests of the method

In this section, the reinforcement learning strategy is applied in the original PSO first and the performance of the RL-PSO is tested by multiple numerical benchmark problems. A modification of the RL-PSO is developed to deal with high-dimensional problems. Next, the proposed strategy is applied to a participating algorithm of CEC 2015, called the self-adaptive dynamic multi-swarm PSO (sDMS-PSO) (Liang *et al.* 2015), as one of the state-of-the-art PSO algorithms, and compared to the original sDMS-PSO using CEC 2015 benchmark problems (Liang *et al.* 2014). Finally, the reinforcement learning strategy is applied in the original PSO to solve an engineering design problem.

4.1. Numerical tests of RL-PSO

The formulae of the problems are shown as follows.

Six-hump camel function (SC)

$$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{4}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \quad x_1 \in [-3,3], \quad x_2 \in [-2,2] \tag{15}$$

Bukin function (BF)

$$f(x) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10|$$
$$x_1 \in [-15,-5], \quad x_2 \in [-3,3] \tag{16}$$

Rosenbrock function

$$f(x) = \sum_{i=1}^{d-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$
$$x_i \in [-10,10], \quad d = 2, 10, 20, 30 \tag{17}$$

Branin function (BR)

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10 \quad x_1 \in [-5,10], \quad x_2 \in [0,15] \tag{18}$$

Goldstein–Price function (GP)

$$f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$\times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$
$$x_1, x_2 \in [-2,2] \tag{19}$$

Griewank function

$$f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$x_i \in [-100, 100], \quad d = 2, 10, 20, 30 \tag{20}$$

Hartmann 3-D function (HN)

$$f(\boldsymbol{x}) = -\sum_{i=1}^{4} \alpha_i exp\left[-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right]$$

$$x_1, x_2, x_3 \in [0, 1]$$

$$\alpha = [1, 1.2, 3, 3.2]^{\mathrm{T}}, \quad \boldsymbol{A} = \begin{bmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix},$$

$$\boldsymbol{P} = 10^{-4}\begin{bmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8808 \end{bmatrix} \tag{21}$$

Ackley function

$$f(x) = -20exp\left[-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right] - exp\left[\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right] + 20 + e$$

$$x_i \in [-32, 32], \quad d = 10, 20, 30 \tag{22}$$

Rastrigin function

$$f(x) = 10d + \sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i)]$$

$$x_i \in [-5.12, 5.12], \quad d = 10, 20, 30. \tag{23}$$

RL-PSO is compared to the original PSO to illustrate the performance of the reinforcement learning strategy. Additionally, a PSO algorithm using normally distributed random numbers for both the *pbest* and *gbest* terms in Equation (2) is also tested, and referred to as PSO-normal. In PSO-normal, the mean and standard deviation are both fixed at 0.5, and the random number generated from the normal distribution is normalized to [0, 1]. In all the three algorithms, the inertia weight w is reduced from 0.9 to 0.4. Both acceleration coefficients are set to be two. In this experiment, the number of function evaluations is limited to a small number to test the efficiency of the RL-PSO method. For 2-D and 3-D problems, the population size is set to be five and the maximal number of iterations is set to be 20 for both methods. Thus, the maximum number of function evaluations is 100. For 10-D, 20-D and 30-D problems, the maximal number of function evaluations is set to be 1000, with population size 10 and 100 iterations. Each problem is run 20 times and the mean optimization results are shown in Table 1.

As shown in Table 1, RL-PSO outperforms PSO in all of the seven low-dimensional problems. The convergence curves of the seven low-dimensional optimization problems are shown in Figure 3. It can be seen that the objective function value in RL-PSO drops faster than in the original PSO for most of the problems for the first five to ten iterations except for the Rosenbrock and GP functions. For low-dimensional problems, PSO-normal performs better than the original PSO but worse than RL-PSO. The boxplots of the optimal results for the low-dimensional optimization problems are shown in Figure 4. In all problems, the range of the optimization results of RL-PSO is smaller than those from

Table 1. Comparison between RL-PSO and PSO on benchmark problems.

		Theoretical	PSO		PSO-normal		RL-PSO	
	Dim	optimum	Mean	Std	Mean	Std	Mean	Std
SC	2	−1.032	−0.944	0.250	−0.954	0.187	**−1.030**	**0.003**
BF	2	0	6.517	3.983	2.317	1.227	**2.073**	**1.152**
BR	2	0.398	0.425	0.063	0.435	0.101	**0.400**	**0.004**
GP	2	3	5.148	**6.157**	25.037	32.984	**4.772**	6.221
HN	3	−3.863	−3.678	0.634	−3.535	0.647	**−3.757**	**0.242**
Rosenbrock	2	0	3.574	4.028	2.259	3.563	**2.174**	**3.351**
	10	0	**417.035**	**602.236**	479.854	1034.7	11433	20006
	20	0	9.291e4	1.186e5	**4.046e4**	**3.209e4**	6.165e5	8.251e5
	30	0	1.452e6	8.858e5	1.375e6	**1.072e6**	**7.114e5**	1.132e6
Griewank	2	0	0.197	0.171	0.185	0.182	**0.168**	**0.132**
	10	0	0.750	**0.185**	**0.742**	**0.225**	2.237	2.106
	20	0	2.121	0.573	**1.550**	**0.287**	5.738	4.451
	30	0	7.006	2.325	**3.198**	**0.725**	6.431	6.131
Ackley	10	0	5.274	4.016	**5.008**	**2.813**	10.896	6.987
	20	0	17.316	2.544	**12.314**	**2.267**	15.678	3.434
	30	0	20.034	**0.616**	**15.228**	0.946	18.141	1.389
Rastrigin	10	0	44.463	14.495	**25.742**	**10.575**	44.014	30.439
	20	0	175.026	38.516	**93.109**	**17.672**	123.324	74.784
	30	0	355.131	58.392	**188.241**	**32.923**	263.409	138.183

Note: Boldface numbers indicates the best performance among the three algorithms.

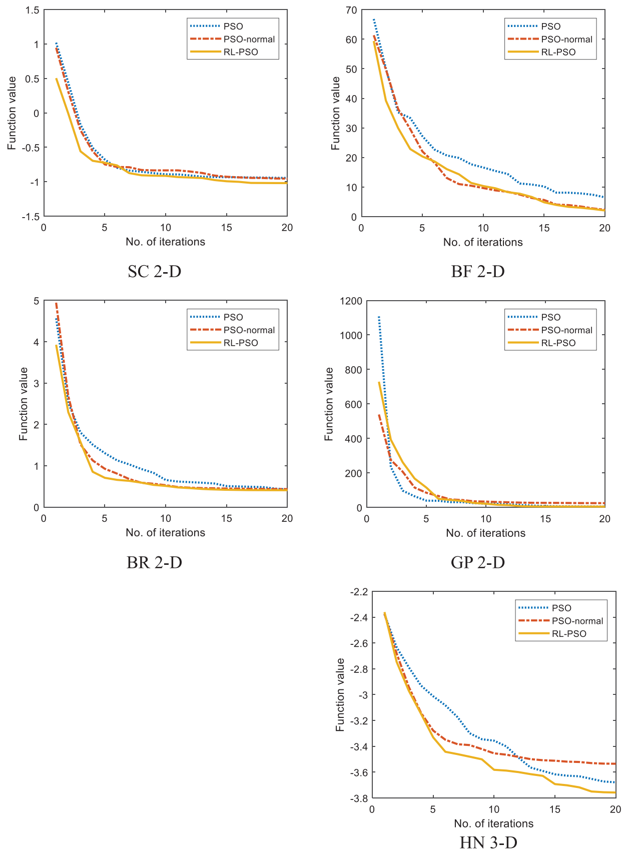

Figure 3. Convergence curves for low-dimensional optimization problems for PSO and RL-PSO.

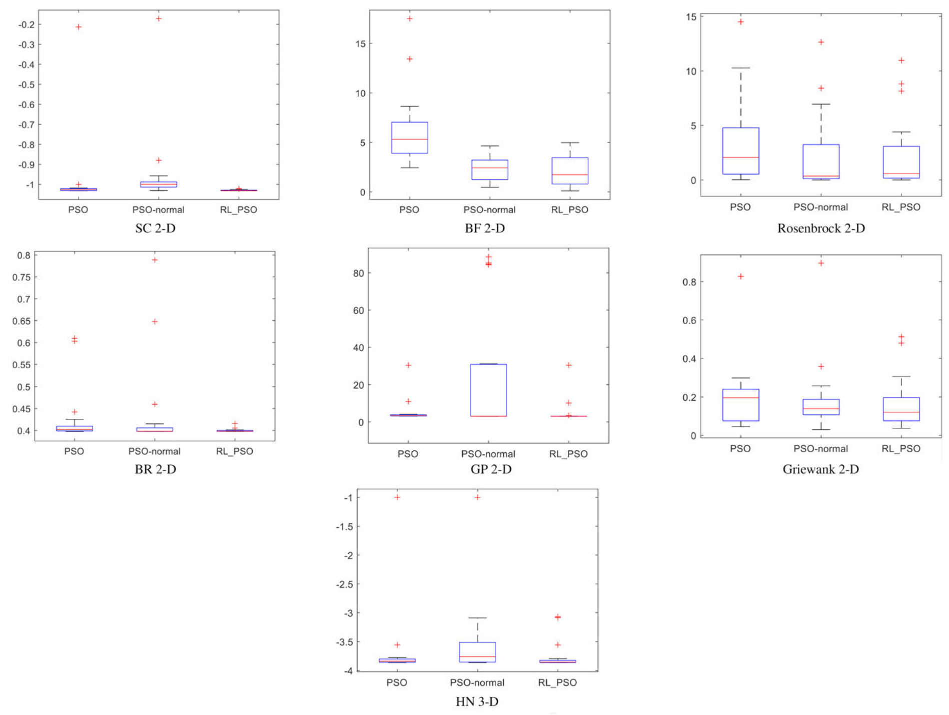

Figure 4. Boxplots of optimization results for low-dimensional problems for PSO and RL-PSO.

PSO. RL-PSO also shows overall smaller ranges than PSO-normal for these functions. For the high-dimensional problems, however, the proposed RL-PSO can only obtain better optimum results for seven out of twelve test problems as compared to the original PSO. PSO-normal is the best algorithm for the high-dimensional test problems, yielding better results than RL-PSO.

The reinforcement learning in RL-PSO seems to limit the exploitation ability of the algorithm, which causes early convergence in RL-PSO tests, especially for high-dimensional problems. In the early stage of the optimization, RL-PSO can find a better solution than the original PSO due to the aggressive updating strategy. Since the networks related to the personal bests and the global best are trained independently, the trained network related to the personal best tends to generate a new point close to the personal best, while the global best based network tends to force the point close to the global best. This independent training can explore the space faster at an early stage of the optimization, but it makes it hard for the RL-PSO algorithm to converge to the global best point because the samples usually spread everywhere in the design space rather than gather around the global best point. As a result, RL-PSO cannot find better solutions in the ensuing iterations.

To improve the performance of RL-PSO for high-dimensional problems, the exploitation ability needs to be enhanced in the RL-PSO process. In other words, the aggressiveness of RL-PSO pursuing the personal best should be reduced to improve the searching ability around the global best. Therefore, the random number for the personal best term (a_1) is generated from a uniform distribution, the same as in the original PSO, while the random number in the global best term (a_2) remains following the normal distribution generated from the reinforcement learning. Because of the normal distribution for the global best term, individuals tend to pursue the global best rather than the personal best. As a result, more individuals will gather around the global best, and the exploitation aspect of RL-PSO algorithm may be improved. The modified RL-PSO, called RL-PSO-G, is employed to solve the high-dimensional problems. Similarly, the original PSO-normal is adjusted to be PSO-normal-G, in which only the random number for the *gbest* term in Equation (2) is changed to be a normal distribution.

Table 2. Comparison results of PSO-normal, PSO-normal-G and RL-PSO-G on high-dimensional problems.

		PSO-normal		PSO-normal-G		RL-PSO-G	
	Dim	Mean	Std	Mean	Std	Mean	Std
Rosenbrock	10	479.854	1034.7	156.734	209.716	**120.804**	**172.382**
	20	4.046e4	3.209e4	8.585e3	8.792e3	**7.411e3**	**6.070e3**
	30	1.375e6	1.072e6	9.261e4	8.721e4	**9.172e4**	**6.884e4**
Griewank	10	0.742	0.225	0.308	0.182	**0.210**	**0.145**
	20	1.550	0.287	1.438	**0.213**	**1.419**	0.307
	30	3.198	0.925	3.043	0.825	**2.761**	**0.708**
Ackley	10	5.008	2.813	3.251	1.876	**2.896**	**1.731**
	20	12.314	2.267	11.141	3.584	**10.812**	**2.761**
	30	15.228	0.946	15.256	1.400	**14.158**	**2.540**
Rastrigin	10	25.742	10.575	21.553	8.519	**20.240**	**7.949**
	20	93.109	17.672	89.376	25.417	**79.951**	**19.910**
	30	188.241	40.717	186.322	30.320	**169.028**	**23.216**

Note: Boldface numbers indicates the best performance among the three algorithms.

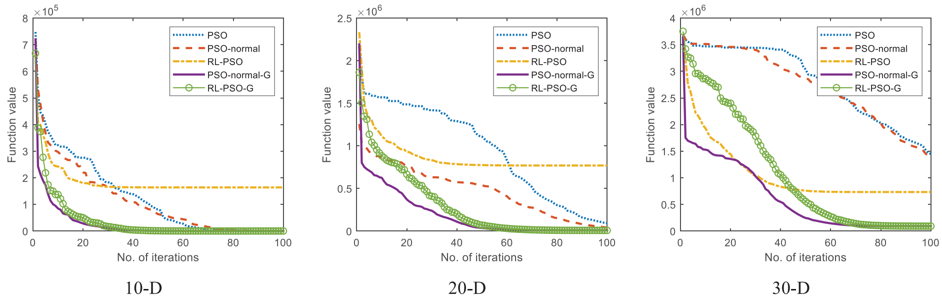

Figure 5. Convergence curves for the Rosenbrock function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

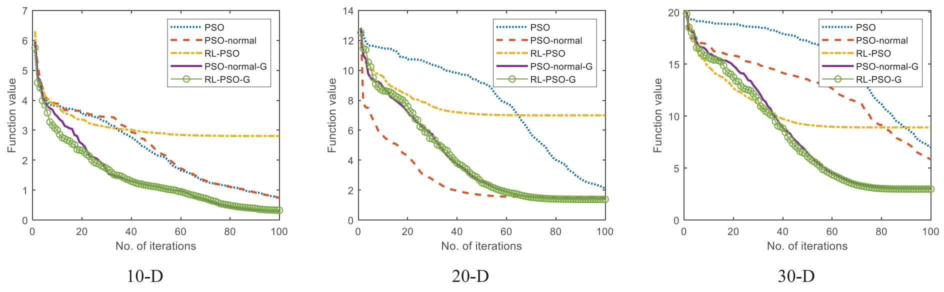

Figure 6. Convergence curves for the Griewank function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

RL-PSO-G is then compared to PSO-normal and PSO-normal-G. The comparison results are shown in Table 2.

As shown in Table 2, the modified RL-PSO-G performs better in all problems with different dimensionalities compared to others, including PSO and RL-PSO, data for which can be found in Table 1. The convergence curves, shown in Figures 5–8, illustrate that RL-PSO-G has an average drop rate at an early stage as compared to others but keeps finding better optimal results in the following iterations. Compared to the original PSO, RL-PSO-G can find smaller solutions and the improvement over PSO

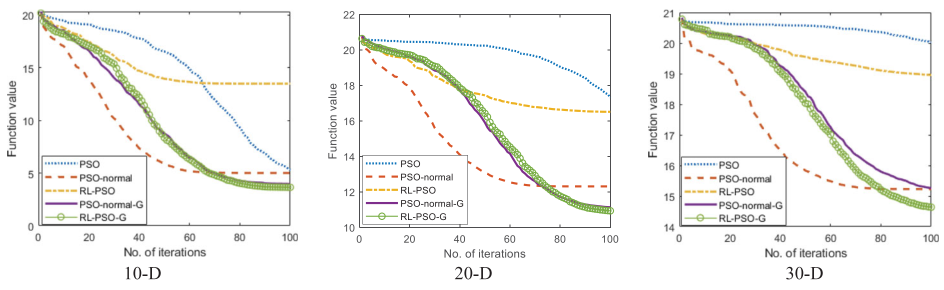

Figure 7. Convergence curves for the Ackley function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

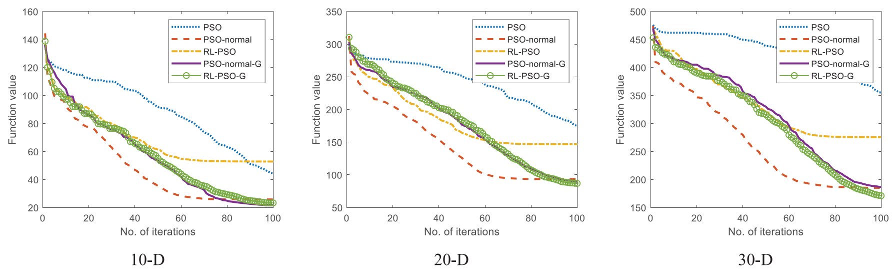

Figure 8. Convergence curves for the Rastrigin function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

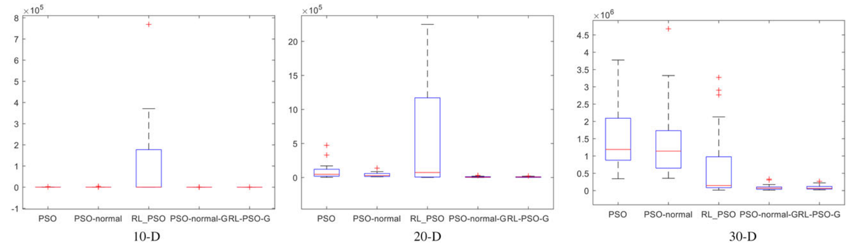

Figure 9. Boxplots of optimization results for the Rosenbrock function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

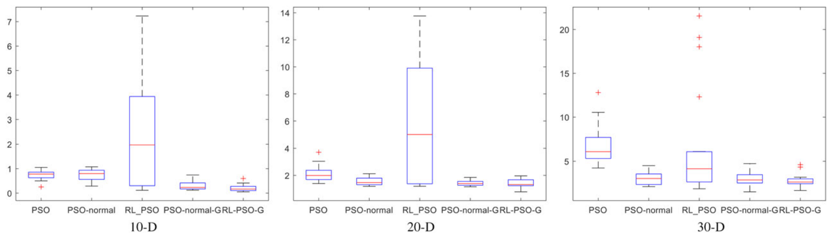

Figure 10. Boxplots of optimization results for the Griewank function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

becomes greater with increasing dimensionality. Figures 9–12 show boxplots for the five optimization algorithms and RL-PSO-G has the highest robustness compared to the other methods except for the Ackley function. The unmodified RL-PSO has the worst robustness for these high-dimensional problems, which confirms the earlier observation that the points spread out in the design space.

By involving the normal distribution in the PSO process, a more aggressive position updating function can be formed since the mean and the standard deviation in the normal distribution represent the

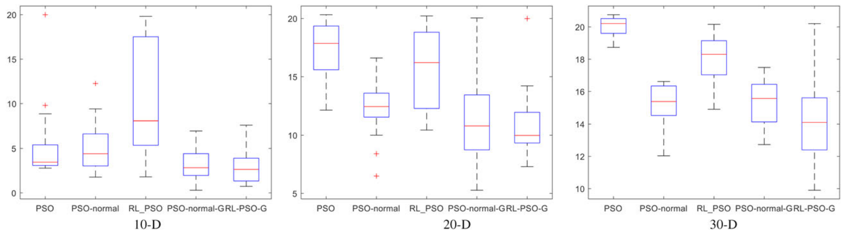

Figure 11. Boxplots of optimization results for the Ackley function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

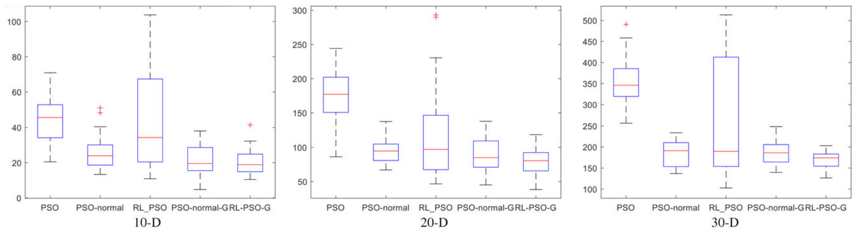

Figure 12. Boxplots of optimization results for the Rastrigin function for PSO, PSO-normal, RL-PSO, PSO-normal-G and RL-PSO-G.

preference of the selection. Additionally, this aggressive updating function is enhanced by learning the behaviour from history through reinforcement learning. By updating policy nets with historical behaviour, a preferred normal distribution can be found to accelerate the convergence speed of PSO. Thus, under the limitation of the available number of function evaluations, RL-PSO outperforms the original PSO. The aggressive updating strategy of RL-PSO, however, causes a convergence issue on high-dimensional problems. Therefore, to extend the exploitation of the RL-PSO algorithm, a normal distribution for the personal best term is changed back to a uniform distribution. As compared to PSO with normal distributions, PSO-normal and PSO-normal-G, reinforcement learning offers clear value and leads to stronger algorithms.

4.2. *Applying reinforcement learning in sDMS-PSO*

To test the performance of the proposed reinforcement learning strategy in different PSO algorithms, the proposed method is applied in sDSM-PSO and compared to the original algorithm using the CEC 2015 Learning-Based Benchmark Suite. sDMS-PSO is a modification of DMS-PSO, which uses the local best position (*lBest*) within a neighbourhood to replace the global best in the original PSO (Liang *et al.* 2015). The velocity update rule is calculated as follows:

$$v_{i+1} = w_i \times v_i + c_1 \times rnd \times (pBest - x_i) + c_2 \times rnd \times (lBest - x_i). \tag{24}$$

Instead of a fixed group of particles, sDMS-PSO regroups the particles frequently to form a dynamic population. The details of sDMS-PSO can be found in Liang *et al.* (2015) and the parameters of sDMS-PSO follow the default settings in that reference.

In this test, the uniformly distributed random number of the local best (which corresponds to the global best in the original PSO) is replaced by a normally distributed number generated from the reinforcement learning structure, as shown in Equation (25):

$$\begin{aligned} v_{i+1} &= w_i \times v_i + c_1 \times a_1 \times (pBest - x_i) + c_2 \times a_2 \times (lBest - x_i) \\ a_2 &\sim N(\mu_2, \sigma_2). \end{aligned} \tag{25}$$

Table 3. Comparison results for 10-D benchmark functions on sDMS-PSO.

Function	sDMS-PSO		RL-sDMS-PSO	
number	Mean	Std	Mean	Std
1	1.164E+04	8.35E+03	**9.531E+03**	**7.48E+03**
2	2.016E+04	**5.88E+03**	**1.817E+04**	6.09E+03
3	8.574E+00	**1.39E+00**	**8.450E+00**	1.84E+00
4	5.474E+02	**1.72E+02**	**5.371E+02**	2.48E+02
5	1.622E+00	5.09E-01	**1.581E+00**	**4.86E-01**
6	4.370E-01	**1.15E-01**	**3.840E-01**	1.35E-01
7	5.040E-01	2.44E-01	**4.795E-01**	**2.26E-01**
8	**4.672E+00**	**1.45E+00**	6.621E+00	3.49E+00
9	3.741E+00	**2.46E-01**	**3.486E+00**	4.45E-01
10	1.012E+05	9.78E+04	**6.098E+04**	**5.65E+04**
11	7.507E+00	2.20E+00	**6.952E+00**	**1.69E+00**
12	1.545E+02	7.16E+01	**1.418E+02**	**5.96E+01**
13	3.254E+02	3.93E+00	**3.236E+02**	**3.75E+00**
14	2.000E+02	**4.23E+00**	**1.970E+02**	4.65E+00
15	3.000E+02	1.85E+02	**2.682E+02**	**1.77E+02**

Note: Boldface numbers indicates the best performance between two algorithms.

Table 4. Comparison results for 30-D benchmark functions on sDMS-PSO.

Function	sDMS-PSO		RL-sDMS-PSO	
number	Mean	Std	Mean	Std
1	7.732E+09	2.50E+09	**7.687E+09**	**2.47E+09**
2	9.870E+04	**1.87E+04**	**8.549E+04**	2.10E+04
3	3.660E+01	2.47E+00	**3.587E+01**	**2.35E+00**
4	4.759E+03	6.01E+02	**4.498E+03**	**4.97E+02**
5	3.436E+00	8.53E-01	**3.434E+00**	**6.34E-01**
6	1.718E+00	7.61E-01	**1.640E+00**	**7.51E-01**
7	2.203E+01	**4.25E+00**	**1.677E+01**	4.29E+00
8	1.992E+05	2.27E+05	**1.557E+05**	**1.09E+05**
9	1.369E+01	2.47E-01	**1.366E+01**	**1.77E-01**
10	2.061E+07	1.16E+07	**1.801E+07**	**9.57E+06**
11	**1.188E+02**	**4.41E+01**	1.261E+02	6.10E+01
12	8.789E+02	2.62E+02	**7.591E+02**	**2.58E+02**
13	**4.558E+02**	**3.34E+01**	4.662E+02	5.59E+01
14	3.060E+02	2.99E+01	**3.045E+02**	**1.47E+01**
15	1.132E+03	2.51E+02	**1.100E+03**	**2.04E+02**

Note: Boldface numbers indicates the best performance between the two algorithms.

To illustrate the performance of the reinforcement learning strategy with limited computational budget, the maximum number of function evaluations of all the tests is set to be 1000. The tests are run 20 times and the results of 10-D and 30-D tests are shown in Tables 3 and 4. For 10-D problems, using reinforcement learning in sDMS-PSO leads to smaller objective values on 14 out of 15 test functions and smaller standard deviations in eight tests. For 30-D problems, sDMS-PSO including reinforcement learning performs better on 13 benchmarks. The limitation of the maximum number of function evaluations is the most difficult challenge of these tests. It can be seen that both algorithms may miss the global optimum under a tight computational budget. RL-sDMS-PSO, however, can find smaller objective values. Overall, the reinforcement learning strategy proposed in this work improves the performance of sDMS-PSO.

4.3. Power converter design problem

The power converter design problem has six design variables (Kott and Gabriele 1993), as shown in Table 5. The upper and lower bounds defined in D. Wang, G.G. Wang, and Naterer (2007) are used

Table 5. Design variables in power converter design.

Variable	Name	Description	Lower bound	Upper bound
x_1	C_w	Core centre leg width	0.001	0.1
x_2	*Turns*	Inductor turns	1.0	10
x_3	A_{cp}	Copper size	$7.29e^{-8}$	$1.0e^{-5}$
x_4	$L_f/PINDUC$	Inductance	$1.0e^{-6}$	$1.0e^{-5}$
x_5	C_f	Capacitance	$1.0e^{-5}$	0.01
x_6	w_w	Core window width	0.001	0.01

Table 6. Mean optimum comparison for PSO, RL-PSO, and RL-PSO-G for the power converter design problem.

No. of function	PSO		RL-PSO		RL-PSO-G	
evaluations	Mean	Std	Mean	Std	Mean	Std
100	0.358	0.191	1.020	**0.076**	**0.271**	0.216
200	0.178	0.116	1.010	**0.047**	**0.142**	0.066
300	0.128	0.078	1.006	**0.024**	**0.088**	0.060
400	0.090	0.054	0.999	**0.012**	**0.084**	0.067
500	0.098	0.060	0.998	0.039	**0.070**	**0.024**

Note: Boldface numbers indicates the best performance among the three algorithms.

in this article. The objective of the problem is to minimize the weight of the power converter. The formulations of the problem are defined as follows and all constant values are taken from D. Wang, G.G. Wang, and Naterer (2007):

$$\min y_1 = W_c + W_w + W_{\text{cap}} + W_{hs}, \tag{26}$$

where $W_c = |DIy_6(ZP_1 + y_7)|$, $ZP_1 = 2(1 + K_2)x_6$, $W_w = |(XMLT)(DC)x_2x_3|$, $XMLT = 2x_1(1 + K_1)FC$, $W_{\text{cap}} = |DK_5x_5|$ and $W_{hs} = |PO/KH((1/y_2)/ - 1)$.

Electrical design state analysis duty cycle:

$$y_3 = \frac{EO}{\left(y_2EI/2\,(XN)\right)}. \tag{27}$$

Minimum duty cycle:

$$y_4 = \frac{EO}{(y_2EIMAX/2(XN))}. \tag{28}$$

Inductor resistance:

$$y_5 = \frac{XMLTx_2(RO)}{x_3}. \tag{29}$$

Core cross-sectional area:

$$y_6 = K_1x_1^2. \tag{30}$$

Magnetic path length:

$$y_7 = \frac{\pi}{2}x_1. \tag{31}$$

Inductor value:

$$y_8 = \frac{(EO + VD)(1 - y_3)}{y_6x_2(FR)}. \tag{32}$$

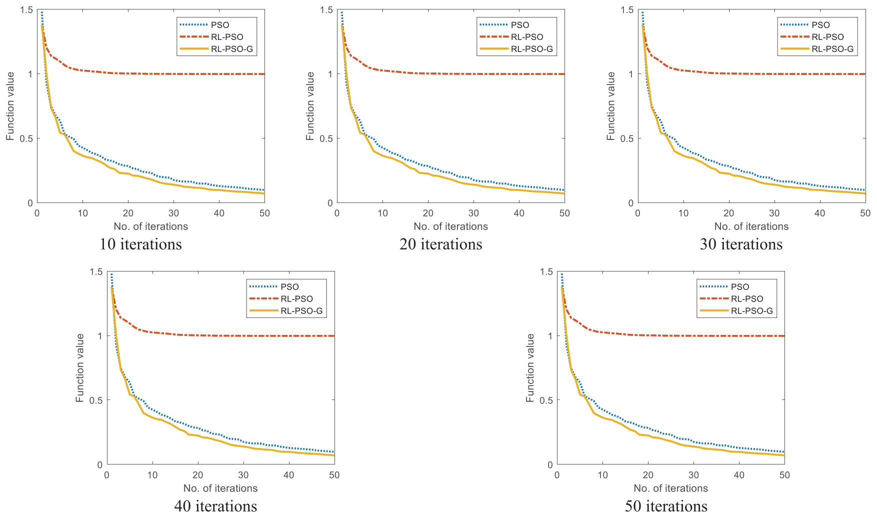

Figure 13. Convergence curves of PSO, RL-PSO and RL-PSO-G for the power converter design problem.

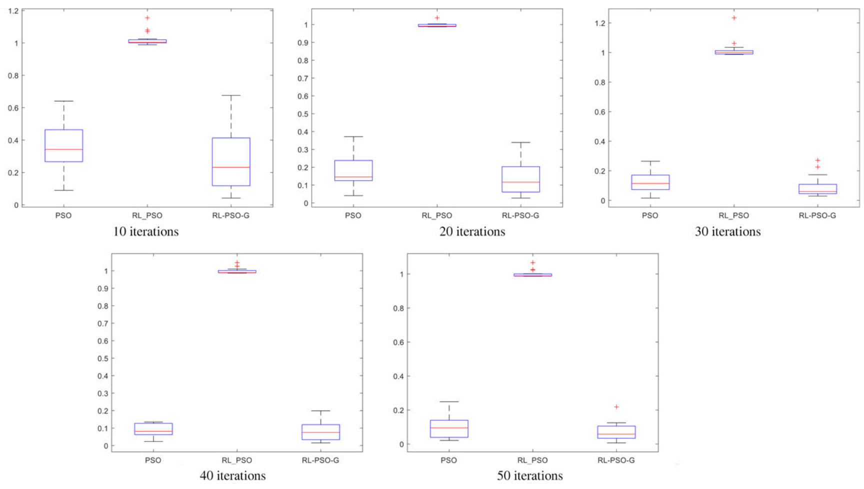

Figure 14. Boxplots of PSO, RL-PSO and RL-PSO-G for the power converter design problem.

Loss design state analysis:

$$y_2 = \frac{PO}{PQ + PD + POF + PXFR}. \tag{33}$$

PSO, RL-PSO and RL-PSO-G are used to solve the power converter design problem. The population size is set to be ten and the maximal number of iterations is set to be 10, 20, 30, 40 and 50. Thus, the maximal number of function evaluations is 100, 200, 300, 400 and 500, respectively, to test the performance of the proposed method with different computational costs. Note that the inertia weight (w in Equation 10) is related with the maximal number of iterations. Therefore, the tests for different numbers of function evaluations are performed independently. Each algorithm is run 20 times for each number of function evaluations and the results are shown in Table 6. The average convergence curves and the boxplots for the three algorithms are shown in Figures 13 and 14, respectively. As shown in Table 6, RL-PSO-G obtains the smallest optimal results among the three algorithms for all of the five scenarios. In particular, RL-PSO-G performs much better than the original PSO when the number of function evaluations is limited because RL-PSO-G is more aggressive in position updating. By employing the normal distribution and the historical information, RL-PSO-G can find a smaller solution with fewer iterations compared to the original PSO using a uniform distribution. The convergence curves in Figure 13 also illustrate the aggressive updating strategy in RL-PSO-G. With increasing numbers of function evaluations, the differences between PSO and RL-PSO-G become smaller, but RL-PSO-G remains better performing as compared to PSO. The boxplots in Figure 14 show that the robustness of RL-PSO-G is better than that of PSO overall. On the other hand, owing to the aggressive position updating strategy of RL-PSO, no better optimal solution can be found after an early stage of the optimization. Thus, as shown in Table 6, although the optimal results of RL-PSO improve with increasing numbers of function evaluations, the improvements are small. Figure 13 shows the convergence issue of RL-PSO. In Figure 14, even though the results of 20 repeated runs of RL-PSO show the smallest ranges among the three algorithms, they also manifest that the large number of iterations in RL-PSO cannot improve the optimal solution further.

5. Conclusion

In this article, reinforcement learning is employed in Particle Swarm Optimization (PSO) methods to tune the random coefficients in the PSO updating function automatically and adaptively. A normal distribution is formed to replace the uniform distribution for the random coefficients for each individual in the swarm. The mean and standard deviation of the normal distribution are predicted from a policy net according to the current state of the individual and the swarm group. The policy gradient method is used to update the policy net by learning from the previous states, *i.e.* promoting positive actions and punishing negative actions. The basic goal of reinforcement learning is to give more intelligence to each individual in the swarm group by learning from historical behaviours. The proposed reinforcement learning strategy is therefore expected to enhance not only the original PSO, but also other variants of PSO-based algorithms. This work tested the proposed method with the original PSO and a start-of-the-art PSO variant, sDSM-PSO. The test results show that employing the proposed strategy in different PSO-based algorithms can improve their respective performances under the constraint of a limited number of function evaluations. Finally, the proposed strategy with the original PSO is applied to a power converter design problem and the results show that the modified RL-PSO-G yields better optima with a higher convergence rate throughout the optimization process with different computational budgets.

For future work, the present authors will apply the reinforcement learning strategy to other swarm-intelligence based algorithms, as well as multi-objective optimization algorithms.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

Funding provided by Natural Sciences and Engineering Research Council (NSERC) of Canada is highly appreciated.

References

Arulkumaran, Kai, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. “Deep Reinforcement Learning: A Brief Survey.” *IEEE Signal Processing Magazine* 34 (6): 26–38. doi:10.1109/MSP.2017.2743240.

Beheshti, Zahra, and Siti Mariyam Hj Shamsuddin. 2013. “A Review of Population-Based Meta-Heuristic Algorithms.” *International Journal of Soft Computing and Its Applications* 5 (1): 1–35.

Bellman, R. 1952. “On the Theory of Dynamic Programming.” *Proceedings of the National Academy of Sciences* 38 (8): 716–719. doi:10.1073/pnas.38.8.716.

Chuang, Li Yeh, Hsiu Chen Huang, Ming Cheng Lin, and Cheng Hong Yang. 2011. "Particle Swarm Optimization with Reinforcement Learning for the Prediction of CpG Islands in the Human Genome." *PLoS ONE* 6 (6). doi:10.1371/journal.pone.0021036.

Clerc, Maurice, and James Kennedy. 2002. "The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space." *IEEE Transactions on Evolutionary Computation* 6 (1): 58–73. doi:10.1109/4235.985692.

Deisenroth, Marc Peter. 2011. "A Survey on Policy Search for Robotics." *Foundations and Trends in Robotics* 2 (1–2): 1–142. doi:10.1561/2300000021.

Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. 1996. "Ant System: Optimization by a Colony of Cooperating Agents." *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26 (1): 29–41. doi:10.1109/3477.484436.

Kadirkamanathan, Visakan, Kirusnapillai Selvarajah, and Peter J. Fleming. 2006. "Stability Analysis of the Particle Dynamics in Particle Swarm Optimizer." *IEEE Transactions on Evolutionary Computation* 10 (3): 245–255. doi:10.1109/TEVC.2005.857077.

Kaelbling, Leslie Pack, Michael L Littman, and Andrew W Moore. 1996. "Reinforcement Learning: A Survey." *Journal of Artificial Intelligence Research* 4: 237–285.

Karaboga, Dervis. 2005. "An Idea Based on Honey Bee Swarm for Numerical Optimization." [Technical Report-TR06]. Department of Computer Engineering, Engineering Faculty, Erciyes University.

Kennedy, James, and Russell C. Eberhart. 1995. "Particle Swarm Optimization." *Proceedings of the IEEE International Conference on Neural Networks* 4:1942–1948. doi:10.1109/ICNN.1995.488968.

Kennedy, James, and Rui Mendes. 2002. "Population Structure and Particle Swarm Performance." In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC* 2002, 1671–1676. Piscataway, NJ: IEEE Computer Society. doi:10.1109/CEC.2002.1004493.

Kiran, Mustafa Servet. 2017. "Particle Swarm Optimization with a New Update Mechanism." *Applied Soft Computing Journal* 60 (Nov.): 670–678. doi:10.1016/j.asoc.2017.07.050.

Kott, G., and G. A. Gabriele. 1993. "Application of Multidisciplinary Design Optimization to the Power Stage Design of A Power Converter." *ASME, Advances in Design Automation* 2: 359–366.

Koziel, Slawomir, and Xin-She Yang. 2011. *Computational Optimization, Methods and Algorithms*. Vol. 356. Berlin: Springer.

Li, Ke, and Jitendra Malik. 2017. "Learning to Optimize Neural Nets." *arXiv Preprint arXiv:1703.00441*.

Liang, J. J., L. Guo, R. Liu, and B. Y. Qu. 2015. "A Self-Adaptive Dynamic Particle Swarm Optimizer." In *2015 IEEE Congress on Evolutionary Computation, CEC 2015 – Proceedings*, 3206–3213. Piscataway, NJ: IEEE. doi:10.1109/CEC.2015.7257290.

Liang, J. J., B. Y. Qu, P. N. Suganthan, and Q. Chen. 2014. "Problem Definitions and Evaluation Criteria for the CEC 2015 Competition on Learning-Based Real-Parameter Single Objective Optimization." Technical Report201411A, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore.

Mendes, Rui. 2006. "Neighborhood Topologies in Fully Informed and Best-of-Neighborhood Particle Swarms." *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 36 (4): 515–519. doi:10.1109/TSMCC.2006.875410.

Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis. 2014. "Grey Wolf Optimizer." *Advances in Engineering Software* 69 (Mar.): 46–61. doi:10.1016/j.advengsoft.2013.12.007.

Misir, Mustafa, Tony Wauters, Katja Verbeeck, and Greet Vanden Berghe. 2009. "A New Learning Hyper-Heuristic for the Traveling Tournament Problem." Paper presented at the 8th Metaheuristic International Conference (MIC'09), Hamburg, Germany, July 13–16.

Piperagkas, Grigoris S., George Georgoulas, Konstantinos E. Parsopoulos, Chrysostomos D. Stylios, and Aristidis C. Likas. 2012. "Integrating Particle Swarm Optimization with Reinforcement Learning in Noisy Problems." In *GECCO'12 – Proceedings of the 14th International Conference on Genetic and Evolutionary Computation*, 65–72. New York: ACM Press. doi:10.1145/2330163.2330173.

Ratnaweera, Asanga, Saman K. Halgamuge, and Harry C. Watson. 2004. "Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients." *IEEE Transactions on Evolutionary Computation* 8 (3): 240–255. doi:10.1109/TEVC.2004.826071.

Rezaei, Farshad, and Hamid R. Safavi. 2020. "GuASPSO: A New Approach to Hold a Better Exploration–Exploitation Balance in PSO Algorithm." *Soft Computing* 24 (7): 4855–4875. doi:10.1007/s00500-019-04240-8.

Samma, Hussein, Chee Peng Lim, and Junita Mohamad Saleh. 2016. "A New Reinforcement Learning-Based Memetic Particle Swarm Optimizer." *Applied Soft Computing Journal* 43 (Jun.): 276–297. doi:10.1016/j.asoc.2016.01.006.

Samma, Hussein, Junita Mohamad-Saleh, Shahrel Azmin Suandi, and Badr Lahasan. 2020. "Q-Learning-Based Simulated Annealing Algorithm for Constrained Engineering Design Problems." *Neural Computing and Applications* 32 (9): 5147–5161. doi:10.1007/s00521-019-04008-z.

Shi, Yuhui, and Russell C. Eberhart. 1998. "A Modified Particle Swarm Optimizer." In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 69–73. Piscataway, NJ: IEEE. doi:10.1109/ICEC.1998.699146.

Shi, Yuhui, and Russell C. Eberhart. 1999. “Empirical Study of Particle Swarm Optimization.” In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999*, 3:1945–1950. Piscataway, NJ: IEEE Computer Society. doi:10.1109/CEC.1999.785511.

Sutton, Richard S, and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning*. Vol. 135. Cambridge: MIT press.

Tian, Dongping, Xiaofei Zhao, and Zhongzhi Shi. 2019. “DMPSO: Diversity-Guided Multi-Mutation Particle Swarm Optimizer.” *IEEE Access*. 7: 124008–124025. doi:10.1109/ACCESS.2019.2938063.

Trelea, Ioan Cristian. 2003. “The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection.” *Information Processing Letters* 85 (6): 317–325. doi:10.1016/S0020-0190(02)00447-7.

Wang, Dapeng, G. Gary Wang, and Greg Naterer. 2007. “Collaboration Pursuing Method for Multidisciplinary Design Optimization Problems.” *AIAA Journal* 45 (5): 1091–1103. doi:10.2514/6.2005-2204.

Wauters, Tony, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. 2013. “Boosting Metaheuristic Search Using Reinforcement Learning.” *Studies in Computational Intelligence* 434: 433–452. doi:10.1007/978-3-642-30671-6_17.

Williams, Ronald J. 1992. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” *Machine Learning* 8 (3–4): 229–256. doi:10.1007/bf00992696.

Xu, Yue, and Dechang Pi. 2020. “A Reinforcement Learning-Based Communication Topology in Particle Swarm Optimization.” *Neural Computing and Applications* 32 (14): 10007–10032. doi:10.1007/s00521-019-04527-9.

Xu, Xia, Yinggan Tang, Junpeng Li, Changchun Hua, and Xinping Guan. 2015. “Dynamic Multi-Swarm Particle Swarm Optimizer with Cooperative Learning Strategy.” *Applied Soft Computing Journal* 29 (Apr.): 169–183. doi:10.1016/j.asoc.2014.12.026.