Applied Soft Computing 111 (2021) 107678

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Differential evolution with mixed mutation strategy based on deep reinforcement learning

Zhiping Tan [a,*], Kangshun Li [b]

[a] College of Electronics and Information, Guangdong Polytechnic Normal University, Guangzhou 510665, China
[b] College of Mathematics and Informatics, South China Agricultural University, Guangzhou 510642, China

ARTICLE INFO

Article history:
Received 7 October 2020
Received in revised form 17 June 2021
Accepted 28 June 2021
Available online 7 July 2021

Keywords:
Differential evolution
Mixed mutation strategy
Fitness landscape
Deep reinforcement learning
Deep Q-learning

ABSTRACT

The performance of differential evolution (DE) algorithm significantly depends on mutation strategy. However, there are six commonly used mutation strategies in DE. It is difficult to select a reasonable mutation strategy in solving the different real-life optimization problems. In general, the selection of the most appropriate mutation strategy is based on personal experience. To address this problem, a mixed mutation strategy DE algorithm based on deep Q-network (DQN), named DEDQN is proposed in this paper, in which a deep reinforcement learning approach realizes the adaptive selection of mutation strategy in the evolution process. Two steps are needed for the application of DQN to DE. First, the DQN is trained offline through collecting the data about fitness landscape and the benefit (reward) of applying each mutation strategy during multiple runs of DEDQN tackling the training functions. Second, the mutation strategy is predicted by the trained DQN at each generation according to the fitness landscape of every test function. Besides, a historical memory parameter adaptation mechanism is also utilized to improve the DEDQN. The performance of the DEDQN algorithm is evaluated by the CEC2017 benchmark function set, and five state-of-the-art DE algorithms are compared with the DEDQN in the experiments. The experimental results indicate the competitive performance of the proposed algorithm.

© 2021 Elsevier B.V. All rights reserved.

1. Introduction

Numerical optimization has attracted considerable attention from scientific researchers and is extensively employed in engineering design [1], task scheduling [2], and robot path planning [3]. However, optimization problems have various unique characteristics and complicated mathematical proprieties. For example, the solutions of these problems can be discrete, continuous, or a mix of both, and the fitness function can be unimodal or multimodal and a hybrid or composite function [4]. Due to these different characteristics, the optimal solution is difficult to be solved by traditional methods. In the past decades, researchers have proposed a lot of evolutionary algorithms to solve numerical optimization problems, such as cuckoo search (CS) [5], firefly algorithm (FA) [6], artificial bee colony (ABC) [7], whale optimization algorithm (WOA) [8] and so on. Although these algorithms can better solve most of the optimization problems, they still exist lower accuracy and more computation time. At the same time, with the increasing complexity of optimization problems, the studies of the optimization methods have become popular topics [9].

Differential evolution, a population-based evolutionary algorithm, was proposed by Storn and Price [10]. Because of its simplicity, easy implementation, and stronger robustness and search ability. DE has been now widely applied to solve various optimization problems from various fields, such as numerical optimization [11], image thresholding segmentation [12], filter design [13], and problem prediction [14]. However, DE is sensitive to the control parameters and mutation strategy. Generally, the most appropriate mutation strategy and parameter settings for different optimization problems are often different. Some mutation strategies are suitable for solving unimodal problems, while some others are effective for multimodal problems. Some parameter settings can maintain the population diversity and enhance global exploration, while some other settings focus on enhancing convergence speed [15]. Therefore, the trial-and-error procedure for setting suitable control parameters and mutation strategy is often time-consuming and tedious. For this reason that some literature summarized some useful rules for the above-mentioned problems, but their universality needs to be further enhanced. To address these problems, researchers have proposed many enhanced DE algorithms such as FADE (with fuzzy self-adapted parameters) [16], chDE (with a chaos-based adaptive parameters) [17], jDE (with self-adapted parameters) [18], JADE

* Corresponding author.
E-mail address: tzp2008ok@163.com (Z. Tan).

https://doi.org/10.1016/j.asoc.2021.107678
1568-4946/© 2021 Elsevier B.V. All rights reserved.

(with "current-to-*p*best/1" mutation strategy and adaptive parameters) [19], SHADE (with historical memory parameters adaptive method) [20], LSHADE (with linear population size reduction) [21], EBLSHADE (with "current-to-ord_*p*best/1" mutation strategy ") [22]. LSHADE-EpSin (with an ensemble sinusoidal-based parameters adaption) [23] and LSHADE-RSP (with rank-based selective pressure strategy) [24].

Furthermore, DE has several commonly used mutation strategies such as "DE/rand/1", "DE/best/1", "DE/current-to-best/1", "DE/rand/2", "DE/best/2", and "DE/current-to-rand/1" [25]. "DE/rand/1", "DE/best/1" and "DE/best/2" usually have stronger local exploitation capability and faster convergence speed. But, they are easy to fall into local optimum. DE/rand/2, DE/current-to-rand/1 and DE/current-to-best/1 bear stronger global exploration capability. However, the convergence rate is very slow. Instead of only utilizing a single mutation strategy, some DE algorithms mainly focus on the combination of different mutation strategies. For instance, SaDE (with adapted mutation strategies and parameters) [26], CoDE (composition of multiple strategies and parameter settings) [27], EPSDE(with an ensemble of mutation strategies and parameters) [28], MPEDE(with multiple population-based ensemble of mutation strategies) [29], DMPSADE (with discrete mutation control parameters) [30] and IMSaDE (with elite archive strategy) [31] and FLDE (with mutation strategy selection based on the random forest) [32]. Through the survey and analysis of these literature, we know that the new mutation operators, multiple strategies, strategy archive and so on are proposed to improve the optimization performance of the DE for solving optimization problems, all the algorithms are only concerned with the algorithm itself and propose various adaptive mutation strategy methods. However, it is still easy to fall into the local optimum. Virtually, the population of DE may evolve through different regions in the search space, within which different strategy selection may be more effective than others. To the best of our knowledge, none of the mutation strategy selections that uses offline training is based on reinforcement learning and the fitness landscape of a problem, although it may help improve the performance of the algorithm.

In this paper, a mixed mutation strategy DE algorithm based on deep Q-network (DQN), named DEDQN is proposed, in which a deep reinforcement learning (RL) method that uses a deep neural network as a prediction model, as an adaptive mutation operator method for DE. In the DEDQN, the DQN is trained offline according to the fitness landscape of the problem. After this training phase, the DEDQN algorithm can be applied to unseen problems. It will analyze the fitness landscape of the problem and predict which mutation strategy should be used at each generation. Furthermore, to improve the performance of the proposed algorithm, a historical memory parameter adaption mechanism is also utilized. In the experiments, the DEDQN is tested on the suit of CEC2017 benchmark functions with 10, 30, 50, and 100 dimensions, respectively. The competitive performance of DEDQN is verified by extensive comparisons with several well-known DE variants.

The remainder of the paper is organized as follows: In Section 2, we introduce the DE algorithm and its related work. The fitness landscape analysis methods are described in Section 3. The framework of the proposed algorithm is shown in Section 4. The experimental results and analysis are provided in Section 5. Section 6 summarizes the conclusions and makes a prospect for future work.

2. Differential evolution and its related work

2.1. Differential evolution

The standard DE consists of mutation, crossover, and selection operators. The three operators may contribute to effectively understand the improved DE. DE is also a population-based optimization algorithm. Assume that a population contains NP individuals. Each individual is a D-dimensional vector, which can be expressed as:

$$x_{i,G} = \left\{x_{i,G}^1, x_{i,G}^2, \ldots, x_{i,G}^D\right\}, \ i = 1, \ldots, NP \tag{1}$$

where G represents the generation. For a bounded optimization problem, each individual should be limited by:

$$x_{\min} = \left\{x_{\min}^1, \ldots, x_{\min}^D\right\} \ and \, x_{\max} = \left\{x_{\max}^1, \ldots, x_{\max}^D\right\} \tag{2}$$

Therefore, the initial value of the j-th decision variable of the i-th population individual at generation $G = 0$ can be generated by:

$$x_{i,0}^j = x_{\min}^j + rand\,(0, 1) * \left(x_{\max}^j - x_{\min}^j\right) \tag{3}$$

where $rand(0, 1)$ represents a uniformly distributed random number within the range [0,1].

Mutation operation: After initialization, each individual $x_{i,G}$ in the current population through the mutation operator generates the mutant vectors $v_{i,G}$. Generally, there are six widely used mutation strategies, which are summarized as follows:

•"DE/rand/1"

$$v_{i,G} = x_{r1,G} + F * \left(x_{r2,G} - x_{r3,G}\right) \tag{4}$$

•"DE/best/1"

$$v_{i,G} = x_{best,G} + F * \left(x_{r1,G} - x_{r2,G}\right) \tag{5}$$

•"DE/rand/2"

$$v_{i,G} = x_{r1,G} + F * \left(x_{r2,G} - x_{r3,G}\right) + F * \left(x_{r4,G} - x_{r5,G}\right) \tag{6}$$

•"DE/best/2"

$$v_{i,G} = x_{best,G} + F * \left(x_{r1,G} - x_{r2,G}\right) + F * \left(x_{r3,G} - x_{r4,G}\right) \tag{7}$$

•"DE/current-to-rand/1"

$$v_{i,G} = x_{i,G} + F * \left(x_{r1,G} - x_{i,G}\right) + F * \left(x_{r2,G} - x_{r3,G}\right) \tag{8}$$

•"DE/current-to-best/1"

$$v_{i,G} = x_{i,G} + F * \left(x_{best,G} - x_{i,G}\right) + F * \left(x_{r1,G} - x_{r2,G}\right) \tag{9}$$

where the indices $r_1, \ldots, r_5$ are different integers randomly generated within the range $[1, NP]$, as well the associated index differs from the index i. These indices are randomly produced once at each generation. F is a scale factor within the range [0,1]. $x_{best,G}$ is the best individual in the current population at generation G.

Crossover operation: The binomial crossover operator is applied to select the trial vector $u_{i,G}$ between $v_{i,G}$ and $x_{i,G}$. According to the crossover strategy, the trial vector is generated by the formula below:

$$u_{i,G}^j = \begin{cases} v_{i,G}^j, & if \ \left(rand_j\,[0, 1] \le CR \, or \, j = j_{rand}\right) \\ x_{i,G}^j, & otherwise \end{cases} \quad j = 1, 2, \ldots, D \tag{10}$$

where $rand_j$ is a uniformly distributed random variable, which is generated within the range [0,1]. $CR \in [0,1]$ is the crossover rate, which is a user-specified constant, and j_{rand} is a random integer chosen in the range $[1,D]$.

Selection operation: After mutation and crossover operations, the selection operation is executed to choose the best individuals according to the fitness values of the trial individuals. For a minimization optimization problem, if the fitness value of the trial individual $f(u_{i,G})$ is less than or equal to the corresponding target individual $f(x_{i,G})$, the trial individual will substitute the target individual and update the population. Otherwise, the target individual will keep in the population for the next generation G+1. This greedy selection operation is performed by the below formula:

$$x_{i,G+1} = \begin{cases} u_{i,G}, & if\, f\left(u_{i,G}\right) \leq f\left(x_{i,G}\right) \\ x_{i,G}, & otherwise \end{cases} \tag{11}$$

2.2. Related work

The mutation strategy and the associated parameters can significantly affect the performance of DE on different types of optimization problems. Therefore, to successfully solve a specific optimization problem, it is generally necessary to perform a time-consuming trial-and-error search for the most appropriate strategy and to tune its associated parameter values. However, such a trial-and-error search process sustains from high computational costs. To tackle this problem, researchers have proposed a lot of improved DE algorithms. Among them, various methods have been suggested for selecting mutation strategies and their associated control parameter settings according to the characteristics of the problem [33]. In this section, some techniques related to the selection of mutation strategy are discussed.

DE has many mutation strategies, each of which has its advantages and disadvantages. Therefore, some improved algorithms pay attention to the ensemble or improvement of mutation strategies. Qin et al. designed a self-adaptive DE (SaDE). In SaDE, at first, four mutation strategies are stored in a strategy pool, each strategy is randomly selected for every individual at each generation. Before long, some promising strategies are picked up into the strategy pool according to the probability of successful strategy producing better offspring during the evolutionary process [26]. Wang et al. proposed a composite DE (CoDE), in which a mutation strategy pool is constituted by adopting three mutation operators. In each generation, the new trial vector originated from the best one is generated by three mutation strategies at the same time [27]. A self-adaptive multi-operator based differential evolution (SAMO-DE) was conceived by Elsayed et al. In SAMO-DE, each search operator has its sub-population. In each generation, the new selected operator relies on the best-performing search operator [34]. Mallipeddi et al. presented an ensemble differential evolution algorithm (EPSDE). In EPSDE, some strategy pools are designed to store the mutation operators and the control parameters. In the initial population, each individual is randomly assigned one strategy from the strategy pool and the associated parameter from the parameter pool. Once the offspring cannot outperform the corresponding parent, a new mutation strategy and parameter values will be randomly chosen from the respective pool [28]. Elsayed et al. proposed a united multi-operator EAs (UMOEAs) technique, in which the population is divided into three sub-populations. Then, each sub-population uses a different mutation operator to generate the mutant vector, and the best-performing mutation strategy is determined by its cumulative probability. In the end, the worst-performing population individuals will be updated by the best-performing one [35]. A multiple operator's DE variant named MPEDE, was proposed by Wu et al. In MPEDE, a multiple population strategy is adapted. Its initial population is grouped into three sub-populations, and three different mutation strategies are assigned to each sub-populations. During the evolutionary process, the best-performing mutation strategy is selected by evaluation indicators and rewards [29]. Zhou et al. proposed an underestimation-based multiple mutation strategies for DE, named UMS-SHADE. In UMS-SHADE, a set of candidate offsprings are generated by utilizing multilevel mutation strategies, the most promising offspring is chosen by the underestimation value [36]. Wang et al. presented an IMSaDE algorithm, in which an improved "DE/rand/2" mutation strategy that utilizes an elite archive strategy to improve the convergence is proposed. In the mutation operation, five random individuals are not chosen blindly from the current population, but purposefully select from the divided population [31]. Li et al. proposed a differential evolution variant with multi-population cooperation and multi-strategy integration (MPMSDE). In MPMSDE, a new mutation strategy "DE/pbad-to-pbest-to-gbest/1" is used to replaced "DE/rand/1" in MPEDE. Besides, a new grouping method is used to improve the MPMSDE [37]. Tan et al. proposed a fitness landscape-based DE algorithm called FLED. In FLDE, a random forest is used to established the relationship between three mutation strategies and the fitness landscape of each training function. During the evolutionary process, the suitable mutation strategy will be predicted by the trained random forest depending on the fitness landscape of the problem [32].

3. Fitness landscape analysis

Fitness landscape theory was proposed by Wright and has been proven to be an effective method to analyze optimization problems [38]. The study of the fitness landscape is an important topic of evolutionary computation. Influenced by biological evolution, researchers began to study the fitness landscape early in the field of evolutionary computation, to understand how evolutionary algorithms behave and to improve the solutions of optimization problems. The fitness landscape can reveal the relationship between the search solution space and individual fitness using features of the landscape information. Several fitness landscape metrics have been developed for analyzing and evaluating the different characteristics of a problem, such as the fitness distance correlation, ruggedness of information entropy, auto-correlation function, and the number of optimal values.

3.1. Fitness distance correlation

The concept of fitness distance correlation (FDC) was proposed by Jones for the first time and is applied to measure the complexity of the optimization problem. Jones' approach suggests that the difficulty of a problem can be evaluated by the relationship between fitness value and the distance of the solution from the optimum. This relationship can be concluded by computing the FDC coefficient, which is the correlation coefficient between the fitness value and the distance to the nearest global optimum for all solutions within the search space. Given a sample of m solutions and the associated fitness value for each solution compose a sequence F={f_1, f_2, ... , f_m}. Hence, the fitness values in the sample are determined. The Euclidean distance D={d_1, d_2, ... , d_m} is computed from each solution to the optimal solution in the sample. Thus, FDC is defined as [39]:

$$FDC = \frac{C_{FD}}{\delta_F \delta_D} \tag{12}$$

where δ_F and δ_D are the variances of F and D, respectively. C_{FD} is the covariance of F and D, which can be calculated by:

$$C_{FD} = \frac{1}{m}\sum_{i=1}^{m}\left(f_i - \bar{f}\right)\left(d_i - \bar{d}\right) \tag{13}$$

where $\bar{f}$ and $\bar{d}$ are the average values of F and D, respectively.

3.2. Ruggedness of information entropy

The ruggedness of information entropy (RIE) is related to the number and distribution of the local optimal solutions. Malan et al. proposed a fitness landscape measurement method based on an information entropy theory [40]. A random walk strategy is used to explore the characteristic of the fitness landscape. Assume that a random walk is performed to obtain a series of fitness value s$\{f_1, f_2, \ldots, f_n\}$. A string $S(\epsilon)$=$s_1, s_2, \ldots, s_i, s_n$ of symbols $s_i \in \{\bar{1}, 0, 1\}$ can be calculated by the following principle:

$$S_i(\varepsilon) = \begin{cases} \bar{1}, & if\ f_i - f_{i-1} < -\varepsilon \\ 0, & if\ |f_i - f_{i-1}| \le \varepsilon \\ 1, & if\ f_i - f_{i-1} > \varepsilon \end{cases} \tag{14}$$

where the accuracy of the calculation of string $S(\varepsilon)$ is determined by the parameter ε.

According to the definition of the string $S(\varepsilon)$, the calculation of entropy $H(\varepsilon)$ is presented in Eq. (15):

$$H(\varepsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 P_{[pq]} \tag{15}$$

where p and q are elements from the set $\{\bar{1}, 0, 1\}$, and $p \neq q$, the total number of different combination $[pq]$ is equal to 6. The probability $P_{[pq]}$ can be calculated by:

$$P_{[pq]} = \frac{n_{[pq]}}{n} \tag{16}$$

where $n_{[pq]}$ is the number of the different combinations $[pq]$ in the string $S(\varepsilon)$.

The string $S(\varepsilon)$ is sensitive to the size of parameter ε. By increasing the value of ε, the entropy $H(\varepsilon)$ will become larger, it indicates the landscape simpler. When the value of ε takes the maximum, the string $S(\varepsilon)$ is the collection of 0, which is called the information stability and is also represented by ε^*. Hence, the value of ε^* would be equal to the largest difference in fitness values. The ε^* can be calculated by:

$$\varepsilon^* = \max \{f_i - f_{i-1}\}_{i=2}^{n} \tag{17}$$

To depict the ruggedness of a fitness landscape more intuitively, a single value R_f is used to measure the ruggedness of information entropy:

$$R_f = \max_{\forall \varepsilon \in [0, \varepsilon^*]} \{H(\varepsilon)\} \tag{18}$$

All ϵ in the set

$$\left\{\varepsilon | \varepsilon = 0, \frac{\varepsilon^*}{128}, \frac{\varepsilon^*}{64}, \frac{\varepsilon^*}{32}, \frac{\varepsilon^*}{16}, \frac{\varepsilon^*}{8}, \frac{\varepsilon^*}{4}, \frac{\varepsilon^*}{2}, \varepsilon^*\right\} \tag{19}$$

Therefore, the higher the value of R_f tends to have a more rugged landscape, while the value of R_f for smooth landscape is small. The value of R_f captures the ruggedness, smoothness, and neutrality of landscapes. The ruggedness of information entropy is an effective measure index to estimate the complexity of the problem.

3.3. Auto-correlation function

The auto-correlation function was also proposed to evaluate the ruggedness of a landscape [41]. If a low auto-correlation between two sets of fitness points separated by some distance, these fitness points are different, it suggests the landscape is more rugged. On the contrary, the landscape is smoother. Given a sequence of fitness values $\{f_1, f_2, \ldots, f_j, f_n\}$, which is gained by a random walk. The following formulas are used to calculate the fitness landscape statistical measure. The auto-correlation function $r(s)$ between sets of fitness points separated with s solutions is computed by:

$$r(s) = \frac{\sum_{j=1}^{n-s} \left(f_j - \bar{f}\right)\left(f_{j+s} - \bar{f}\right)}{\sum_{j=1}^{n} \left(f_j - \bar{f}\right)^2} \tag{20}$$

where $\bar{f}$ represents the average value of sequence f_j.The auto-correlation coefficient τ is defined as:

$$\tau = r(1) \tag{21}$$

3.4. The number of optimal values

A simple method to measure the number of optimal values in the fitness landscape is based on the achievements of Sheng et al. [42]. Firstly, for a population $(x_1, x_2, \ldots, x_\mu)$, find out the best individual among the whole population, and tagged it with x_{best}. Then calculate the Euclidean distance between each individual x_i $(i = 1, \cdots, \mu)$ and x_{best}, which can be expressed as:

$$d_i = \sum_{j=1}^{n} \left(x_{i_j} - x_{best_j}\right)^{\frac{1}{2}} \tag{22}$$

Secondly, sort the individuals according to the distance value, resulting in the following in ascending order: $k_1, k_2, \ldots, k_\mu$. After that, calculate the local fitness landscape evaluation metric of the individual. The measure metric value is denoted by χ. Suppose that the initial value is 0. In the end, the value will be increased to 1, if $f_{k_{i+1}} \le f_{k_i}$. Normalize the value as:

$$\varphi = \frac{\chi}{\mu} \tag{23}$$

On account of the fitness landscape is actual an ambiguous concept, the landscape feature φ can be deemed as the roughness of the observed fitness landscape, and judge the complexity of the optimization problem. If the value of $\varphi = 0$, then fitness landscape is more like a unimodal landscape, it means that the optimization problem is easy to solve. On the contrary, if the value of $\varphi = 1$, it indicates that the local fitness landscape is very rough.

3.5. The calculation of the fitness landscape

The global fitness landscape of an optimization problem is difficult to calculate. Usually, we calculate the local fitness landscape by a random walk sampling algorithm, which could be described as follows: Firstly, randomly generating a landscape point in the domain space, whose neighbor points are produced by a random walk strategy. Then, determining whether the generated neighbor points are within the boundary of a problem, if they are not in the boundary, the new neighborhood points will be regenerated. Finally, the random walk sequence is obtained. The corresponding pseudo-code is given in Table 1.

4. Proposed algorithm

In this section, we describe the state features, action space, and reward definition for the DQN, and introduce the training process of DQN, and parameter adaption strategy for DEDQN. Finally, the framework of the DEDQN is described in detail.

4.1. The related settings of DQN

In reinforcement learning (RL), an agent performs an action in an environment then obtains a reward and the next state [43]. The purpose is to maximize the cumulative reward after executing each action operation. RL estimates the value of an action given a state called Q-value to learn a policy that returns an action given a state. After, an agent performs an action a in the state s, according to the optimal strategy π, the action function $Q(s, a)$ is composed of the instant reward $R(s, a)$ and the maximum long-term cumulative reward $Q(s', a')$ in the next state s' and action a', which can be expressed as :

$$Q(s, a) = R(s, a) + \gamma \max Q\left(s', a'\right) \tag{24}$$

Table 1
Random walk algorithm.

```
Random walk algorithm
Input: the population (P) and the dimension (D) of the problem.
1. Initialize the minimum and maximum sequence value $P_{\min}$, $P_{\max}$, respectively, and the steps size (size);
2. for $i$=1,2,...,$D$ do
3.     $P^i_{\min}$ =the minimum value of $i$th for all the individual in the population (P);
4.     $P^i_{\max}$ =the maximum value of $i$th for all the individual in the population (P);
5.     $size_i = P^i_{\max} - P^i_{\min}$;
6. end for
7. Obtain the domain=($P_{\min}$, $P_{\max}$), the steps size (size);
8. Define the walk steps (steps);
9. Initialize $count = 1$, $walk=\phi$;
10. Generate a random number for walk[1];
11. while $count < steps$ do
12.     for $i$=1,2,..,$D$ do
13.         Generate a random number (step) in the range [0, $size_i$];
14.         Set $walk[count+1]_i = walk[count]_i + step$;
15.         if $walk[count+1]_i$ > the maximum bound of the space then
16.             Set $walk[count+1]_i = walk[count]_i -$(range of the current domain);
17.         end if
18.     end for
19. $count = count+1$;
20. end
21. Obtain the random walk sequence.
22. Analyze the local fitness landscape.
Output: The fitness landscape feature values.
```

where γ is the discount factor.

In the iteration process, the agent will update the $Q(s, a)$ constantly, the update formula can be described as:

$$Q(s, a) = (1-\alpha) Q(s, a) + \alpha \left(R(s, a) + \gamma \max Q\left(s', a'\right)\right) \tag{25}$$

where α is a constant within [0,1] used to control the convergence rate.

Later, a lot of different RL methods are used a Q table to store the states and actions, then the value function is calculated by searching for the Q table, but when the state is continuous or the set of states is large, therefore, it will take a lot of time to find and store the Q table. To address this problem, the deep reinforcement learning technique was proposed. In deep RL, this value function is replaced by a deep neural network. DQN is a deep RL technique that extends Q-learning to continuous features by approximating a non-linear Q-value function of the state features using a neural network.

Reinforcement learning is a process in which the agent interacts with the environment. Through continuous training, the agent finally learns to make decisions. In the proposed algorithm, through training the DQN, it becomes an agent capable of autonomously selecting mutation strategies based on environmental observations. The environment is composed of DEDQN algorithm, fitness function, and fitness landscape calculation module, which is shown in Fig. 1. It can also find that DQN as an agent needs to define state space, action space, and reward value. In the following, we will explain them in detail.

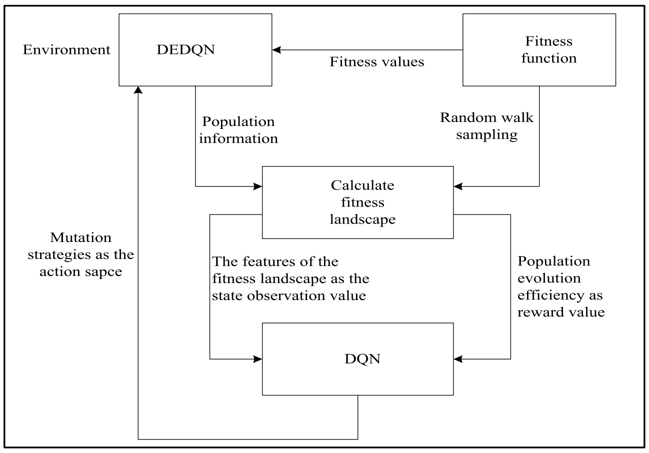
Environment
DEDQN
Fitness values
Fitness function
Population information
Random walk sampling
Calculate fitness landscape
Mutation strategies as the action sapce
The features of the fitness landscape as the state observation value
Population evolution efficiency as reward value
DQN

Fig. 1. The working environment of DQN.

State representation: The environment state representation can provide sufficient information to guide the agent to select a more suitable mutation strategy at each step. A state vector composed of four fitness landscape features, which can be expressed as:

$$state = \{FDC, RIE, r(1), NNUM\} \tag{26}$$

Action space: Mutation strategy is important for DE. In this work, we choose three mutation strategies such as "DE/rand/1", "DE/current to rand/1" and "DE/best/2". "DE/current to rand/1" can be used for global search, "DE/rand/1" is utilized for local search. However, "DE/best/2" can make the algorithm convergence rapidly. Therefore, the action space consists of the three mutation strategies, which can be expressed as follow:

$$Action_{mu} = \{rand/1, current\ to\ rand/1, best/2\} \tag{27}$$

Reward definition: In RL, the value function is used to guide the agent to make decisions according to the state information. That is, the agent measures whether its actions are reasonable according to the return value of the reward function. The higher the reward value, the more reasonable the agent's actions in the previous state. Evolutionary ability is a concept in dynamic fitness landscape analysis, which refers to the ability of a population to produce better offspring. In this paper, population evolutionary efficiency is presented as a measure index of evolutionary ability, and is used as the reward function value of the agent.

Defining the live algebraic $survival_i$ which represents the individual x_i from population **P** survival from the first generation to the current generation, the $survival_i$ can be described as follow:

$$survival_i = \begin{cases} 1, & if\ f(u_i) \le f(x_i) \\ survival_i + 1, & otherwise \end{cases} \tag{28}$$

And defining the individual evolutionary efficiency e_i, which can be expressed as:

$$e_i = \begin{cases} survival_i^{-1}, & if\ f(u_i) \le f(x_i) \\ 0, & otherwise \end{cases} \tag{29}$$

Then the evolutionary efficiency of the population can be expressed as:

$$eep(P) = \frac{\sum_{i=1}^{NP} e_i}{NP} \tag{30}$$

where NP represents the population size.

4.2. Training phase

The purpose of the training phase is to train the deep neural network of the DQN so that it learns to approximate the target $\hat{Q}$ function. The training data is a memory of observations that are collected by running DEDQN several times on training benchmark functions. In the experiment, a total of 45 benchmark functions originated from CEC2014 and CEC2015 are used to train the DQN. Training the deep neural network involves finding its weights θ through gradient descent.

The DQN training algorithm is shown in Table 2. Firstly, the parameters of DE algorithm and DQN algorithm are initialized, and calculating the features of the fitness landscape as s_t. When DE is executed T times, and each run is stopped after $FES^{\max}$ function evaluations (line 6). For each solution in the population, a mutation strategy is selected randomly with ϵ probability, if not the mutation strategy with maximum Q value is selected. Then, performing the mutation,crossover and selection operation (line 15 and line 16). Using the current fitness landscape represents state s_t, the deep neural network is responsible for generating a Q-value per possible mutation strategy. A new fitness landscape state s_{t+1} is achieved (line 17), and a reward value r_t is computed by measuring the evolutionary efficiency of the population at this step (line 18). Randomly draw mini-batches of observations from memory to perform a step of gradient optimization, which can help to improve the robustness of the deep neural network (line 20). In the next, the deep neural network is used to predict the next mutation strategy $\hat{a}_{t+1}$ and its reward, without actually applying the mutation. If the run terminates, the budget assigned to the problem is finished, r^{target} is the same as the reward r_t (line 22). Otherwise, r^{target} is estimated as a linear combination of the current reward r_t and the predicted reward $\gamma\hat{Q}\left(s_{t+1}, \hat{a}_{t+1}\right)$, where $\hat{Q}$ is the (predicted) target Q-value and γ is the discount factor that makes the training focus more on immediate results compared to future rewards (line 25). Next, a target reward value r^{target} is used to train the deep neural network and to find the weights θ that minimizes the loss function $(r^{\text{target}}\text{-}Q(s_j, a_j; \theta))^2$(line 27). Finally, returning the weights θ of the deep neural network of DQN.

4.3. DEDQN algorithm with parameter adaptation

A historical memory parameter adaptive mechanism for both F and Cr is proposed by Tanabe et al. [20], which will be applied to improve the DEDQN algorithm. The parameter values are recorded in memory with length H and are denoted as $M_{F,i}$ and $M_{Cr,i}$, $(i = 1,2,\ldots,H)$, respectively. In the beginning, these values are set to 0.5. For each generation, F and Cr for individual x_i are created by choosing a random index r_i from [1, H] and updated by the following formulas:

$$F_{i,G} = randc\left(M_{F,r_i}, 0.1\right) \tag{31}$$

$$CR_{i,G} = randn\left(M_{Cr,r_i}, 0.1\right) \tag{32}$$

where $CR_{i,G}$ and $F_{i,G}$ obey the Gaussian distribution and Cauchy distribution, respectively. The standard deviations are preset to a fixed constant of 0.1.

At the Gth generation, if a trial vector $u_{i,G}$ is chosen into the new population, the associated $F_{i,G}$ and $Cr_{i,G}$ values are considered as successful and recorded as S_F and S_{Cr}, respectively. Afterward, at the end of the generation, the content of F memory $(M_{F,k}^{G+1})$ is updated as follow:

$$M_{F,k}^{G+1} = \begin{cases} mean_{WL}(S_F) & if\ S_F \ne \phi \\ M_{F,k}^{G}, & otherwise \end{cases} \tag{33}$$

where $mean_{WL}(S_F)$ is the weighted Lehmer mean computed by:

$$mean_{WL}(S_F) = \frac{\sum_{k=1}^{|S_F|} \omega_k S_{F,k}^2}{\sum_{k=1}^{|S_F|} \omega_k S_{F,k}} \tag{34}$$

where $\omega_k = \Delta f_k / \sum_{k=1}^{|S_F|} \Delta f_k$ and $\Delta f_k = \left|f\left(u_{k,G}\right) - f\left(x_{k,G}\right)\right|$.

On the other hand, the memory for $Cr\left(M_{Cr,k}^{G+1}\right)$ is updated as follow:

$$M_{Cr,k}^{G+1} = \begin{cases} mean_{WL}(S_{Cr}) & if\ S_{Cr} \ne \phi \\ M_{Cr,k}^{G}, & otherwise \end{cases} \tag{35}$$

where $mean_{WA}(S_{Cr})$ is the weighted arithmetic mean calculated by:

$$mean_{WA}(S_{Cr}) = \sum_{k=1}^{|S_{Cr}|} \omega_k S_{Cr,k} \tag{36}$$

Index k (0<k<k+1) determines the position in the memory to be updated.

Table 2
DQN training algorithm.

```
DQN training algorithm
1. Initialize parameter values of DE ($M_F$ , $NP$, $M_{CR}$);
2. Initialize replay $D$ memory to capacity $N$;
3. Initialize Q-value for each action by setting random weights $\theta$ of deep neural network;
4. Initialize target Q-value $\hat{Q}$ for each action by setting weights $\hat{\theta}=\theta$;
5. Random walk sampling; Calculate the features of the fitness landscape as $s_t$;
6. for run 1,...,$T$ do
7.     $t$=0;
8.     while $t<FES^{\max}$ do
9.         for $i$=1,...,$NP$ do
10.            if rand(0,1)<$\epsilon$ then
11.              Randomly select a mutation strategy $a_t$;
12.            else
13.              Select $a_t$ = argmax$_a$ $Q(s_t, a; \theta)$;
14.            end if
15.            Generate trial vector $\vec{u}_i$ for parent $\vec{x}_i$ using mutation $a_t$;
16.            Evaluate trial vector and select the best among $\vec{x}_i$ and $\vec{u}_i$;
17.            Random walk sampling; Calculate the features of the fitness landscape as $s_{t+1}$;
18.            Calculate evolutionary efficiency of population as $r_t$;
19.            Store observation ($s_t$, $a_t$ , $r_t$ , $s_{t+1}$) in $D$;
20.            Sample random mini batch of observations from $D$;
21.            if run terminates at step $t$+1 then
22.              $r^{\text{target}}=r_t$;
23.            else
24.              $\hat{a}_{t+1}=\arg\max_a Q(s_{t+1},a;\theta)$;
25.              $r^{\text{target}}=r_t+\gamma\hat{Q}(s_{t+1},\hat{a}_{t+1};\hat{\theta})$;
26.            end if
27.            Perform a gradient descent step on $(r^{\text{target}}-Q(s_j, a_j; \theta))^2$ with respect to $\theta$;
28.            Every $C$ steps set $\hat{\theta}=\theta$;
29.            $t$=$t$+1;
30.        end for
31.    end
32. end for
33.Return $\theta$
```

4.4. The framework of the proposed algorithm

Once the offline training is finished, the weights of the deep neural network of DQN are fixed. Then the trained DQN can be applied to DE algorithm, which uses the problem's fitness landscape information and DQN to realize the adaptive mutation strategy at each generation. The pose-code of the algorithm is given in Table 3. The process of the proposed algorithm is explained as follows: At first, randomly generating a population, calculating the fitness values and the features of the fitness landscape. Then the DQN will select a mutation strategy depending on the features. In the following, the crossover and selection operations will be performed and each individual will be evaluated. At the end of each generation, the control parameters are updated. This process continues until the stopping criterion is satisfied.

5. Experimental results and analysis

In this section, the performance of DEDQN is tested using CEC2017 benchmark function set, and a total of five advanced DE variants are used for comparison.

5.1. Benchmark functions and parameter settings

The real-parameter single-objective test functions originated from CEC2017 competition are performed to evaluate the performance of the proposed DEDQN algorithm. This benchmark set contains 30 functions with different characteristics. For all of the problems, the domain of definition is $[-100, 100]^D$. These functions can be classified into four types: unimodal functions f_1–f_3, basic multimodal functions f_4–f_{10}, hybrid functions f_{11}–f_{20}, and composite functions f_{21}–f_{30}. Furthermore, all the test functions are shifted and rotated. Note that the function f_2 was excluded from the comparison due to its numerical instability on the higher-dimension problem [44].

The performance of the proposed algorithm is compared with five well-known DE variants, such as jDE [18], EPSDE [30], JADE [19], SHADE [20], and LSHADE [21]. The parameter values of each algorithm are adopted according to the suggestions of the cited paper. Besides, the DEDQN algorithm used 4 layers neural network, the input layer has 4 neurons, the second layer and the third layer are full connection layers, we set it to 10 neurons, the last layer is the output layer which is set to 12 neurons. When

Table 3
Pseudo-code of the DEDQN algorithm.

```
DEDQN algorithm
Input: population size $NP$, problem dimension $D$, DQN, Generations $G_{max}$;
1. Initial population P= $\{x_{i,0}^{D} \mid i=1,2,\ldots,NP\}$;
2. Set $M_{CR}$=0.5, $M_F$=0.5, $mu=\phi$;
3. Set the individual survival counter survival= $NP$;
4. Calculate the fitness values of the population P;
5. Calculate the local fitness landscape act as the state, transport the parameters to DQN, then
   obtain the mutation strategy $mu'$;
6. Set $mu=mu'$;
7. for $G$=1,2,..,$G_{max}$ do
8.     $S_{CR}=\phi$; $S_F=\phi$;
9.     Initial the collection of individual evolutionary efficiency $S_e=\phi$;
10.    Initial offspring population O=$\phi$;
11.    for $i$=1,2,...,$NP$ do
12.        $F_i=randc(M_F, 0.1)$, $CR_i=randn(M_{CR}, 0.1)$;
13.        Randomly choose part of individuals from P for mutation operation;
14.        Generate trial vector $u_i$ by mutation $mu$;
15.        if $u_i$ better than $x_i$ then
16.          Calculate the individual evolutionary efficiency $e_i$ and add to $S_e$;
17.          Add $x_i$ to offspring collection O, and survival$_i$+=1;
18.          $CR_{i,G} \rightarrow S_{CR}$, $F_{i,G} \rightarrow SF$;
19.        end if
20.    end for
21.    Calculate population evolutionary efficiency $epp$(P), then update;
22.    Calculate the new state, transport the parameters to DQN, and update the mutation
       strategy $mu'$;
23.    Set $mu=mu'$;
24.    P=O;
25.    Update $M_{CR}$, $M_F$ based on $S_{CR}$, $S_F$;
26. end for
Output: the best solution.
```

training the DQN, each function is solved 10 times, and performed 500 generations evaluation. Besides, all the population size is 10 times the problem dimension.

5.2. Algorithm complexity

The run-time complexity of a classic DE is $O(G_{\max}\cdot\ NP\cdot\ D)$, where $G_{\max}$ is maximum number of generations, and NP is the number of population, and D is the dimension of the problem. The total complexity of JADE is $O(G_{\max}\cdot\ NP\cdot\ [D+\log(NP)])$. Considering that JADE, jDE, EPSDE, SHADE and LSHADE share the same framework of algorithm, so the total complexity of these algorithms are the same. DEDQN extends JADE in terms of the selection of mutation strategy, in which a random walk strategy is used for DQN. Therefore, the complexity of DEDQE is $(G_{\max}\cdot\ NP\cdot\ [D+\log(NP)]\cdot\ steps)$, where *steps* is number of random walk steps. Compared with a classic DE and the above mentioned DE variants, DEDQN increases the run-time complexity. Table 4 shows the complexity of the DEDQN algorithm for testing the problem of 10, 30, 50, and 100 dimensions. All experiments were performed and executed using MATLAB R2018a run on a PC with an Intel Core i7-7700 (3.60 GHz) CPU and 16 GB of RAM on a Windows 10 system. As defined in, T_0 is the time calculated by running the loop test [21]. Moreover, T_1 is the time required to execute 200000 evaluations of the benchmark function f_{18} by itself with D dimensions, and T_2 is the time required to execute DEDQN based on 200000 evaluations of f_{18} with D dimensions. T_3 is the mean T_2 value of 5 runs. It can be found that the overall computational time complexity of the DEDQN algorithm is not high.

Table 4
Algorithm complexity.

	T_0	T_1	T_3	$(T_3\text{-}T_1)/T_0$
$D = 10$	0.07692	0.41125	0.65328	3.14651
$D = 30$		0.62406	1.08564	6.00078
$D = 50$		0.94537	1.93706	12.8924
$D = 100$		2.53146	6.87532	56.4724

5.3. Algorithm performance

The experiments are conducted with the benchmark functions. According to practical application demands, we utilize the solution error measure $f(x)$ -$f(x^{*})$ as the final result, and error values and standard deviations less than 10^{-8} are regarded as zero, where x is the best solution searched by each algorithm in each generation and x^{*} is the well-known global optimal solution of each test function. Every test for each function and each algorithm runs 51 times independently, the condition for loop termination is set to $10000\times D$, which is the maximum number of function

Table 5
Results for 10*D* and 30*D*.

	10*D*					30*D*				
	Best	Worst	Median	Mean	Std. Dev.	Best	Worst	Median	Mean	Std. Dev.
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_4	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	6.4117E+01	5.8561E+01	5.6918E+01	1.1752E+01
f_5	0.0000E+00	6.9647E+00	2.2376E+00	2.6144E+00	1.3933E+00	1.5232E+01	2.7247E+01	2.0793E+01	2.0888E+01	2.6197E+00
f_6	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_7	1.1292E+01	1.4957E+01	1.2461E+01	1.2610E+01	8.1034E−01	4.4888E+01	6.0282E+01	5.0964E+01	5.1904E+01	3.8317E+00
f_8	9.9495E−01	5.1387E+00	2.4125E+00	2.6657E+00	8.9886E−01	1.2240E+01	2.7018E+01	2.1098E+01	2.0534E+01	2.8449E+00
f_9	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_{10}	4.0286E+00	2.8400E+02	5.2709E+01	9.4548E+01	7.8016E+01	1.8325E+03	3.2033E+03	2.7152E+03	2.6739E+03	2.6910E+02
f_{11}	0.0000E+00	2.8906E+00	1.2592E+00	1.2576E+00	8.4655E−01	4.9747E+00	8.7837E+01	2.1889E+01	3.1090E+01	2.3138E+01
f_{12}	0.0000E+00	3.3708E+02	1.1994E+02	1.2181E+02	6.7330E+01	3.4366E+02	1.9965E+03	1.1776E+03	1.1908E+03	3.3128E+02
f_{13}	0.0000E+00	8.5079E+00	4.8371E+00	3.6987E+00	2.5329E+00	4.9747E+00	4.4306E+01	2.3442E+01	2.3067E+01	9.4436E+00
f_{14}	0.0000E+00	5.8943E−01	0.0000E+00	2.2275E−02	8.8430E−02	2.0037E+00	3.2568E+01	2.5243E+01	2.3738E+01	6.7099E+00
f_{15}	2.1046E−05	4.9901E−01	2.7209E−02	1.4394E−01	1.8740E−01	2.4104E+00	2.3403E+01	1.0339E+01	1.1261E+01	4.9523E+00
f_{16}	2.7861E−02	1.1646E+00	4.4119E−01	4.5852E−01	2.5642E−01	2.4548E+01	5.0833E+02	1.5956E+02	1.7467E+02	1.1278E+02
f_{17}	5.7894E−02	1.0233E+00	3.4285E−01	3.7545E−01	2.2585E−01	4.6672E+01	8.7820E+01	6.5753E+01	6.5185E+01	8.2095E+00
f_{18}	4.0824E−06	1.3904E+01	2.3263E−01	2.4704E−01	2.7762E−01	2.1387E+01	9.7615E+01	2.5725E+01	3.0124E+01	1.3589E+01
f_{19}	1.1145E−05	3.7423E−02	1.9431E−02	1.4971E−02	1.1073E−02	3.9249E+00	1.4233E+01	7.8896E+00	7.9336E+00	2.4157E+00
f_{20}	0.0000E+00	3.1217E−01	0.0000E+00	6.1210E−03	4.3712E−02	5.2593E+01	1.9680E+02	6.6725E+01	7.4328E+01	2.3178E+01
f_{21}	1.0000E+02	2.0704E+02	2.0251E+02	1.5728E+02	5.0982E+01	2.1239E+02	2.2749E+02	2.1909E+02	2.1927E+02	3.1112E+00
f_{22}	0.0000E+00	1.0082E+02	1.0000E+02	9.6165E+01	1.9622E+01	1.0000E+02	1.0000E+02	1.0000E+02	1.0000E+02	1.0046E−13
f_{23}	3.0000E+02	3.0670E+02	3.0411E+02	3.0385E+02	1.6940E+00	3.5484E+02	3.8059E+02	3.6879E+02	3.6533E+02	5.1065E+00
f_{24}	1.0000E+02	3.3470E+02	3.3104E+02	3.0059E+02	7.2862E+01	4.2653E+02	4.4719E+02	4.3735E+02	4.3579E+02	3.3756E+00
f_{25}	3.9774E+02	4.4581E+02	3.9807E+02	4.1701E+02	2.2951E+01	3.8672E+02	3.8793E+02	3.8679E+02	3.8672E+02	1.6736E−02
f_{26}	3.0000E+02	3.0000E+02	3.0000E+02	3.0000E+02	0.0000E+00	9.5675E+02	1.2064E+03	1.0456E+03	1.0513E+03	5.4737E+01
f_{27}	3.8900E+02	3.8951E+02	3.8951E+02	3.8944E+02	1.7886E−01	4.8805E+02	5.1671E+02	5.0405E+02	5.0402E+02	6.3783E+00
f_{28}	3.0000E+00	6.1182E+02	3.0000E+02	3.8730E+02	1.3681E+02	3.0000E+02	4.5393E+02	3.0000E+02	3.3541E+02	5.3739E+01
f_{29}	2.2777E+02	2.4929E+02	2.3663E+02	2.3663E+02	4.4501E+00	4.1854E+02	5.0624E+02	4.8331E+02	4.8065E+02	1.6379E+01
f_{30}	3.9450E+02	5.1757E+05	1.7265E+02	1.7468E+04	1.6019E+05	1.9431E+03	2.3838E+03	2.0256E+03	2.0581E+03	1.0824E+02

evaluations (FES). The experimental results of the DEDQN carry out on the test functions with 10, 30, 50, and 100 dimensions are listed in Tables 5–6. The results include the obtained best, median, worst, mean, and standard deviation error values from the optimum solution of the proposed DEDQN over 51 runs for 29 benchmark functions.

Analyzing the optimal solution, the proposed DEDQN algorithm is capable of obtaining the optimal solutions for the unimodal function f_1 and f_3 for all dimensions; Among the multimodal functions f_4-f_{10}, the global optimum is only achieved for f_6 and f_9 for all dimensions and f_{10} is hard to solve; For hybrid functions f_{11}-f_{20}, f_{12} seems to be a complex function, and the error values for these functions are very large. For the composite functions f_{21}-f_{30}, the DEDQN algorithm appears to frequently fall into local optima, and the error values almost exceed 100. Hence, the proposed algorithm can obtain the desired results in most instances. For the worst solution, the proposed algorithm almost fell into local optimum solutions.

5.4. Statistical tests and comparisons to other algorithms

In this section, we compare the performance of the DEDQN with the performance of jDE, EPSDE, JADE, SHADE, and LSHADE based on the CEC2017 benchmark functions. The results are given in Tables 7–10 for each dimension. In these Tables, the average values of errors and the corresponding standard deviation values are shown for the six algorithms. The best result for each problem is shown in bold, and the statistical results are given in a separate column. The symbols +, -, and $\approx$ indicate whether a given algorithm performed significantly better (+), significantly worse (-), or not significantly different better or worse ($\approx$) than the DEDQN algorithm according to the Wilcoxon rank-sum test at the 0.05 significance level.

The summarized results of the statistical testing are presented in Table 11 and the number of wins (+) and losses (-) are compared. DEDQN displays better performance than the other five DE variants for all dimensions. Besides, based on the average results obtained, the average ranking of all algorithms, as produced by the Friedman rank test is summarized in Table 12. The results in Table 12 are consistent with the results in Table 11, in which DEDQN has the best rank. Furthermore, Fig. 2 shows the score of each algorithm, which consists of the summation of error values and rank-based mean values for each problem for all dimensions. In the evaluation method, the higher weights are given for higher dimensions [32]. The results show that the proposed algorithm obtains the highest score, LSHADE gets second place. All the experimental results indicate that the DEDQN algorithm achieves a better effect on CEC2017 benchmark function set.

6. Conclusion and future work

In this paper, we presented a DEDQN algorithm, a deep reinforcement learning method to realize an adaptive mutation strategy differential evolution, in which the suitable mutation strategy is selected at each generation according to the fitness landscape of the problem. DEDQN has two phase, offline training and online prediction phase. Firstly, the neural network of DQN is trained by multiple running DEDQN to solve a training function, then making it to learn the most rewarding mutation given the fitness landscape information of the problem. Finally, applying the trained DQN to predict the mutation strategy when solving a new problem. Experiments are performed on CEC2017 benchmark suite, each function is evaluated with 10, 30, 50, and 100 dimensions. The Wilcoxon rank-sum test and Friedman rank test are shown that the DEDQN algorithm is significantly

Table 6
Results for 50*D* and 100*D*.

	50*D*					100*D*				
	Best	Worst	Median	Mean	Std. Dev.	Best	Worst	Median	Mean	Std. Dev.
f_1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_4	0.0000E+00	1.4230E+02	3.2499E+01	6.1181E+01	4.7097E+01	5.6552E−01	2.3934E+02	1.4970E+02	1.4255E+02	6.1391E+01
f_5	4.8651E+01	8.9023E+01	7.5527E+01	7.4570E+01	8.2128E+00	2.7827E+02	3.3546E+02	3.0592E+02	3.0791E+02	1.4159E+01
f_6	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_7	1.0641E+02	1.4688E+02	1.3442E+02	1.3457E+02	7.3966E+00	3.6037E+02	4.7121E+02	4.3229E+02	4.3196E+02	1.7315E+01
f_8	5.9779E+01	8.8732E+01	7.7028E+01	7.6111E+01	6.1344E+00	2.8264E+02	3.4522E+02	3.0989E+02	3.0796E+02	1.3300E+01
f_9	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
f_{10}	6.0707E+03	7.4655E+03	6.9510E+03	6.9251E+03	3.3160E+02	1.8644E+04	2.1237E+04	2.0004E+04	1.9957E+04	5.0622E+02
f_{11}	5.9058E+01	1.3268E+02	9.1891E+01	9.3402E+01	1.8498E+01	3.6305E+02	1.1064E+03	7.0820E+02	7.1602E+02	1.3529E+02
f_{12}	1.0551E+03	6.6223E+03	2.1175E+03	2.3906E+03	1.0165E+03	1.0623E+04	3.5708E+04	2.1534E+04	2.1299E+04	1.4237E+04
f_{13}	1.4215E+01	6.9834E+01	4.1157E+01	4.3803E+01	4.4529E+01	5.3208E+02	3.9503E+03	2.1242E+03	2.0660E+03	8.3687E+02
f_{14}	2.5135E+01	8.0278E+01	4.9185E+01	5.4543E+01	1.5973E+01	2.0605E+02	3.6996E+02	2.4366E+02	2.3255E+02	4.4848E+01
f_{15}	1.3983E+01	7.4006E+01	3.7581E+01	3.8260E+01	4.8427E+01	1.5139E+02	3.1243E+02	2.2281E+02	2.2814E+02	5.1292E+01
f_{16}	3.8290E+02	1.0027E+03	7.3024E+02	7.2964E+02	1.3411E+02	2.1927E+03	3.6513E+03	3.1247E+03	3.1553E+03	3.3060E+02
f_{17}	3.6846E+02	7.9954E+02	5.5845E+02	5.6267E+02	1.1881E+02	1.6129E+03	2.6061E+03	2.1661E+03	2.1496E+03	2.0400E+02
f_{18}	3.2010E+01	3.4112E+02	1.4628E+02	1.5455E+02	7.8382E+01	1.7643E+02	4.8576E+02	2.6810E+02	2.7279E+02	6.6666E+01
f_{19}	2.8520E+00	2.7293E+01	4.0028E+01	2.8171E+01	7.6047E+00	1.1170E+02	2.1611E+02	1.7201E+02	1.6739E+02	3.1315E+01
f_{20}	2.2390E+02	6.6486E+02	4.3449E+02	4.4138E+02	1.0956E+02	2.2918E+03	3.2135E+03	2.7665E+03	2.7486E+03	2.1353E+02
f_{21}	2.5624E+02	2.8631E+02	2.7531E+02	2.7493E+02	6.9355E+00	4.5056E+02	5.5084E+02	5.2222E+02	5.1858E+02	1.8504E+01
f_{22}	1.0000E+02	7.8498E+03	7.2266E+03	5.4220E+03	2.9270E+03	7.9702E+02	1.7819E+03	1.1952E+03	1.2923E+03	4.4661E+02
f_{23}	4.7384E+02	5.0961E+02	4.9582E+02	4.9405E+02	7.3541E+00	5.8061E+02	7.8093E+02	7.5062E+02	7.1554E+02	7.1751E+01
f_{24}	5.4485E+02	5.7282E+02	5.5673E+02	5.5717E+02	6.8599E+00	8.1672E+02	1.3594E+03	9.8933E+02	1.0524E+03	4.3669E+01
f_{25}	4.0094E+02	4.6531E+02	4.2672E+02	4.2236E+02	3.1819E+01	6.3931E+02	8.4998E+02	7.6102E+02	7.4090E+02	4.5134E+01
f_{26}	1.2130E+03	1.7868E+03	1.5988E+03	1.5757E+03	1.2714E+02	3.1540E+03	5.2080E+03	3.6019E+03	3.6099E+03	2.8382E+02
f_{27}	4.8918E+02	5.8184E+02	5.0922E+02	5.0024E+02	1.7998E+01	5.3407E+02	6.7000E+02	6.2463E+02	6.2135E+02	2.6530E+01
f_{28}	4.5884E+02	5.6962E+02	4.9942E+02	4.8639E+02	2.6749E+01	4.7840E+02	6.2186E+02	5.2831E+02	5.3603E+02	2.4756E+01
f_{29}	4.5152E+02	6.5862E+02	5.0628E+02	5.1534E+02	4.3708E+01	1.6714E+03	2.9746E+03	2.2416E+03	2.2867E+03	2.8844E+02
f_{30}	4.7941E+05	6.3959E+05	5.4927E+05	6.0829E+05	1.0272E+04	2.0644E+03	2.7350E+03	2.4101E+03	2.4266E+03	1.3904E+02

Table 7
Comparison for 10*D*.

	jDE Mean (Std Dev)	EPSDE Mean (Std Dev)	JADE Mean (Std Dev)	SHADE Mean (Std Dev)	LSHADE Mean (Std Dev)	DEDQN Mean (Std Dev)
f_1	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_3	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_4	2.8171E−02(3.73E−02) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_5	5.9044E+00(1.16E+00) -	4.9700E+00(1.25E+00) -	3.3752E+00(9.50E−01) -	2.6819E+00(9.51E−01) ≈	3.1046E+00(7.61E−01) -	**2.6144E+00(1.39E+00)**
f_6	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_7	1.7550E+01(1.53E+00) -	1.5913E+01(1.21E+00) -	1.3446E+01(1.55E+00) -	1.2839E+01(7.47E−01) ≈	1.5252E+01(8.81E−01) -	**1.2610E+01(8.10E−01)**
f_8	6.7356E+00(1.39E+00) -	5.3575E+00(1.44E+00) -	3.6204E+00(9.41E−01) -	2.8406E+00(8.08E−01) ≈	3.4938E+00(9.82E−01) -	**2.6657E+00(8.98E−01)**
f_9	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_{10}	2.4716E+02(8.98E+01) -	4.1825E+02(1.06E+02) -	8.7772E+01(5.57E+01) +	**5.8839E+01(5.85E+01) +**	6.3443E+01(6.46E+01) +	9.4548E+01(7.80E+01)
f_{11}	1.8023E+00(1.22E+00) -	2.3126E+00(9.41E−01) -	2.2982E+00(6.53E−01) -	**1.1324E−01(4.17E−01) +**	1.5711E+00(5.64E−01) -	1.2576E+00(8.46E−01)
f_{12}	2.9719E+02(5.38E+01) -	1.2966E+02(1.37E+02) ≈	1.1871E+02(9.39E+01) +	9.7312E+01(7.85E+01) +	**4.4523E+01(6.40E+01) +**	1.2537E+02(1.20E+02)
f_{13}	6.4760E+00(1.67E+00) -	6.4252E+00(1.86E+00) -	4.4309E+00(3.62E+00) -	3.9197E+00(2.17E+00) -	4.0194E+00(2.08E+00) -	**3.6987E+00(2.53E+00)**
f_{14}	5.1879E−01(4.77E−01) -	2.0041E+01(1.87E+00) -	5.5307E−01(3.83E−01) -	2.6186E−01(5.67E−01) -	2.9698E−01(3.41E−01) -	**2.2275E−02(8.84E−02)**
f_{15}	1.8349E−01(1.67E−01) ≈	9.6345E−01(6.80E−01) -	5.3267E−01(1.66E−01) -	**1.1479E−01(1.71E−01) +**	2.2072E−01(2.13E−01) -	1.4394E−01(1.87E−01)
f_{16}	6.5455E−01(2.60E−01) -	1.3212E+01(6.70E+00) -	1.3405E+00(5.54E−01) -	4.9638E−01(1.95E−01) -	6.4848E−01(1.79E−01) -	**4.5852E−01(2.56E−01)**
f_{17}	1.0527E+00(4.67E−01) -	2.1008E+01(4.04E+00) -	5.2248E−01(4.30E−01) ≈	4.1586E−01(2.46E−01) ≈	**2.0571E−01(2.33E−01) +**	3.7545E−01(2.25E−01)
f_{18}	3.6582E+01(5.58E+00) -	1.8854E+01(4.58E+00) -	6.9920E−01(1.39E+00) -	**1.3228E−01(1.79E−01) +**	2.1619E−01(1.97E−01) ≈	2.4704E−01(2.77E−01)
f_{19}	3.3816E−02(2.38E−02) -	7.6153E−01(3.33E−01) -	4.6226E−02(2.21E−02) -	6.0500E−02(2.19E−01) -	2.0137E−02(1.17E−02) ≈	**1.4971E−02(1.10E−02)**
f_{20}	6.1210E−03(4.37E−02) ≈	1.6370E+01(5.26E+00) -	6.1210E−03(4.37E−02) ≈	3.6726E−02(1.01E−01) -	**0.0000E+00(0.00E+00) +**	6.1210E−03(4.37E−02)
f_{21}	1.6609E+02(5.36E+01) -	1.5486E+02(5.02E+01) ≈	1.5558E+02(4.97E+01) ≈	1.6237E+02(5.17E+01) ≈	**1.5058E+02(4.84E+01)** +	1.5728E+02(5.09E+01)
f_{22}	**8.4366E+01(3.46E+01) +**	9.2972E+01(2.50E+01) ≈	9.7022E+01(1.57E+01) ≈	9.8062E+01(1.40E+01) ≈	1.0000E+02(3.37E−09) -	9.6165E+01(1.96E+01)
f_{23}	3.0686E+02(1.53E+00) ≈	3.1299E+02(2.19E+00) -	3.0502E+02(1.38E+00) ≈	3.0430E+02(1.47E+00) ≈	3.0444E+02(1.35E+00) ≈	**3.0385E+02(1.69E+00)**
f_{24}	**2.8434E+02(9.76E+01) +**	3.3563E+02(2.68E+00) -	3.0474E+02(6.89E+01) ≈	3.1063E+02(6.66E+01) -	3.0784E+02(6.90E+01) -	3.0059E+02(7.28E+01)
f_{25}	**4.0433E+02(1.60E+01) +**	4.1899E+02(2.33E+01) ≈	4.1963E+02(2.32E+01) ≈	4.2051E+02(2.32E+01) ≈	4.1048E+02(2.05E+01) +	4.1701E+02(2.29E+01)
f_{26}	3.0000E+00(0.00E+00) ≈	**2.9549E+02(8.45E+01) +**	3.0000E+02(0.00E+00) ≈	3.0000E+00(0.00E+00) ≈	3.0000E+00(0.00E+00) ≈	3.0000E+02(0.00E+00)
f_{27}	3.8975E+02(1.05E+00) ≈	**3.8498E+02(2.02E+01) +**	3.8917E+02(7.45E−01) ≈	3.8944E+02(1.79E−01) ≈	3.8926E+02(2.59E−01) ≈	3.8944E+02(1.78E−01)
f_{28}	**3.1167E+02(5.84E+01) +**	4.7352E+02(2.30E+00) -	3.8119E+02(1.33E+02) +	3.9507E+02(1.42E+02) -	3.3503E+02(9.70E+01) +	3.8730E+02(1.36E+02)
f_{29}	2.4829E+02(4.87E+00) -	2.5380E+02(9.45E+00) -	2.4469E+02(6.27E+00) -	2.3801E+02(6.58E+00) ≈	2.3824E+02(3.81E+00) ≈	**2.3663E+02(4.45E+00)**
f_{30}	1.6443E+04(1.14E+05) +	**2.1360E+02(2.26E+01) +**	2.6562E+03(9.99E+03) +	9.6553E+04(2.65E+05) -	1.6613E+04(1.14E+05) +	1.7468E+04(1.60E+05)

Table 8
Comparison for 30*D*.

	jDE Mean (Std Dev)	EPSDE Mean (Std Dev)	JADE Mean (Std Dev)	SHADE Mean (Std Dev)	LSHADE Mean (Std Dev)	DEDQN Mean (Std Dev)
f_1	1.2305E+00(7.24E−01) -	4.6790E−05(2.29E−04) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_3	9.8624E+03(5.43E+03) -	1.9560E+04(3.78E+04) -	1.6086E+04(2.58E+04) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_4	8.6636E+01(3.68E+00) -	5.9395E+01(2.31E+00) -	5.8779E+01(1.08E+00) -	5.8561E+01(1.13E−14) -	5.8561E+01(3.21E−14) -	**5.6918E+01(1.17E+01)**
f_5	7.5209E+01(7.89E+00) -	8.7224E+01(7.11E+00) -	5.5881E+01(5.80E+00) -	5.1778E+01(6.17E+00) -	2.1306E+01(3.12E+00) ≈	**2.0888E+01(2.61E+00)**
f_6	5.0161E−07(1.47E−07) -	6.0837E−04(3.45E−04) -	8.2029E−04(4.13E−04) -	8.0526E−09(3.25E−08) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_7	1.1389E+02(7.62E+00) -	1.2041E+02(8.01E+00) -	8.6781E+01(5.08E+00) -	8.6756E+01(7.85E+00) -	5.2933E+01(3.34E+00) -	**5.1904E+01(3.83E+00)**
f_8	7.6541E+01(6.95E+00) -	8.8707E+01(7.64E+00) -	5.5223E+01(4.87E+00) -	4.9703E+01(6.67E+00) -	2.1359E+01(3.08E+00) ≈	**2.0534E+01(2.84E+00)**
f_9	2.2835E−08(3.20E−08) -	7.3851E−01(2.11E+00) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_{10}	3.7834E+03(2.85E+02) -	4.8211E+03(2.91E+02) -	3.3728E+03(2.10E+02) -	3.1996E+03(2.47E+02) -	2.8504E+03(2.03E+02) -	**2.6739E+03(2.69E+02)**
f_{11}	4.2146E+01(2.27E+01) -	2.6192E+01(5.65E+00) +	3.9831E+01(2.86E+01) -	4.1116E+01(2.96E+01) -	**2.2623E+01(2.66E+01) +**	3.1090E+01(2.31E+01)
f_{12}	1.1271E+04(7.45E+02) -	2.8537E+04(2.70E+04) -	1.1534E+03(4.32E+02) ≈	1.0386E+03(3.78E+02) +	**1.0172E+03(3.56E+02) +**	1.1908E+03(3.31E+02)
f_{13}	1.1000E+02(2.60E+01) -	1.9276E+02(3.02E+02) -	5.9370E+01(3.09E+01) -	4.6390E+01(2.15E+01) -	2.7206E+01(5.85E+00) -	**2.3067E+01(9.44E+00)**
f_{14}	5.1105E+01(4.76E+00) -	4.3817E+01(4.41E+00) -	6.2950E+03(9.47E+03) -	3.1524E+01(1.96E+00) -	2.6645E+01(3.18E+00) -	**2.3738E+01(6.70E+00)**
f_{15}	3.1655E+01(5.80E+00) -	7.3511E+01(9.96E+01) -	1.9110E+03(6.62E+03) -	9.7746E+00(3.48E+00) +	**3.4842E+00(1.86E+00) +**	1.1261E+01(4.95E+00)
f_{16}	6.2340E+02(1.23E+02) -	9.1398E+02(1.56E+02) -	5.1846E+02(1.35E+02) -	3.5244E+02(1.18E+02) -	1.9889E+02(9.24E+00) -	**1.7467E+02(1.12E+02)**
f_{17}	1.5552E+02(2.55E+01) -	3.4543E+02(7.87E+01) -	1.3590E+02(2.57E+01) -	1.1597E+02(1.65E+01) -	8.9369E+01(7.13E+00) -	**6.5185E+01(8.20E+00)**
f_{18}	5.4167E+01(1.02E+01) -	6.1509E+01(4.74E+01) -	1.6282E+04(5.66E+04) -	2.4108E+01(3.93E+00) +	**1.0526E+01(7.35E+01) +**	3.0124E+01(1.35E+01)
f_{19}	2.4323E+01(2.80E+00) -	2.1495E+01(2.26E+00) -	6.4864E+02(3.95E+03) -	1.4661E+01(1.85E+00) -	**6.1323E+00(1.91E+00) +**	7.9336E+00(2.41E+00)
f_{20}	1.5537E+02(4.29E+00) -	2.6369E+02(7.48E+01) -	1.5864E+02(4.57E+01) -	1.3286E+02(2.79E+01) -	8.1352E+01(1.46E+01) -	**7.4328E+01(2.31E+01)**
f_{21}	2.7732E+02(9.18E+00) -	2.9674E+02(6.28E+00) -	2.5594E+02(5.46E+00) -	2.4910E+02(6.13E+00) -	2.2238E+02(4.25E+00) ≈	**2.1927E+02(3.11E+00)**
f_{22}	1.0000E+02(3.18E−08) ≈	2.0717E+03(1.40E+03) -	**1.0000E+02(1.00E−13)** ≈	**1.0000E+00(1.00E−13)** ≈	**1.0000E+02(1.00E−13)** ≈	**1.0000E+02(1.00E−13)**
f_{23}	4.1859E+02(8.18E+00) -	4.4519E+02(9.21E+00) -	3.9958E+02(5.62E+00) -	3.9702E+02(7.02E+00) -	3.6948E+02(3.89E+00) -	**3.6533E+02(5.10E+00)**
f_{24}	4.9650E+02(8.66E+00) -	5.2948E+02(9.82E+00) -	4.6140E+02(7.02E+00) -	4.6475E+02(6.08E+00) -	4.3795E+02(5.30E+00) -	**4.3579E+02(3.37E+00)**
f_{25}	3.8684E+02(7.67E−02) ≈	3.8685E+02(2.27E−02) ≈	3.8689E+02(9.52E−02) ≈	3.8675E+02(1.37E−02) ≈	3.8672E+02(2.39E−02) ≈	**3.8672E+02(1.67E−02)**
f_{26}	1.6767E+03(7.40E+01) -	1.6110E+03(7.28E+01) -	1.3993E+03(5.67E+01) -	1.3523E+03(6.34E+01) -	1.1012E+03(7.86E+01) -	**1.0513E+03(5.47E+01)**
f_{27}	5.0417E+02(5.99E+00) ≈	**5.0000E+02(8.72E−05) +**	5.0180E+02(4.81E+00) +	5.0395E+00(5.45E+00) ≈	5.0437E+02(5.22E+00) ≈	5.0402E+02(6.37E+00)
f_{28}	4.0372E+02(1.57E+01) -	4.9958E+02(1.40E+00) -	**3.2290E+02(4.41E+01) +**	3.3262E+02(5.15E+01) ≈	3.2966E+02(4.48E+01) +	3.3541E+02(5.37E+01)
f_{29}	5.8939E+02(3.62E+01) -	6.3442E+02(1.07E+02) -	5.2521E+02(2.51E+01) -	5.5059E+02(2.36E+01) -	**4.3810E+02(1.51E+01) +**	4.8065E+02(1.63E+01)
f_{30}	2.3414E+03(2.01E+02) -	2.2302E+02(1.17E+02) -	2.0200E+03(7.09E+01) +	2.0205E+03(8.91E+01) +	**2.0037E+03(6.09E+01)** +	2.0581E+03(1.08E+02)

Table 9
Comparison for 50*D*.

	jDE Mean (Std Dev)	EPSDE Mean (Std Dev)	JADE Mean (Std Dev)	SHADE Mean (Std Dev)	LSHADE Mean (Std Dev)	DEDQN Mean (Std Dev)
f_1	4.0278E+03(3.21E+03) -	9.9122E+02(2.24E+02) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.00000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_3	1.7686E+05(2.55E+04) -	4.2046E+05(9.51E+04) -	4.6375E+04(6.44E+04) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_4	1.2586E+02(2.55E+01) -	**3.5106E+01(6.31E−01) +**	6.7272E+01(4.44E+01) -	5.4965E+01(3.98E+01) +	7.93326E+01(2.15E+02) -	6.1181E+01(4.70E+01)
f_5	1.9772E+02(1.08E+01) -	2.4857E+02(1.16E+01) -	1.4369E+02(9.55E+00) -	1.6546E+02(1.21E+01) -	**3.34954E+01(6.98E+00) +**	7.4570E+01(8.21E+00)
f_6	3.1902E−04(7.05E−05) -	3.9610E−02(7.96E−03) -	7.0401E−04(6.63E−04) -	9.0986E−09(3.13E−08) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_7	2.6080E+02(1.10E+01) -	2.9510E+02(1.17E+01) -	1.9785E+02(8.92E+00) -	2.3801E+02(1.42E+01) -	1.57634E+02(4.53E+00) -	**1.3457E+02(7.39+00)**
f_8	1.9758E+02(1.14E+01) -	2.4365E+02(1.36E+01) -	1.4327E+02(1.07E+01) -	1.7067E+02(1.08E+01) -	8.24236E+01(5.79E+00) -	**7.6111E+01(6.13E+00)**
f_9	3.6400E−01(4.21E−01) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_{10}	7.7098E+03(3.47E+02) -	1.0616E+04(3.58E+02) -	7.2544E+03(2.46E+02) -	7.6944E+03(7.04E+02) -	**3.23224E+03(2.15E+02) +**	6.9251E+03(3.31E+02)
f_{11}	1.0351E+02(1.01E+01) -	1.0257E+02(3.62E+01) -	1.9720E+02(2.08E+02) -	**3.7013E+01(6.08E+00) +**	4.73087E+01(9.57E+00) +	9.3402E+01(1.84E+01)
f_{12}	2.3899E+05(1.55E+05) -	2.5588E+05(1.89E+05) -	2.4053E+03(9.15E+02) -	2.4076E+03(7.28E+02) -	2.41259E+03(4.39E+02) -	**2.3906E+03(1.01E+03)**
f_{13}	2.2163E+03(1.63E+03) -	1.5402E+03(1.91E+03) -	1.7563E+02(5.63E+01) -	1.8485E+02(4.59E+01) -	5.50384E+01(2.33E+01) -	**4.3803E+01(4.45E+01)**
f_{14}	1.3229E+02(1.19E+01) -	1.1591E+02(1.68E+01) -	9.2783E+03(3.75E+04) -	6.1860E+01(1.00E+01) -	5.69760E+01(2.83E+00) -	**5.4543E+01(1.59E+01)**
f_{15}	1.8618E+02(2.59E+01) -	1.8790E+02(1.93E+02) -	4.7692E+01(1.33E+01) -	4.0183E+01(1.12E+01) -	4.34640E+01(1.16E+01) -	**3.8260E+01(4.84E+01)**
f_{16}	1.6550E+03(2.08E+02) -	2.1429E+03(1.68E+02) -	1.4051E+03(1.65E+02) -	1.1070E+03(1.47E+02) -	7.92701E+02(1.34E+02) -	**7.2964E+02(1.34E+02)**
f_{17}	1.0932E+03(1.49E+02) -	1.3425E+03(1.16E+02) -	9.9429E+02(1.48E+02) -	7.4039E+02(1.19E+02) -	**3.60627E+02(8.61E+01) +**	5.6267E+02(1.18E+02)
f_{18}	7.6458E+03(4.67E+03) -	1.3100E+03(9.68E+02) -	7.8208E+01(4.70E+01) +	5.8122E+01(2.30E+01) +	**5.08801E+01(2.29E+01) +**	1.5455E+02(7.83E+01)
f_{19}	7.7800E+01(9.83E+00) -	5.8178E+01(8.37E+00) -	3.8550E+01(1.43E+01) -	3.3427E+01(1.04E+01) -	2.93105E+01(8.54E+00) -	**2.8171E+01(7.60E+00)**
f_{20}	9.1350E+02(1.09E+02) -	1.0573E+03(1.37E+02) -	8.0131E+02(1.26E+02) -	7.3080E+02(1.48E+02) -	5.60250E+02(8.38E+01) -	**4.4138E+02(1.09E+02)**
f_{21}	4.0298E+02(1.32E+01) -	4.5125E+02(1.25E+01) -	3.4710E+02(9.28E+00) -	3.6976E+02(1.27E+01) -	**2.64270E+02(7.45E+00) +**	2.7493E+02(6.93E+00)
f_{22}	5.9694E+03(3.64E+03) -	1.1105E+04(3.54E+02) -	**3.3168E+03(3.01E+03) +**	6.2074E+03(3.97E+03) -	9.72543E+03(1.52E+03) -	5.4220E+03(2.92E+03)
f_{23}	6.1763E+02(1.35E+01) -	6.4263E+02(1.76E+01) -	5.6135E+02(1.21E+01) -	5.9231E+02(1.25E+01) -	5.50672E+02(6.89E+00) -	**4.9405E+02(7.35E+00)**
f_{24}	6.8945E+02(2.08E+01) -	7.9954E+02(1.59E+01) -	6.0916E+02(1.10E+01) -	6.5724E+02(1.24E+01) -	5.89163E+02(5.56E+00) -	**5.5717E+02(6.85E+00)**
f_{25}	4.8858E+02(5.48E+00) -	4.3126E+02(3.33E−02) -	5.0495E+02(3.21E+01) -	4.8121E+02(3.15E+00) -	4.82119E+02(4.24E+00) -	**4.2236E+02(3.18E+01)**
f_{26}	2.9933E+03(1.39E+02) -	3.3568E+03(4.56E+02) -	2.2638E+03(1.02E+02) -	2.4833E+03(1.26E+02) -	1.64866E+03(6.52E+01) -	**1.5757E+03(1.27E+02)**
f_{27}	5.2460E+02(9.07E+00) -	**5.0001E+02(7.62E−05)** ≈	5.2357E+02(8.53E+00) -	5.2127E+02(6.17E+00) -	5.38692E+02(1.22E+01) -	5.0024E+02(1.79E+01)
f_{28}	4.6066E+02(6.77E+00) -	5.0001E+02(5.82E−05) -	4.7608E+02(2.35E+01) -	4.9539E+02(1.66E+01) -	4.78475E+02(1.42E+01) -	**4.5639E+02(2.67E+01)**
f_{29}	9.0024E+02(1.10E+02) -	1.5760E+03(2.14E+02) -	6.1683E+02(5.80E+01) -	6.8811E+02(4.90E+01) -	**3.57170E+02(1.06E+01) +**	5.1534E+02(4.37E+01)
f_{30}	6.6870E+05(3.34E+04) -	**4.3502E+02(2.13E+02) +**	6.2638E+05(4.51E+04) -	6.2581E+05(4.28E+04) -	6.64365E+05(7.91E+04) -	6.0829E+05(1.02E+04)

Table 10
Comparison for 100*D*.

	jDE Mean (Std Dev)	EPSDE Mean (Std Dev)	JADE Mean (Std Dev)	SHADE Mean (Std Dev)	LSHADE Mean (Std Dev)	DEDQN Mean (Std Dev)
f_1	6.1310E+03(2.95E+03) -	3.7470E+03(3.55E+03) -	3.2446E+05(1.92E+05) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_3	5.6056E+05(2.63E+04) -	9.6216E+05(1.56E+05) -	2.0402E+02(2.76E+01) -	5.8040E−02(9.29E−02) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_4	2.5720E+02(1.33E+01) -	**9.3834E+01(7.22E−01) +**	4.0479E+02(1.50E+01) -	1.9993E+02(6.64E+00) -	1.9397E+02(1.13E+01) -	1.4255E+02(6.13E+01)
f_5	6.2935E+02(2.49E+01) -	7.2475E+02(2.11E+01) -	6.3754E+02(2.54E+01) -	5.3856E+02(7.32E+01) -	5.9655E+02(8.65E+00) -	**3.0791E+02(1.41E+01)**
f_6	7.3990E−01(8.38E−01) -	1.3219E+00(4.10E−01) -	4.3615E+00(1.08E−01) -	9.5144E−01(1.53E−02) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_7	7.4841E+02(2.11E+01) -	8.2121E+02(1.92E+01) -	5.2011E+02(1.46E+01) -	5.3086E+02(2.28E+01) -	**1.5694E+02(7.56E+00) +**	4.3196E+02(1.73E+01)
f_8	6.1604E+02(2.66E+01) -	7.2562E+02(1.77E+01) -	4.0278E+02(1.59E+01) -	5.3086E+02(2.28E+01) -	5.1008E+02(7.76E+00) -	**3.0796E+02(1.33E+01)**
f_9	8.9004E+01(3.44E+01) -	1.1011E−01(3.06E−01) -	3.3190E−01(3.80E−01) -	1.7554E−03(1.25E−02) -	**0.0000E+00(0.00E+00)** ≈	**0.0000E+00(0.00E+00)**
f_{10}	2.0853E+04(4.18E+02) -	2.7852E+04(5.09E+02) -	2.0257E+04(4.09E+02) -	2.5648E+04(1.10E+03) -	**1.9889E+04(5.59E+02)** ≈	1.9957E+04(5.06E+02)
f_{11}	5.7529E+03(1.29E+03) -	7.4214E+02(1.75E+02) -	2.3924E+04(1.47E+04) -	**3.6249E+02(8.70E+01) +**	5.2715E+02(9.21E+03) +	7.1602E+02(1.35E+02)
f_{12}	1.0071E+07(2.68E+06) -	1.9335E+07(3.17E+07) -	2.2997E+04(8.51E+03) -	2.2315E+04(9.01E+03) -	2.4887E+04(9.84E+03) -	**2.1299E+04(1.42E+04)**
f_{13}	2.7154E+03(9.94E+02) -	1.0486E+05(1.60E+05) -	2.3774E+03(1.08E+03) -	2.1964E+03(1.02E+03) -	**1.2433E+03(9.49E+02) +**	2.0660E+03(8.36E+02)
f_{14}	1.9764E+04(1.02E+04) -	2.5214E+03(2.37E+03) -	2.8721E+02(4.30E+01) -	2.7704E+02(3.71E+01) -	2.5451E+02(3.42E+01) -	**2.3255E+02(4.48E+01)**
f_{15}	1.3365E+03(1.02E+03) -	2.9128E+03(2.05E+03) -	2.4973E+02(5.30E+01) -	2.4301E+02(4.97E+01) -	2.3894E+02(4.02E+01) -	**2.2814E+02(5.12E+01)**
f_{16}	5.4632E+03(3.76E+02) -	7.0483E+03(3.30E+02) -	4.6626E+03(3.34E+02) -	4.7072E+03(2.77E+02) -	**3.1383E+03(2.75E+02) +**	3.1553E+03(3.30E+02)
f_{17}	3.6751E+03(2.16E+02) -	4.4448E+03(2.53E+02) -	3.2677E+03(2.38E+02) -	2.8271E+03(2.62E+02) -	2.4423E+03(1.91E+02) -	**2.1496E+03(2.04E+02)**
f_{18}	6.7863E+05(1.85E+05) -	1.5662E+05(7.03E+04) -	2.7273E+02(8.23E+01) ≈	2.4771E+02(5.73E+01) +	**2.2428E+02(4.90E+01) +**	2.7279E+02(6.66E+01)
f_{19}	1.0116E+03(1.08E+03) -	4.2246E+02(2.81E+02) -	1.7156E+02(2.81E+01) -	1.7558E+02(2.96E+01) -	1.6882E+02(2.17E+01) ≈	**1.6739E+02(3.13E+01)**
f_{20}	3.4720E+03(1.69E+02) -	3.9315E+03(1.84E+02) -	3.4039E+03(2.08E+02) -	3.4215E+03(2.58E+02) -	**2.7027E+03(2.08E+02) +**	2.7486E+03(2.13E+02)
f_{21}	8.5116E+02(2.36E+01) -	9.5014E+02(1.87E+01) -	6.2840E+02(1.88E+01) -	7.5788E+02(1.66E+01) -	6.8448E+02(8.71E+00) -	**5.1858E+02(1.85E+01)**
f_{22}	2.1557E+04(5.66E+02) -	2.8631E+04(5.54E+02) -	2.1182E+04(3.96E+02) -	2.6964E+04(7.92E+02) -	**1.1477E+04(5.23E+02)** +	1.2923E+04(4.46E+02)
f_{23}	9.5119E+02(1.74E+01) -	1.2222E+03(1.59E+01) -	8.8377E+02(1.17E+01) -	1.0464E+03(2.05E+01) -	7.9996E+02(9.05E+00) -	**7.1554E+02(7.17E+01)**
f_{24}	1.4246E+03(2.26E+01) -	1.5984E+03(1.95E+01) -	1.2295E+03(1.66E+01) -	1.3940E+03(2.16E+01) -	**9.3433E+02(7.88E+00) +**	1.0524E+03(4.36E+01)
f_{25}	8.5593E+02(3.43E+01) -	7.5104E+02(1.46E+01) -	7.5088E+02(2.83E+01) -	7.5631E+02(1.99E+01) -	7.5089E+02(2.55E+01) -	**7.4090E+02(4.51E+01)**
f_{26}	9.0811E+03(1.98E+02) -	1.0878E+04(1.76E+02) -	6.3301E+03(1.84E+02) -	7.2947E+03(8.21E+02) -	**3.5949E+03(9.91E+01)** ≈	3.6099E+03(2.83E+02)
f_{27}	6.9566E+02(2.38E+01) -	**5.0002E+02(7.10E−05) +**	6.3156E+02(1.39E+01) -	6.0949E+02(1.19E+01) +	6.3901E+02(1.93E+01) -	6.2135E+02(2.65E+01)
f_{28}	6.0669E+02(2.33E+01) -	**5.0004E+02(7.65E−05) +**	5.6832E+02(3.34E+01) -	5.2442E+02(1.47E+01) +	5.1446E+02(1.76E+01) +	5.3603E+02(2.47E+01)
f_{29}	3.8514E+03(2.51E+02) -	4.7916E+03(2.48E+02) -	3.0753E+03(1.85E+02) -	3.0395E+03(2.06E+02) -	**1.3623E+03(1.68E+02) +**	2.2867E+03(2.88E+02)
f_{30}	2.8251E+04(4.99E+03) -	**4.9755E+02(1.13E+02) +**	2.4112E+03(2.73E+02) +	2.6950E+03(5.59E+02) -	2.4273E+03(1.51E+02) ≈	2.4266E+03(1.39E+02)

Table 11
Summarized results of the Wilcoxon rank-sum test.

Vs. DEDQN		$D = 10$	$D = 30$	$D = 50$	$D = 100$
jDE	+ (win)	5	0	0	0
	- (lose)	15	26	29	29
	≈ (equal)	9	3	0	0
EPSDE	+ (win)	3	2	2	4
	- (lose)	17	26	25	25
	≈ (equal)	9	1	2	0
JADE	+ (win)	4	3	2	1
	- (lose)	11	21	25	27
	≈ (equal)	14	5	2	1
SHADE	+ (win)	5	4	3	4
	- (lose)	8	18	23	24
	≈ (equal)	16	7	3	1
LSHADE	+ (win)	8	8	7	10
	- (lose)	11	11	18	11
	≈ (equal)	10	10	4	8

Table 12
Friedman rank test for all algorithms.

Algorithms	Rank	F-rank			
		$D = 10$	$D = 30$	$D = 50$	$D = 100$
jDE	1	4.05	4.87	5.03	5.06
EPSDE	2	4.56	5.13	4.96	4.89
JADE	3	3.72	4.05	3.52	3.68
SHADE	4	3.10	2.82	3.20	3.41
LSHADE	5	2.91	2.13	2.58	2.10
DEDQN	6	2.63	1.93	1.67	1.82

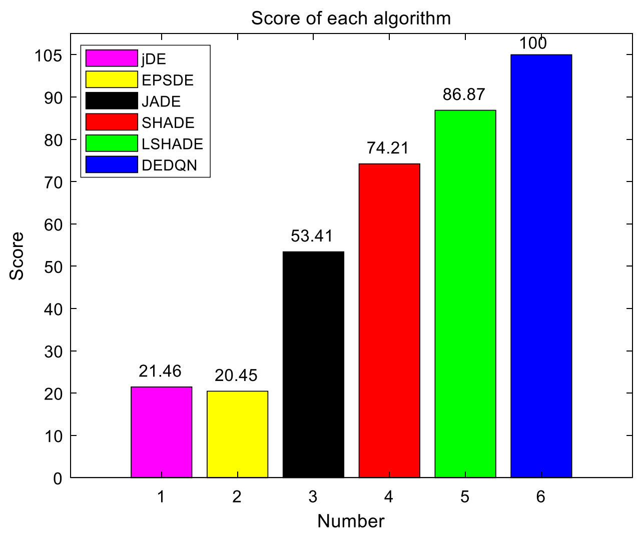

Fig. 2. The score of each algorithm.

better than other five DE variants. It also proved that the fitness landscape information and reward function can effectively train the DQN and using DQN to implement adaptive mutation strategy DE algorithm is effective. In future research, we plan to apply the DQN to realize the adaptive control parameters and combined them with the mutation strategy.

CRediT authorship contribution statement

Zhiping Tan: Methodology, Software, Writing - original draft. **Kangshun Li:** Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key R&D Program of China with Grant No. 2018YFC0831100, the National Natural Science Foundation of China with Grant No. 61773296, the Natural Science Foundation of Guangdong Province of China with the Grant No. 2020A1515010784, and the Key R&D Program of Guangdong Province, China with No. 2019B020219003.

References

[1] N. Bilel, N. Mohamed, A. Zouhaier, et al., An efficient evolutionary algorithm for engineering design problems, Soft Comput. 23 (15) (2019) 6197–6213.

[2] M. Abd Elaziz, S. Xiong, K.P.N. Jayasena, et al., Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution, Knowl.-Based Syst. 169 (2019) 39–52.

[3] N. Zeng, H. Zhang, Y. Chen, et al., Path planning for intelligent robot based on switching local evolutionary PSO algorithm, Assem. Autom. 36 (2) (2016) 120–126.

[4] W. Deng, J. Xu, Y. Song, et al., Differential evolution algorithm with wavelet basis function and optimal mutation strategy for complex optimization problem, Appl. Soft Comput. 100 (2021) 106724.

[5] H. Abedi Firouzjaee, J.K. Kordestani, M.R. Meybodi, Cuckoo search with composite flight operator for numerical optimization problems and its application in tunnelling, Eng. Optim. 49 (4) (2017) 597–616.

[6] S. Yu, S. Zhu, Y. Ma, et al., A variable step size firefly algorithm for numerical optimization, Appl. Math. Comput. 263 (2015) 214–220.

[7] L. Cui, G. Li, X. Wang, et al., A ranking-based adaptive artificial bee colony algorithm for global numerical optimization, Inform. Sci. 417 (2017) 169–185.

[8] S. Chakraborty, A.K. Saha, S. Sharma, et al., A novel enhanced whale optimization algorithm for global optimization, Comput. Ind. Eng. 153 (2021) 107086.

[9] K. Gao, Z. Cao, L. Zhang, et al., A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems, IEEE/CAA J. Autom. Sin. 6 (4) (2019) 904–916.

[10] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.

[11] Z. Meng, J.S. Pan, K.K. Tseng, PaDE: An enhanced differential evolution algorithm with novel control parameter adaptation schemes for numerical optimization, Knowl.-Based Syst. 168 (2019) 80–99.

[12] A.K. Bhandari, A novel beta differential evolution algorithm-based fast multilevel thresholding for color image segmentation, Neural Comput. Appl. 32 (9) (2020) 4583–4613.

[13] P.P. Biswas, P.N. Suganthan, G.A.J. Amaratunga, Minimizing harmonic distortion in power system with optimal design of hybrid active power filter using differential evolution, Appl. Soft Comput. 61 (2017) 486–496.

[14] L. Peng, S. Liu, R. Liu, et al., Effective long short-term memory with differential evolution algorithm for electricity price prediction, Energy 162 (2018) 1301–1314.

[15] M. Pant, H. Zaheer, L. Garcia-Hernandez, et al., Differential evolution: a review of more than two decades of research, Eng. Appl. Artif. Intell. 90 (2020) 103479.

[16] J. Liu, J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Comput. 9 (6) (2005) 448–462.

[17] Y. Tan, G.Z. Tan, T. Li, Novel chaos differential evolution algorithm, Comput. Eng. 35 (11) (2009) 216–217.

[18] J. Brest, S. Greiner, B. Boskovic, et al., Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (6) (2006) 646–657.

[19] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.

[20] R. Tanabe, A. Fukunaga, Evaluating the performance of SHADE on CEC 2013 benchmark problems, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 1952–1959.

[21] R. Tanabe, A.S. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1658–1665.

[22] A.W. Mohamed, A.A. Hadi, K.M. Jambi, Novel mutation strategy for enhancing SHADE and LSHADE algorithms for global numerical optimization, Swarm Evol. Comput. 50 (2019) 100455.

[23] N.H. Awad, M.Z. Ali, P.N. Suganthan, et al., An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 2958–2965.

[24] V. Stanovov, S. Akhmedova, E. Semenkin, LSHADE algorithm with rank-based selective pressure strategy for solving CEC 2017 benchmark problems, in: 2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2018, pp. 1–8.

[25] Q. Fan, X. Yan, Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies, IEEE Trans. Cybern. 46 (1) (2015) 219–232.

[26] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2008) 398–417.

[27] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.

[28] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, et al., Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696.

[29] G. Wu, R. Mallipeddi, P.N. Suganthan, et al., Differential evolution with multi-population based ensemble of mutation strategies, Inform. Sci. 329 (2016) 329–345.

[30] Q. Fan, X. Yan, Self-adaptive differential evolution algorithm with discrete mutation control parameters, Expert Syst. Appl. 42 (3) (2015) 1551–1572.

[31] S. Wang, Y. Li, H. Yang, et al., Self-adaptive differential evolution algorithm with improved mutation strategy, Soft Comput. 22 (10) (2018) 3433–3447.

[32] Z. Tan, K. Li, Y. Wang, Differential evolution with adaptive mutation strategy based on fitness landscape analysis, Inform. Sci. 549 (2021) 142–163.

[33] G. Wu, X. Shen, H. Li, et al., Ensemble of differential evolution variants, Inform. Sci. 423 (2018) 172–186.

[34] S.M. Elsayed, R.A. Sarker, D.L. Essam, A three-strategy based differential evolution algorithm for constrained optimization, in: International Conference on Neural Information Processing, Springer, Berlin, Heidelberg, 2010, pp. 585–592.

[35] S.M. Elsayed, R.A. Sarker, D.L. Essam, Multi-operator based evolutionary algorithms for solving constrained optimization problems, Comput. Oper. Res. 38 (12) (2011) 1877–1896.

[36] X.G. Zhou, G.J. Zhang, Differential evolution with underestimation-based multimutation strategy, IEEE Trans. Cybern. 49 (4) (2018) 1353–1364.

[37] X. Li, L. Wang, Q. Jiang, et al., Differential evolution algorithm with multi-population cooperation and multi-strategy integration, Neurocomputing 421 (2021) 285–302.

[38] S. Wright, The roles of mutation, inbreeding, crossbreeding, and selection in evolution, in: Proc. 6th Congress on Genetics, 1932, pp. 356–366.

[39] M. Tomassini, L. Vanneschi, P. Collard, et al., A study of fitness distance correlation as a difficulty measure in genetic programming, Evol. Comput. 13 (2) (2005) 213–239.

[40] K.M. Malan, A.P. Engelbrecht, Quantifying ruggedness of continuous landscapes using entropy, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 1440–1447.

[41] G. Merkuryeva, V. Bolshakovs, Benchmark fitness landscape analysis, Int. J. Simul. Syst. Sci. Technol. 12 (2) (2011) 38–45.

[42] L. Shen, J. He, A mixed strategy for evolutionary programming based on local fitness landscape, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–8.

[43] K. Arulkumaran, M.P. Deisenroth, M. Brundage, et al., Deep reinforcement learning: A brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38.

[44] K.M. Sallam, S.M. Elsayed, R.A. Sarker, et al., Multi-method based orthogonal experimental design algorithm for solving CEC2017 competition problems, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1350–1357.