Complex & Intelligent Systems
https://doi.org/10.1007/s40747-023-01243-9

ORIGINAL ARTICLE

Dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning for optimization problems

Qingyong Yang[1] · Shu-Chuan Chu[1] · Jeng-Shyang Pan[1,2] · Jyh-Horng Chou[3,4] · Junzo Watada[5]

Received: 23 May 2023 / Accepted: 9 September 2023
© The Author(s) 2023

Abstract
The introduction of a multi-population structure in differential evolution (DE) algorithm has been proven to be an effective way to achieve algorithm adaptation and multi-strategy integration. However, in existing studies, the mutation strategy selection of each subpopulation during execution is fixed, resulting in poor self-adaptation of subpopulations. To solve this problem, a dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning (RLDMDE) is proposed in this paper. By employing reinforcement learning, each subpopulation can adaptively select the mutation strategy according to the current environmental state (population diversity). Based on the population state, this paper proposes an individual dynamic migration strategy to "reward" or "punish" the population to avoid wasting individual computing resources. Furthermore, this paper applies two methods of good point set and random opposition-based learning (ROBL) in the population initialization stage to improve the quality of the initial solutions. Finally, to evaluate the performance of the RLDMDE algorithm, this paper selects two benchmark function sets, CEC2013 and CEC2017, and six engineering design problems for testing. The results demonstrate that the RLDMDE algorithm has good performance and strong competitiveness in solving optimization problems.

Keywords Differential evolution · Multi-population · Population diversity · Reinforcement learning · Individual dynamic migration

Introduction

In the real word, "optimization problems" widely exist in various fields [1–3], such as engineering design, economy, transportation, and automatic control. According to different search spaces, "optimization problems" are broadly classified into two categories. One is the function optimization problem in continuous space, and the representative problem is engineering design optimization problem [1]. The other is the combinatorial optimization problem in discrete space, and representative problems include job-shop scheduling problem (JSP) [2], knapsack problem, traveling salesman problem (TSP) [4], transportation problem, and so on.

How to design an algorithm to effectively solve the above optimization problems has become a common research focus in both academia and industry. Initially, the size of the optimization problem is small. Traditional exact algorithms, such as steepest descent method, branch-and-bound method, and simplex method are the main research objects. However, such

✉ Shu-Chuan Chu
scchu0803@gmail.com

Qingyong Yang
yang_qy@sdust.edu.cn

Jeng-Shyang Pan
jengshyangpan@gmail.com

Jyh-Horng Chou
choujh@nkust.edu.tw

Junzo Watada
junzo.watada@gmail.com

1 College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China

2 Department of Information Management, Chaoyang University of Technology, Taichung, Taiwan

3 Department of Healthcare Administration and Medical Informatics, Kaohsiung Medical University, Kaohsiung 807, Taiwan

4 Department of Mechanical and Computer-Aided Engineering, Feng-Chia University, Taichung 407, Taiwan

5 Graduate School of Information, Production and Systems, Waseda University, Kitakyushu 808-0135, Japan

Published online: 05 October 2023

algorithms usually require high characteristics of the problem to be solved. At present, with the development of science and technology, there is an increasing demand for higher solution efficiency and accuracy in algorithms [5, 6]. Therefore, in the face of large-scale complex optimization problems, these algorithms have obvious limitations. For improving the efficiency of solving the problem, some intelligent optimization algorithms are proposed. These intelligent optimization algorithms [2, 7] employ a global random search strategy with mechanisms of self-adaptation and learning evolution. Even when solving complex optimization problems with high dimensions, these algorithms can still be applied. At present, intelligent optimization algorithms have been widely used in various fields. According to different search mechanisms, these algorithms can be divided into two categories: evolutionary computation (EC) and swarm intelligence (SI) [8]. Evolutionary computation mainly includes genetic algorithm (GA) [9] and differential evolution (DE) algorithm [10, 11]. Swarm intelligence includes many kinds of algorithms [12, 13], representative ones are: ant colony optimization (ACO) [14], particle swarm optimization (PSO) [15, 16], artificial bee colony (ABC) algorithm [17], bat algorithm (BA) [18], grey wolf optimization (GWO) [19], gannet optimization algorithm (GOA) [20], etc.

In 1997, the DE algorithm [10] was proposed by Storn and Price. The algorithm is a population-based random search algorithm, which has the advantages of easy implementation, strong robustness and convergence. It has been applied with great success in different fields, including image processing, wireless sensor networks, energy, and transportation. However, the algorithm performance is greatly influenced by the setting of mutation strategy and parameters. For optimization problems in different fields, the DE algorithm usually requires trial and error to determine the most appropriate combination of parameters. This process is extremely time-consuming. Therefore, how to realize the adaptive strategy and parameters of the algorithm has gradually become a hot topic of current research [8, 21, 22]. Over the past 2 decades, the DE variants with adaptive mechanism are proposed, such as the jDE [23], JADE [24], SaDE [25], CoDE [26], MPEDE [27], and Hip-DE [28] algorithms.

In the way of realizing the DE algorithm adaptation and multi-strategy integration, the multi-population structure is a simple and effective way, and its feasibility has been successfully proved in [27, 29–31]. Multi-population is to divide the original single population into multiple subpopulations of equal (or unequal) size, and each subpopulation has different mutation strategy and parameter settings. Each subpopulation is relatively independent during execution process, and information exchanges between subpopulations will be carried out regularly. It is worth noting that although the above-mentioned multi-population DE algorithms integrate multiple mutation strategies, the selection of mutation strategies for each subpopulation in the actual execution process is fixed. This results in poor self-adaptation of the subpopulation. When dealing with some complex multi-extreme problems, the subpopulation may fall into local optimum due to the inability to adjust the mutation strategy in time. This can adversely affect the overall performance of the algorithm. Re-dividing subpopulations can change the execution of the mutation strategy, but this may lose some useful information in the population, resulting in slow convergence of the algorithm and a waste of computing resources. In summary, the current studies on the multi-population DE algorithm still focus on how to integrate different mutation strategies, parameter adaptation, and the development of communication strategies. Fewer studies, almost none, have been conducted on how to improve the self-adaptation of subpopulations.

As a machine learning algorithm, reinforcement learning [32] is used to solve the problem of maximizing returns by learning strategies during the interaction between agents and the environment. During the learning process, the agent can perceive the environment and take an action based on the current state. This action will influence the current environment, and the environment will provide feedback to the agent regarding the "cost" (punishment or reward) of the action. Repeatedly, the agent continuously improves its action strategy to maximize the expected reward or return. Figure 1 shows a schematic diagram of the interaction between the agent and the environment. At present, with the continuous development of algorithms and the improvement of computing power, reinforcement learning has demonstrated significant potential in solving complex decision-making problems [33, 34]. It has been successfully applied in various fields, including industrial control [35, 36], gaming, smart city [37], finance, robot control [38], energy management [39], and medical diagnosis [40]. Regarding the combination of reinforcement learning and intelligent optimization algorithms, literature [4, 41–49] gives relevant research. The results indicate that by introducing reinforcement learning, the adaptation and solution performance of the algorithm are improved.

To further improve the ability of multi-population DE algorithm in dealing with complex problems and improve the self-adaptation of subpopulations, a dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning (RLDMDE) is proposed in this paper. In the RLDMDE algorithm, the mutation strategy of each subpopulation will be adaptively selected through the Q-learning algorithm [50]. To this end, this paper establishes a population diversity detection mechanism to judge the state of the population. Initially, each subpopulation contains the same number of individuals. Some subpopulations may stagnate as the iteration proceeds. To avoid the waste of computing resources, this paper proposes an individual dynamic

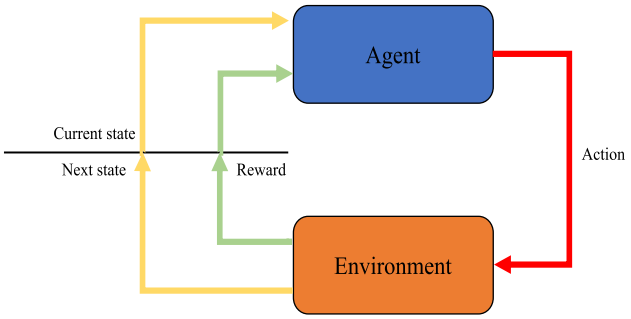


Fig. 1 The schematic diagram of the interaction between the agent and the environment

migration strategy to "reward" or "punish" subpopulations. Furthermore, to improve the quality of the initial solutions, this paper uses a combination of good point set [51] and random opposition-based learning (ROBL) [52] for population initialization. The RLDMDE algorithm is evaluated on two benchmark function sets, CEC2013 [53] and CEC2017 [54], and six real-world engineering design problems. Compared with the results of several state-of-the-art DE variants on CEC2013 and 2017, as well as the experimental results of existing literature on engineering design problems, the RLDMDE algorithm all shows strong competitiveness. This paper makes the following contributions:

1. Aiming at the problem of poor self-adaptation of subpopulations in the multi-population DE algorithm, this paper introduces the reinforcement learning to solve it. Specifically, the mutation strategy of each subpopulation is adaptively selected through the Q-learning algorithm. This is also the first study to introduce reinforcement learning to improve subpopulation self-adaptation in the multi-population DE algorithm.
2. Good point set and random opposition-based learning are used to initialize the population. The two methods are executed successively to jointly improve the quality of the initial solutions.
3. The population diversity detection mechanism is established to realize effective monitoring of population state and provide judgment basis for the reasonable decision-making of the agent.
4. An individual dynamic migration strategy based on the population state is proposed to "reward" the computational resources of the dominant subpopulation and "punish" the inferior subpopulation, avoiding the waste of individual computational resources.

The remaining sections are structured as follows: the section "Related works" briefly describes the principle of the original DE algorithm and summarizes the relevant literature on the improvement of DE algorithm in recent years. The section "Proposed algorithm" presents the principle and design details of the RLDMDE algorithm. The experimental results of the RLDMDE algorithm on two benchmark function sets, CEC2013 and CEC2017, are analyzed and discussed in the section "Analysis of experimental results. The section "Engineering design problems" is the test experiment of the RLDMDE algorithm on six real-world engineering design problems. The section "Conclusions" is the summary and prospect of this paper.

Related works

In this section, the principle of the original DE algorithm and a brief review of the relevant literature on its improvement are introduced.

The original DE algorithm

The DE algorithm mainly contains four steps: population initialization, mutation, crossover, and selection. In the population initialization stage, the DE algorithm uses a random way to initialize the population with a scale of ps and a spatial dimension of D. For an individual X_i in the population, the initialization formula is as follows:

$$X_{i,j} = lb_j + (ub_j - lb_j) \cdot rand(0, 1), \tag{1}$$

where $rand(0, 1)$ represents a random number. j is the dimension, and ub_j and lb_j denote the upper and lower bounds of the dimension, respectively.

Mutation: This stage is the core of the DE algorithm. First, the differential vector is generated by randomly or fixedly selecting several different parent individuals. Then, it is summed with the selected basis vector to produce mutant offspring of current population. The following are some common mutation strategies:

DE/rand/1:

$$V_i^l = X_{r_1}^l + F_i^l \cdot (X_{r_2}^l - X_{r_3}^l). \tag{2}$$

DE/best/1:

$$V_i^l = X_{best}^l + F_i^l \cdot (X_{r_1}^l - X_{r_2}^l). \tag{3}$$

DE/best/2:

$$V_i^l = X_{best}^l + F_i^l \cdot (X_{r_1}^l - X_{r_2}^l) + F_i^l \cdot (X_{r_3}^l - X_{r_4}^l). \tag{4}$$

DE/rand/2:

$$V_i^l = X_{r_1}^l + F_i^l \cdot (X_{r_2}^l - X_{r_3}^l) + F_i^l \cdot (X_{r_4}^l - X_{r_5}^l). \tag{5}$$

DE/current-to-best/1:

$$V_i^l = X_i^l + F_i^l \cdot (X_{best}^l - X_i^l) + F_i^l \cdot (X_{r_1}^l - X_{r_2}^l). \tag{6}$$

DE/current-to-rand/1:

$$V_i^l = X_i^l + F_i^l \cdot (X_{r_1}^l - X_i^l) + F_i^l \cdot (X_{r_2}^l - X_{r_3}^l), \tag{7}$$

where i denotes the individual number and l is the iteration number. V_i^l represents the mutant offspring. X_{best}^l is the best individual. F_i^l is the scaling factor, a real number between the interval $[0, 1]$. r_1, r_2, r_3, r_4, r_5 are integers selected randomly from the interval $[1, ps]$ and different from each other.

It should be noted that some dimensions of the mutant individual V_i^l may exceed the scope of the search space. When this happens, these dimensions need to be bounded. It is usually handled as follows:

$$V_{i,j}^l = \begin{cases} \frac{V_{i,j}^l + lb_j}{2} & \text{if } V_{i,j}^l < lb_j \\ \frac{V_{i,j}^l + ub_j}{2} & \text{if } V_{i,j}^l > ub_j. \end{cases} \tag{8}$$

Crossover: After the mutation operation, the DE algorithm recombines the mutant individual V_i^l with the parent individual X_i^l to produce the crossover individual U_i^l. There are mainly two crossover methods, one is binomial crossover and the other is exponential crossover. Currently, binomial crossover is mainly adopted in the DE algorithm. For binomial crossover, the formula is expressed as follows:

$$U_{i,j}^l = \begin{cases} V_{i,j}^l & \text{if } rand(0, 1) \le CR_i^l \text{ or } j = j_{rand} \\ X_{i,j}^l & \text{else,} \end{cases} \tag{9}$$

where CR_i^l denotes the crossover rate, a real number between the interval $[0, 1]$. j_{rand} represents a random integer on the interval $[1, D]$.

Selection: Assume that the problem is a minimization problem. If the fitness of the U_i^l is less than or equal to the X_i^l, the parent individual will be replaced with the crossover individual. Otherwise, the parent individual is still retained to the next generation. The formula of this operation is represented as follows:

$$X_{i,j}^{l+1} = \begin{cases} U_{i,j}^l & \text{if } f(U_{i,j}^l) \le f(X_{i,j}^l) \\ X_{i,j}^l & \text{else,} \end{cases} \tag{10}$$

where f represents the fitness function.

Improvement of the DE algorithm

Researchers have shown great interest in the DE algorithm since its proposal [55]. To further enhance the optimization performance, the DE algorithm has been improved from various perspectives. The literature [8] and [21] reviewed the DE variants in recent years. At present, the improvements of the DE algorithm mainly focus on the following four aspects: (1) the improvement of mutation strategy, (2) adaptive adjustment of parameters, (3) combination with other technologies, and (4) integration of multiple mutation strategies and parameter control methods.

For (1), literature [56] proposes a trigonometric DE (TDE) algorithm, which improves the mutation operation in the original DE algorithm by trigonometric mutation to speed up the convergence speed. In [24], a novel mutation strategy named "DE/current-to-pbest/1" is proposed. It is a generalization of "DE/current-to-best/1" and will help improve the convergence performance and population diversity. Literature [29] proposes the "DE/pbad-to-pbest-to-gbest/1". This strategy will be more conducive to the balance of exploration and exploitation, as well as the improvement of convergence speed. The "DE/best-random/1" is proposed in [31], which enhances the local exploitation ability through using the linear combination of the best vector and the random vector.

For (2), the DE algorithm mainly contains three parameter settings: population size ps, scaling factor F, and crossover rate CR. The detailed discussion on the setting of ps can be found in [57]. The parameter setting of F and CR has always been the research focus in the DE algorithm. The F and CR are usually fixed in the original DE algorithm, which limits the adaptation of the algorithm to different problems. To this end, literature [58] proposes a self-adaptive DE algorithm (SDE) that generates the scaling factor F through the normal distribution. Literature [23] proposes a DE algorithm with self-adapting control parameters (jDE), and presents an self-adaptive scheme design for F and CR. The JADE algorithm is proposed in [24]. The F and CR are adaptively generated by the Cauchy distribution and normal distribution, respectively. On the basis of the JADE algorithm, literature [59] proposes the SHADE algorithm. Compared with the JADE algorithm, the SHADE algorithm introduces the historical memory factor in the process of parameter update. The CoBiDE algorithm proposed in [60] adopts two Cauchy distributions to adaptively generate the F and CR.

For (3), the literature [61] proposes to combine the DE algorithm with the EDA algorithm, so that the algorithm can fully utilize the global and local search information. Literature [62] proposes the opposition-based DE (ODE) algorithm, which accelerates the convergence speed through opposition-based learning strategy. Drawing inspiration from the community concept in the PSO algorithm [63] integrates the concept of neighborhood and proposes the DEGL algorithm. Literature [64] considers multiple population neighborhood topologies, and the proposed NDE algorithm adaptively selects topologies during operation. The multi-view concept in machine learning is introduced to the DE

algorithm in [65]. Literature [66] designs a compact DE (cDE) algorithm suitable for memory constraints based on the probabilistic model. Literature [67] proposes the OLELDE-DE algorithm. The stagnation and convergence problems of the algorithm are solved by an orthogonal learning strategy, and the elite local search mechanism is employed to further enhance the convergence ability.

For (4), there have been many mutation strategies proposed so far, and these strategies have different features. For example, "DE/rand/1" and "DE/rand/2" excel in conducting global exploration, but have a weak local exploitation ability. Conversely, "DE/best/1" and "DE/best/2" excel in local exploitation, but the global exploration ability is weak. "DE/current-to-best/1" and "DE/current-to-rand/1" achieve a better balance between these two capabilities. Therefore, how to effectively combine various mutation strategies and parameter adaptive control techniques has become a hot research topic in the improvement of DE. The DE algorithm with strategy adaptation (SaDE) is proposed in [25]. Multiple mutation strategies form a strategy pool, and the parameters F and CR are adaptively tuned. Literature [26] proposes the CoDE algorithm; each individual has a parameter candidate pool and a mutation strategy candidate pool respectively. The two candidate pools are randomly combined to produce different offspring. The EPSDE algorithm proposed in [68] is similar to the CoDE algorithm, but differs in the method of offspring generation. Some scholars also propose to adopt the multi-population structure to achieve the integration of multiple strategies. In contrast to the single population structure, the multi-population structure is easier to balance the exploitation and exploration. The MPEDE algorithm is proposed in [27]. In the MPEDE algorithm, the mutation strategy is different in each subpopulation. In terms of parameter control, the MPEDE algorithm adopts the method in the JADE algorithm. Aiming at the problems existing in the algorithm, the MPEDE algorithm is impoved in [29] and [30]. The mDE-brM algorithm is proposed in [31] for large-scale optimization problems. The algorithm also adopts a multi-population structure. By adjusting the number of individual in the subpopulation reasonably, the exploration and exploitation can be balanced.

Reinforcement learning, as a machine learning algorithm, aims to maximize returns and allows agents to autonomously adjust their learning strategies through interaction with the environment. At present, reinforcement learning has been used to the parameter control of intelligent optimization algorithms. For example, the Q-learning algorithm is employed to control annealing factor and mutation rate of the simulated annealing (SA) algorithm in [47]. Literature [43] applies the Q-learning algorithm to select topological structure of the PSO algorithm. The literature [46] embeds the Q-learning algorithm into the SCA algorithm to adjust parameters $r1$ and $r3$ in the algorithm. The GA algorithm [4], the TLBO algorithm [45], and the GWO algorithm [44] are also realized and combined with reinforcement learning. The combination of reinforcement learning and single population DE algorithm can be found in [41, 42, 48, 49].

Drawing on the experience in the above articles, this paper effectively combines the multi-population DE algorithm with reinforcement learning. The mutation strategy for each subpopulation will no longer be fixed, but adaptively selected by reinforcement learning. In the next section, this paper will introduce the proposed RLDMDE algorithm in detail.

Proposed algorithm

In this section, we will introduce the principle and operation flow of the dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning (RLDMDE). The overall framework of the RLDMDE algorithm is shown in Fig. 2. The following is an introduction to the design details of the RLDMDE algorithm.

Population structure and initialization

In the population structure, the RLDMDE algorithm employs a multi-population structure. In this paper, the number of subpopulations S is set to 3 by default. Each subpopulation contains the following data structure:

$$Sub\_Pop =< Pos, eval, Qt, st, s\_record, s\_cnt, ps\_num >, \tag{11}$$

where Pos represents the set of individuals and $eval$ is the set of individual fitness values. Qt represents the Qtable of the current subpopulation, a matrix of size 3×3. In the following "Reinforcement learning", we will give a detailed definition of the Qtable. st is the state of the current subpopulation. s_record is an array that records the state of the current subpopulation, and the default initialization size is an all-zero array of $state\_max$. $state\_max$ is the maximum threshold of the subpopulation state continuously unchanged. s_cnt is a counter whose subpopulation status has not changed continuously, and is initialized to 1 by default. ps_num is the number of individuals in the current subpopulation.

The quality of initial population will influence the convergence speed and solution accuracy. At present, most algorithms adopt random initialization in the way of population initialization. The population generated by random initialization is usually uneven, which will lead to the decrease of the population diversity and the solution performance of the algorithm. To improve the quality of the initial solutions, this paper adopts the combination of good point set and ROBL methods in the way of population initialization.

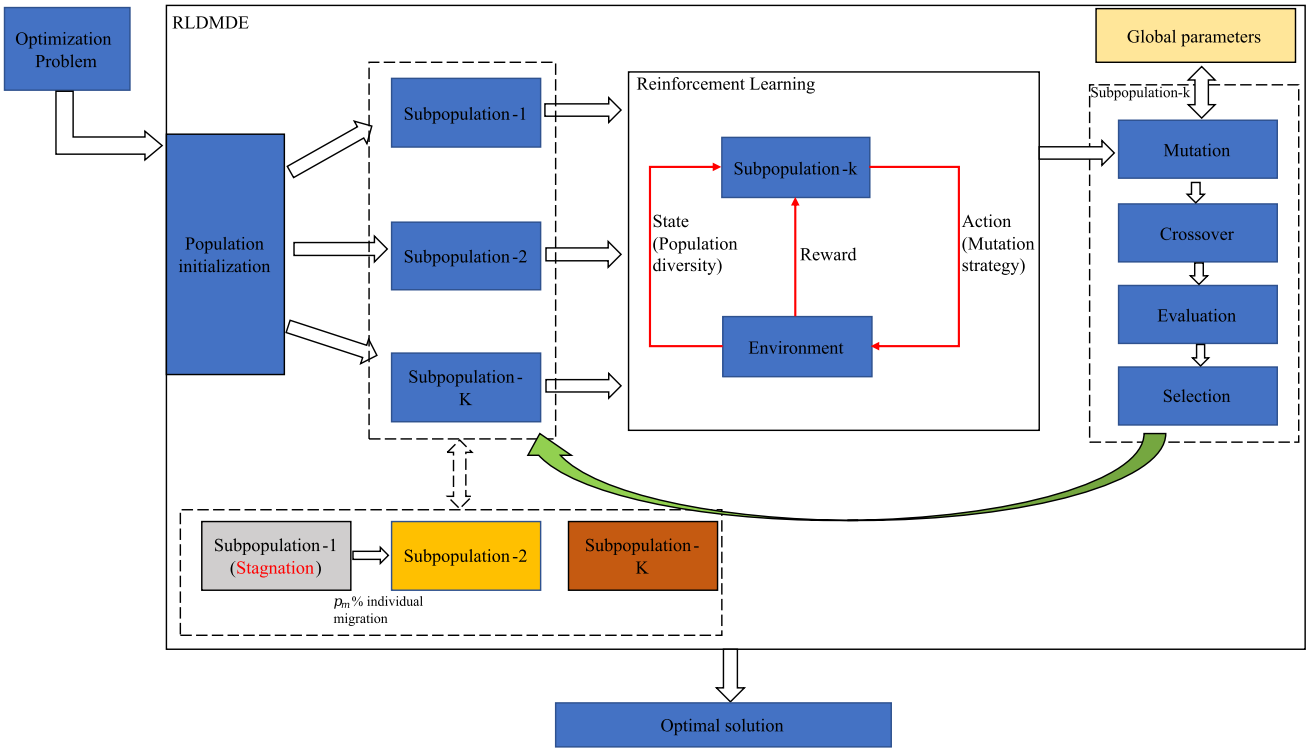


Fig. 2 The overall framework of the RLDMDE algorithm

The good point set was proposed in [51]. For an individual X_i in space, the following formula can be used for initialization:

$$X_{i,j} = lb_j + \left(ub_j - lb_j\right) \cdot mod\left(2cos\left(\frac{2\pi j}{P}\right) \cdot i, 1\right), \tag{12}$$

where P is the smallest prime number satisfying $\frac{P-3}{2} \geq D$. Figure 3 gives the comparison of the population distribution in two-dimensional space, when the population number is 100. As shown in figure, the population initialized by the good point set is more evenly distributed in space, which is beneficial to increase the diversity of the initial population and reduce the risk falling into the local optimum.

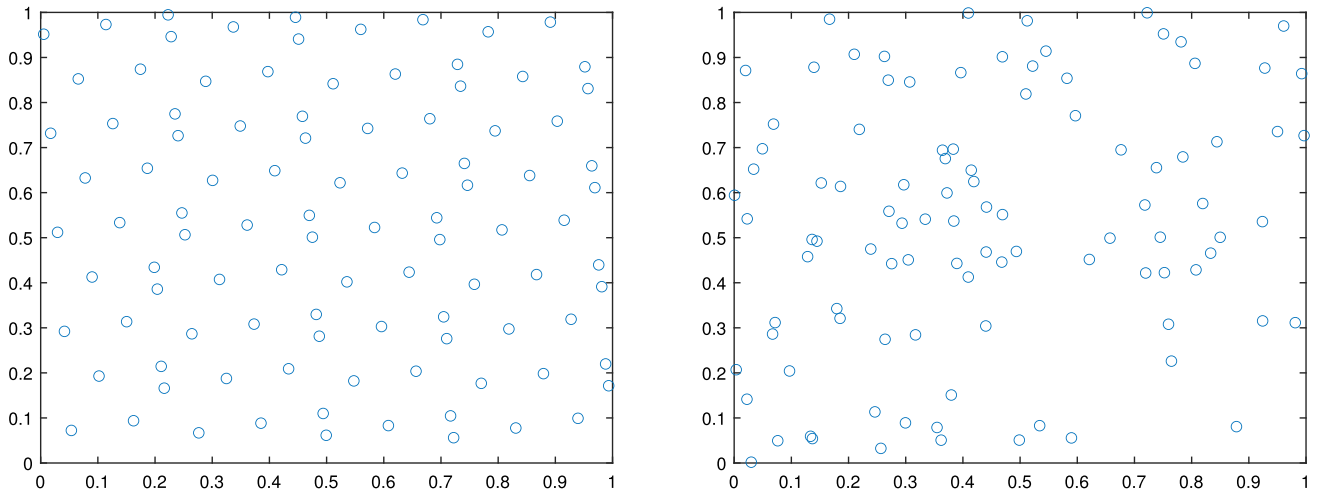
In addition, the ROBL method [52] is also introduced in the initialization stage to further enhance the population diversity. After the initialization of the good point set is completed, the ROBL method is performed on the generated individuals, the formula is as follows:

$$X_{i,j}^{op} = lb_j + ub_j - rand \cdot X_{i,j}. \tag{13}$$

By sorting the fitness values, the first ps individuals are finally selected as the initial population. Compared with the original random initialization, the advantages of the initialization method proposed in this paper are: The good point set can achieve uniform sampling of the entire search space, which helps to improve the search efficiency of the algorithm. The scope of the search can be further expanded by performing the ROBL method on the individuals generated by the good point set. By combining these two methods, the algorithm can better avoid falling into local optimal solution in the initialization stage, and help to accelerate the convergence of the algorithm.

Population diversity measurement

Population diversity reflects the state of a population at a certain moment. When the diversity is large, it means that the individuals are more scattered, and the possibility of global exploration is greater. However, the algorithm may face the difficulty of achieving effective convergence. When the diversity is low, it means that the individuals are more concentrated and the possibility of local exploitation is greater. However, there is also a risk of falling into the local optima. Therefore, intuitive and accurate measurement of the population diversity is an important prerequisite for the algorithm to adjust the optimization strategy in time. Literature [69] lists several methods for measuring population diversity, and gives the relevant experimental comparisons. The results demonstrate that the method based on the population average distance can better measure the diversity of the population. This method is also adopted in this paper. In practical prob-



(a) Good point set initialization (b) Random initialization

Fig. 3 The comparison of the population distribution in two-dimensional space (ps=100)

lems, there may be inconsistencies in the range of variables in different dimensions. Therefore, before measuring the diversity of the current population, this paper first normalizes the different dimensions of individuals. Equation 14 gives the population diversity measurement formula in this paper

$$Div = \frac{1}{ps}\sum_{i=1}^{ps}\sqrt{\sum_{j=1}^{D}(X_{i,j}^{norm} - \bar{X}_{j}^{norm})}, \tag{14}$$

where $norm$ represents the normalized flag, and $\bar{X}_j^{norm}$ represents the average value of all individuals on the jth dimension. Two boundary values of population diversity are given in [70], that is, $d_{low} = 5e-6$ and $d_{high} = 0.25$. When $Div < d_{low}$, the algorithm is in the exploitation stage. When $Div > d_{high}$, the algorithm is in the exploration stage. And when Div is between the two, the algorithm is in the balance stage. In the RLDMDE algorithm, this paper also uses these two boundary values as the basis for judging the state of the population.

Reinforcement learning

In the RLDMDE algorithm, the algorithm used in the reinforcement learning part is the Q-learning algorithm. An important component of the Q-learning algorithm is the Q-table. The Q-table includes two parts, one is the state set, and the other is the action set. According to the population diversity, there are three states for the agent. Thus, the set of states can be expressed as $State = \{s_1, s_2, s_3\} = \{d_{low}, d_{mid}, d_{high}\}$. In this paper, the action of agent is the selection of mutation strategy. The RLDMDE algorithm includes three different mutation strategies: "DE/rand/1", "DE/current-to-pbest/1" and "DE/current-to-pbest/1 (without archive)". The latter two mutation strategies are shown in Eqs. 15 and 16, respectively

Table 1 The Q-table structure (initial state)

	a_1	a_2	a_3
s_1	0	0	0
s_2	0	0	0
s_3	0	0	0

$$V_i^l = X_i^l + F_i^l \cdot (X_{pbest}^l - X_i^l + X_{r_1}^l - \widetilde{X}_{r_2}^l) \tag{15}$$

$$V_i^l = X_i^l + F_i^l \cdot (X_{pbest}^l - X_i^l) + K \cdot (X_{r_1}^l - X_{r_2}^l), \tag{16}$$

where X_{pbest}^l represents the dominant individual in the top $100p\%$ of the population. $\widetilde{X}$ indicates the individual that is randomly chosen from the union of the current population and the external archive A. A is used to store suboptimal solutions generated during the algorithm update process. K is a random number between 0 and 1. In the way of individual crossover, this paper adopts the binomial crossover method. Therefore, the action set can be expressed as $Action = \{a_1, a_2, a_3\} = \{DE/rand/1/bin, DE/current-to-pbest/1/bin, DE/current-to-pbest/1/bin\ (without\ archive)\}$. Finally, the Q-table structure is shown in Table 1.

In the choice of action, the agent adopts the $\varepsilon - greedy$ strategy. The formula is as follows:

$$a_t = \begin{cases} argmax_a Q(s_t, a), & \text{with probability } \varepsilon \\ \text{Random action}, & \text{with probability } 1-\varepsilon, \end{cases} \tag{17}$$

where t represents the moment. $Q(s_t, a)$ represents the values of all actions a in the current state s_t. The ε is a parameter less than 1. In the text, we set it to 0.8.

In addition, it also needs to construct the reward function. In reinforcement learning, the reward function usually denoted by r_{t+1} is used to represent the value of the environmental feedback after the agent performs an action. The reward function has a significant impact guiding the agent to make the right decision. Therefore, the reward function needs to be reasonably designed according to the actual usage scenario. In the RLDMDE algorithm, we associate the reward with the survival rate of offspring individuals. The survival rate is defined as follows:

$$suc\_rate = \frac{sum(I)}{ps\_num}, \tag{18}$$

where I is a binary sequence. 0 means that the parent is superior, 1 means that the offspring is superior. Then, the reward function is defined as follows:

$$r_{t+1} = \begin{cases} -suc\_rate, & \text{if } suc\_rate \leq \frac{1}{ps\_num} \\ 6 \times suc\_rate, & \text{elif} suc\_rate > \frac{1}{ps\_num} \\ & \quad \&\& \, suc\_rate \leq 0.5 \\ 8 \times suc\_rate, & \text{elif} suc\_rate > 0.5 \\ & \quad \&\& \, suc\_rate \leq 0.75 \\ 10 \times suc\_rate, & \text{else.} \end{cases} \tag{19}$$

The above formula indicates that the mutation strategy adopted by the agent makes the higher the survival rate of the offspring, the higher the reward. The lower the survival rate, the less reward the agent obtained. The effect of this setting is that in the early stage of the algorithm, the search space is relatively open, and the possibility of successfully updating individuals is relatively high. At this time, the agent can select a mutation strategy that makes the offspring have a higher survival rate based on the survival rate reward, and guide the population individuals to quickly converge to an area that is more likely to have the global optimal solution. In the later stage of the algorithm, the search space is gradually reduced, and the possibility of successfully updating individuals is also gradually reduced. At this time, the agent can choose the mutation strategy based on the survival rate reward that can keep individuals in the region where the global optimal solution may exist, and further exploit the current region. In this way, the environment can make a reasonable judgment (reward or punishment) on the action of the agent (the choice of the mutation strategy).

After each interaction, the Bellman equation is used in the Q-learning algorithm to iteratively update Table 1. The update formula is as follows:

$$Q_{t+1}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \, argmax_a Q(s_{t+1}, a) - Q(s_t, a_t)], \tag{20}$$

where α is the learning rate and γ is the discount factor, both in the range of [0, 1]. α is set to 0.2 and γ is 0.8 in this paper.

Individual dynamic migration strategy

In the initialization stage, the RLDMDE algorithm has S subpopulations with equal number of individuals. The subpopulations are independent of each other. However, with the iteration of the algorithm, some subpopulations may have evolutionary stagnation. If these subpopulations are not processed in time, it may cause a waste of computing resources. Therefore, this paper proposes an individual dynamic migration strategy.

First of all, for the judgment of stagnation phenomenon, this paper formulates the following judgment rules: when the s_cnt of the subpopulation reaches the threshold $state\_max$ (the default setting is 3) and the survival rate suc_rate is less than or equal to $\frac{1}{ps\_num}$, the current subpopulation is judged to have stagnation evolution phenomenon, and the stagnation flag $stag\_flag(i)$ is set to 1. For the situation that s_cnt does not reach the threshold, or s_cnt reaches the threshold, but the survival rate suc_rate is still high, this paper considers that the current subpopulation is less likely to stagnate, and the $stag\_flag(i)$ is set to 0. The judging criteria that the two conditions are met at the same time can better reduce the risk of misjudgment caused by the judgment of a single condition. We call the subpopulation with stagnation as "inferior subpopulation" and the subpopulation without stagnation as "dominant subpopulation". For the "inferior subpopulations", this paper divides them into two cases to deal with: (1) Only a part of the subpopulations with stagnant phenomenon. (2) All subpopulations with stagnant phenomenon.

For case 1, we "punish" the "inferior subpopulation" and "reward" the "dominant subpopulation". "Punishment" or "reward" are quantified as a decrease or increase in the number of individuals in the population. Specifically, the "inferior subpopulation" contributes part of its own resources to the "dominant subpopulation" (that is, a part of individuals migrate to the "dominant subpopulation"). The number of individuals migrated from the "inferior subpopulation" is $p_m\%$ of the total number of individuals (the default setting is 10%), and these individuals are randomly selected.

Suppose that the set of individuals selected from all the "inferior subpopulations" is represented as Mig_Pop. The "dominant subpopulation" participates in the division of Mig_Pop according to the proportion of its own number of individuals to the total number of all "dominant subpopulations". The greater the number of individuals, the more individuals can be obtained from Mig_Pop, and the more

computing resources can be obtained. Figure 4 presents a schematic diagram of individual migration in case 1.

For case 2, the subpopulations are all "inferior subpopulations". We merge all subpopulations and re-shuffle the order of individuals. The subpopulations are reallocated according to the number of individuals before the merger. Figure 5 gives a schematic diagram of individual migration in case 2.

The pseudocode for this part is shown in Algorithm 1. Step 7 in Algorithm 1 is explained as follows: Since the three mutation strategies require at least two individuals to participate. When there is only one individual left in the "inferior subpopulation" after migration, this subpopulation cannot continue to operate. Therefore, this paper chooses to clear the subpopulation directly.

Algorithm 1 Individual dynamic migration

Require: Sub_Pops, $stag\_flag$, S, p_m
Ensure: New Sub_Pops (after migration)
1: **if** $sum(stag\_flag) != S$ **then** //For case 1
2: Mig_Pop = [], mig_sum = 0;
3: **for** i=1 to S **do**
4: **if** $stag\_flag(i) == 1$ **then** //The current subpopulation is stagnant
5: $nps\_num = floor(Sub\_Pop(i).ps\_num * (1 - p_m))$; //Calculate the individuals that need to be discarded
6: **if** $nps\_num == 1$ **then**
7: mig_ct = 2; $Sub\_Pop(i).ps\_num$ = 0;
8: **else**
9: $mig\_ct = Sub\_Pop(i).ps\_num - nps\_num$;
10: **end if**
11: Randomly select mig_ct individuals from $Sub\_Pop(i)$ and add them to Mig_Pop;
12: The selected mig_ct individuals are deleted from $Sub\_Pop(i)$;
13: Update st and s_record in $Sub\_Pop(i)$;
14: Set $Sub\_Pop(i).ps\_num = nps\_num$;
15: $mig\_sum = mig\_sum + mig\_ct$;
16: **end if**
17: **end for**
18: Calculate the total number of individuals DP_sum in the "dominant subpopulations";
19: **for** i=1 to S **do**
20: **if** $stag\_flag(i) == 0$ **then**
21: Select $round(\frac{mig\_sum * Sub\_Pop(i).ps\_num}{DP\_num})$ individuals from Mig_Pop in order and add them to $Sub\_Pop(i)$;
22: Update st and s_record in $Sub\_Pop(i)$;
23: **end if**
24: **end for**
25: **else**//For case 2
26: Merge individuals in all Sub_Pops and shuffle them.
27: Reassign individuals according to the original ps_num of Sub_Pop;
28: Update st, s_record and s_cnt in all Sub_Pops;
29: **end if**

Global parameter settings

In the RLDMDE algorithm, the control parameters F and CR of each subpopulation are uniformly generated and distributed, specifically using the generation method in [59]. There is a history memory matrix M with a size of $2 \times ps$ in the algorithm to store the control parameters M_F and M_{CR} required to generate F and CR. In the initialization stage, the values in M are initialized to 0.5.

In each iteration, the individual X_i first randomly generates a random integer r_i between $1 \sim ps$, and obtains the corresponding (M_{F,r_i}, M_{CR,r_i}) from M. The F_i and CR_i are then randomly generated by the Cauchy distribution in Eq. 21 and the normal distribution in Eq. 22, respectively

$$F_i = randc(M_{F,r_i}, 0.1) \tag{21}$$

$$CR_i = randn(M_{CR,r_i}, 0.1), \tag{22}$$

where $randc(M_{F,r_i}, 0.1)$ represents the Cauchy distribution with mean M_{F,r_i} and variance 0.1. $randn(M_{CR,r_i}, 0.1)$ represents the normal distribution with mean M_{CR,r_i} and variance 0.1. When F_i is greater than 1, F_i is set to 1. And when F_i is less than or equal to 0, F_i is re-generated according to Eq. 21 until it is reasonable. Similarly, when generated CR_i is not in the interval [0, 1], CR_i is set to the value (0 or 1) close to the boundary of the interval. In each generation, the parameters F and CR that make the offspring superior to the parent are recorded in the sets S_F and S_{CR}. These two sets are used to update the content in M, and the update formulas are as follows:

$$M_{F,k,new} = \begin{cases} mean_{WL}(S_F), & \text{if } S_F \neq \emptyset \\ M_{F,k} & \text{else} \end{cases} \tag{23}$$

$$M_{CR,k,new} = \begin{cases} mean_{WA}(S_{CR}), & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k}, & \text{else,} \end{cases} \tag{24}$$

where k represents the position to be updated in M, and is initialized to 1. When the content in M is updated, k is incremented. When $k > ps$, k is reset to 1. $mean_{WL}$ is the weighted Lehmer mean, and the calculation formula is in Eq. 25. $mean_{WA}$ is the weighted arithmetic mean, and the calculation formula is in Eq. 26

$$mean_{WL}(S_F) = \frac{\sum_{i=1}^{|S_F|} \omega_i \cdot S_{F,i}^2}{\sum_{i=1}^{|S_F|} \omega_i \cdot S_{F,i}} \tag{25}$$

$$mean_{WA}(S_{CR}) = \sum_{i=1}^{|S_{CR}|} \omega_i \cdot S_{CR,i}; \tag{26}$$

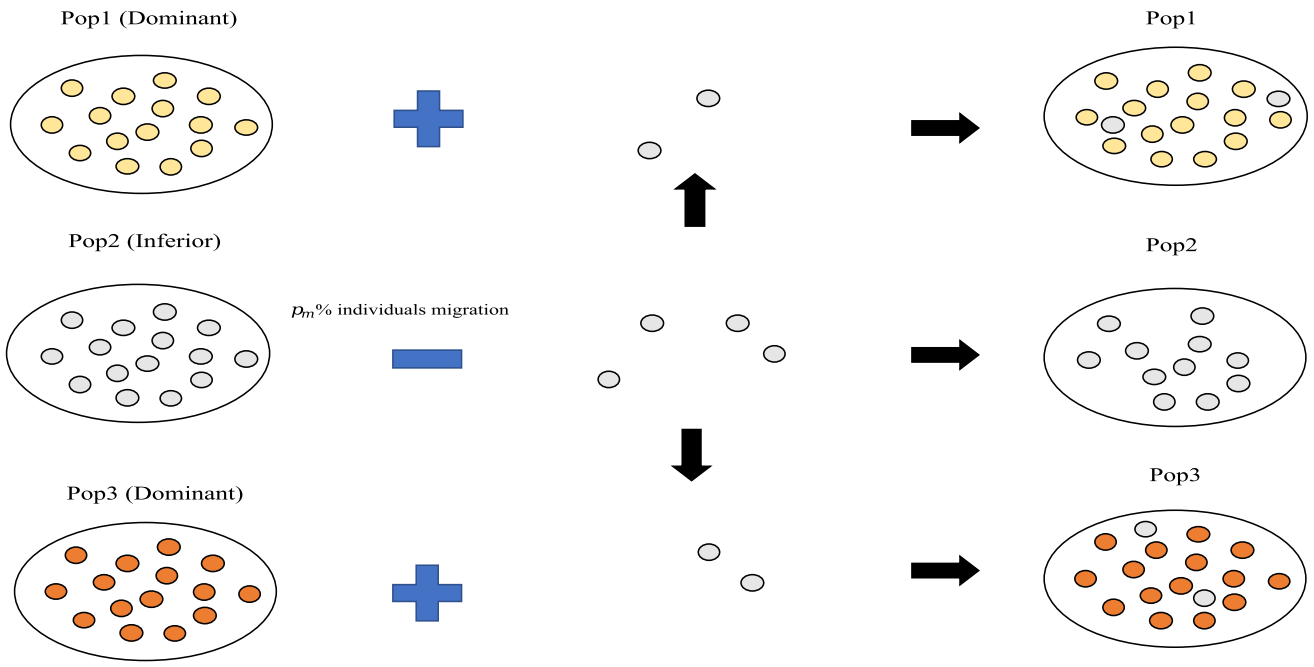


Fig. 4 The schematic diagram of individual migration in case 1

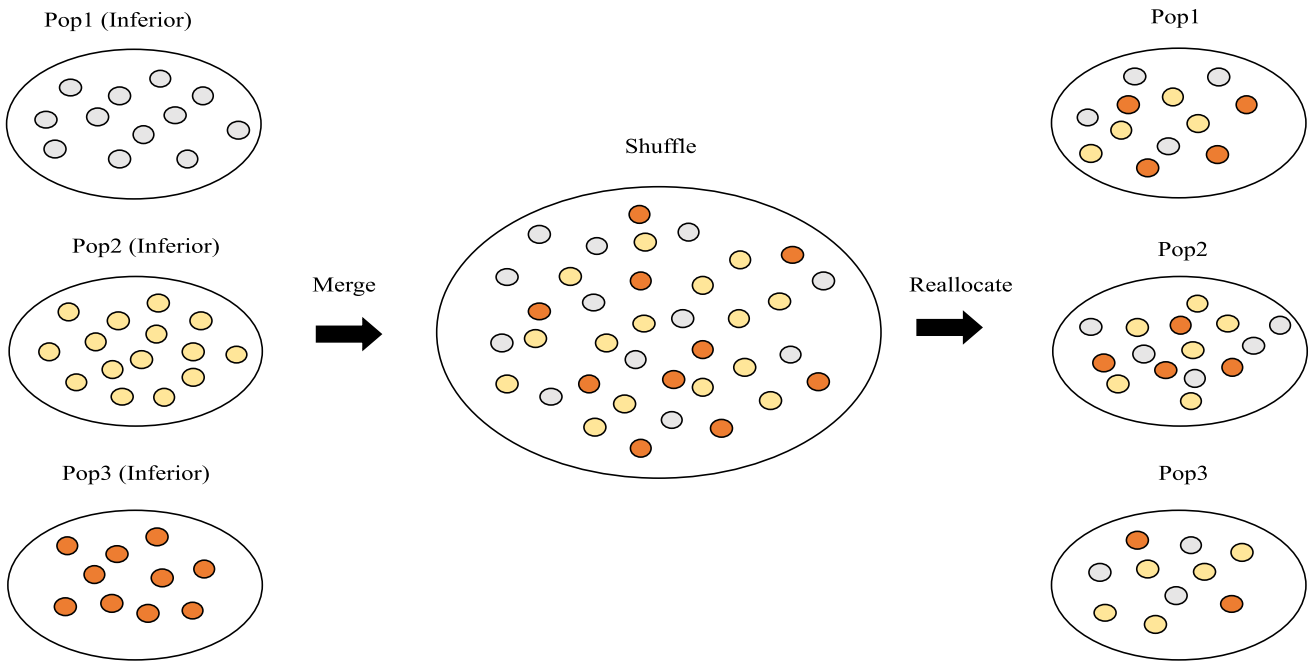


Fig. 5 The schematic diagram of individual migration in case 2

ω_i represents the weight, which is used to improve the adaptability of the parameters. The calculation formula is as follows:

$$\omega_i = \frac{|f(X_{new,i}) - f(X_i)|}{\sum_{j=1}^{|S_{CR}|} |f(X_{new,j}) - f(X_j)|}. \tag{27}$$

In addition to the parameters F and CR which are generated globally, external archive A is also global and its size is set to $2 * ps$. By setting the above parameters as global, it will be beneficial to the exchange of information among subpopulations.

Overall framework of the RLDMDE algorithm

Algorithm 2 presents the pseudocode of the RLDMDE algorithm.

Algorithm 2 Overall framework of the RLDMDE algorithm

Require: f, ps, D, nfe_max, $state\_max$, S, lb, ub, α, γ, $d_{low}^{threshold}$, $d_{high}^{threshold}$, p, p_m
Ensure: Optimal solution
1: Initialize each individual in Pop by using Eqs. 12 and 13;
2: Calculate the fitness value $f(X)$ for each individual and keep only the first ps individuals in Pop according to $f(X)$;
3: $FES = FES + 2 * ps$
4: Randomly shuffle the order of individuals in the Pop;
5: Initialize the contents in each Sub_Pop;
6: Set all values in M to 0.5 and index counter k to 1;
7: Set archive $A = \emptyset$;
8: **while** $FES \leq nfe\_max$ **do**
9: Set $S_F = \emptyset$, $S_{CR} = \emptyset$ and $tmp\_A = \emptyset$;
10: Generate the parameters F and CR for each individual by using Eqs. 21 and 22;
11: **for** i=1 to K **do**
12: Select a mutation strategy by using Eq. 17;
13: Generate offspring individuals by performing the strategy;
14: Boundary detection;
15: **for** j=1 to $Sub\_Pop.ps\_num$ **do**
16: **if** U_j is superior to $Sub\_Pop(i).Pop(j)$ **then**
17: $F_j \rightarrow S_F, CR_j \rightarrow S_{CR}$;
18: Add $Sub\_Pop(i).Pop(j)$ to tmp_A and replace $Sub\_Pop(i).Pop(j)$ with U_j;
19: **end if**
20: **end for**
21: Calculate suc_rate by using Eq. 18;
22: Calculate the reward by using Eq. 19 and the new state $Sub\_Pop(i).st$ by using Eq. 14;
23: Update $Sub\_Pop(i).s\_record$ and $Sub\_Pop(i).s\_cnt$;
24: Update $Sub\_Pop(i).Qtable$ by using Eq. 20;
25: **if** $suc\_rate \leq \frac{1}{Sub\_Pop(i).ps\_num}$ && $Sub\_Pop(i).s\_cnt == state\_max$ **then**
26: $stag\_flag(i) = 1$;
27: **else**
28: $stag\_flag(i) = 0$;
29: **end if**
30: **end for**
31: $A = A \cup tmp\_A$;
32: **if** $|A| > 2 * ps$ **then**
33: Randomly delete $2 * ps - |A|$ individuals from A;
34: **end if**
35: Update $M_{F,k}$, $M_{CR,k}$ based on S_F, S_{CR};
36: $k = k + 1$;
37: **if** $k > ps$ **then**
38: $k = 1$;
39: **end if**
40: Perform Algorithm 1;
41: $FES = FES + ps$
42: **end while**

In Algorithm 2, the time complexity of step 1 is $O(2 * ps * D)$. Assume that the time complexity of solving the fitness function $f(X)$ is T_f. The maximum time complexity of step 2 is $O(2 * ps * T_f + 4 * ps^2 + ps)$. The steps 4–6 are $O(ps)$. In the main loop, the step 10 is $O(2 * ps)$. The maximum time complexity of steps 11–30 is $O(ps * D + ps * T_f)$. The time complexity of steps 31–34 is at most $O(ps)$. The step 35 is $O(ps)$. The time complexity of Algorithm 1 is at most $O(2 * ps)$, that is, in steps 24-27. Assuming that the RLDMDE algorithm iterates $MaxIt$ times in total, the maximum time complexity is $O(MaxIt * ps * (D + T_f + ps))$.

Analysis of experimental results

To evaluate the solution performance of the RLDMDE algorithm, relevant experimental simulations are carried out in this section.

Experimental settings

In this paper, two benchmark function sets, CEC2013 and CEC2017, are selected to evaluate the RLDMDE algorithm. The CEC2013 contains 28 benchmark functions, which are divided into three types: unimodal functions $f_{a_1} - f_{a_5}$, basic multimodal functions $f_{a_6} - f_{a_{20}}$, and composition functions $f_{a_{21}} - f_{a_{28}}$. The CEC2017 contains 29 benchmark functions, including four types: unimodal functions $f_{b_1} - f_{b_3}$, simple multimodal functions $f_{b_4} - f_{b_{10}}$, hybrid functions $f_{b_{11}} - f_{b_{20}}$, and composition function $f_{b_{21}} - f_{b_{30}}$. Detailed descriptions of the two benchmark function sets are given in [53] and [54], respectively. In the experiments, the dimensions D tested in this paper are 30 and 50, respectively.

In this paper, the original DE algorithm and eight state-of-art DE variants are selected for comparison. These variants are: the jDE, SaDE, JADE, CoDE, CoBiDE, SinDE [71], SHADE, and MPEDE algorithms. All the algorithms compared adopt the parameter settings in the original text. In the RLDMDE algorithm, the population size ps is 120.

All algorithms run on the same PC. The CPU model is AMD 5800x, and its basic frequency is 3.80 GHz. The memory is 32 GB. The simulation software is Matlab 2019b. Each algorithm is run 30 times independently. The maximum number of function evaluations is $10000 \times D$. Five performance evaluation indicators are chosen: the mean, standard deviation, Wilcoxon rank-sum test, Friedman test, and algorithm convergence. The average performance and stability of the algorithm can be effectively reflected by calculating the mean and standard deviation of $(f(x) - f(x^*))$. The $f(x^*)$ is the known optimal value and the $f(x)$ is the optimal value solved by the algorithm. The Wilcoxon rank-sum test and the Friedman test can test the difference of the experimental data, which can make a further verification of the statistical conclusions drawn from the mean and standard deviation. The convergence of the algorithm is used to evaluate the convergence speed and whether the algorithm can find the optimal solution within a limited number of iterations.

CEC2013 test comparison

Tables 2 and 3, respectively, record the results of the RLDMDE algorithm and the other nine algorithms on the CEC2013 of 30D and 50D, respectively. The best means are bolded. On 30D, except for the SinDE algorithm and SHADE algorithm, the other algorithms obtain the optimal value of f_{a_1} 30 consecutive times. On f_{a_5}, only the SaDE algorithm and the CoDE algorithm achieve the optimal value. The solution results of the remaining algorithms are close to the optimal solution, and there is a gap between 1e–14 and 1e–13. On $f_{a_{11}}$, the RLDMDE, jDE, JADE, CoDE, and SHADE algorithms achieve the optimal value 30 consecutive times. On 50D, the SaDE algorithm achieves the solution of the optimal value on both f_{a_1} and f_{a_5}. The CoDE algorithm and the CoBiDE algorithm achieve the solution of the optimal value on f_{a_1}. As can be seen from the tables, the capacity of the RLDMDE algorithm in solving the optimal value is not particularly outstanding, but the solution results are relatively close to the optimal solution. To easily reflect the comparison of the RLDMDE algorithm with other algorithms on different function types, this paper organizes the data in Tables 2 and 3. The number of mean values obtained by the RLDMDE algorithm on different function types that are superior to other algorithms is counted. Tables 4 and 5 present the sorted results. The content in brackets indicates that the obtained mean values are equal.

Unimodal functions $f_{a_1} - - f_{a_5}$: In the comparison of 30D, except for the MPEDE algorithm, the number of RLDMDE algorithm winning the other algorithms exceeds half of the total number of unimodal functions. Moreover, the RLDMDE algorithm performs superiorly on unimodal functions than the SinDE algorithm. In the comparison of 50D, the RLDMDE algorithm wins the number of other algorithms are more than half of the total number of unimodal functions. On 30D, the number of the RLDMDE algorithm winning the MPEDE algorithm is the same as the number of failures. However, on 50D, the RLDMDE algorithm outperforms than the MPEDE algorithm in all the unimodal functions.

Basic multimodal functions $f_{a_6} - - f_{a_{20}}$: In the comparison of 30D, the number of the RLDMDE algorithm winning the jDE algorithm is the same as the number of failures. Compared with the SHADE algorithm, the RLDMDE algorithm performs poorly, only performs well on 5 functions. In particular, there is a clear gap between the RLDMDE algorithm and the SHADE algorithm on the functions $f_{a_{12}}$, $f_{a_{13}}$, $f_{a_{16}}$, and $f_{a_{18}}$. Except for the jDE algorithm and the SHADE algorithm, the number of RLDMDE algorithm winning the other algorithms exceeds half of the total number of basic multimodal functions. In the comparison of 50D, the RLDMDE algorithm is only weaker than the SHADE algorithm. In comparison with the DE, jDE, SaDE, JADE, CoDE, and MPEDE algorithms, the RLDMDE algorithm wins more than 10.

Composition functions $f_{a_{21}} - f_{a_{28}}$: In the comparison of 30D, the number of RLDMDE algorithm winning the other algorithms exceeds half of the total number of composition functions. In the comparison of 50D, the RLDMDE algorithm is only slightly weaker than the DE algorithm and the SinDE algorithm.

On the whole, the performance of the RLDMDE algorithm on CEC2013 is superior to other algorithms.

CEC2017 test comparison

To further verify the RLDMDE algorithm, this paper selects the CEC2017 benchmark function set for further testing. Tables 6 and 7 record the experimental results of all algorithms on the CEC2013 of 30D and 50D, respectively. The best means are also bolded. On 30D, only the CoBiDE algorithm achieves the optimal value of the function f_{b_1} 30 consecutive times. On the function f_{b_2}, only the CoDE algorithm achieves the solution of the optimal value 30 consecutive times. The RLDMDE algorithm achieves the optimal value of the function f_{b_9} 30 consecutive times. In addition to the RLDMDE algorithm, the jDE, JADE, CoDE, and CoBiDE algorithms are also archived. On all the test functions of 50D, all algorithms failed to achieve the optimal values 30 consecutive times. It can be seen that compared with the CEC2013, the CEC2017 is more complex and can better test the solving capability of the algorithm. Similarly, to more intuitively reflect the comparison on CEC2017, the data in Tables 6 and 7 are further organized. The sorted results are shown in Tables 8 and 9.

Unimodal functions f_{b_1} and f_{b_3}: In the comparison of 30D, the RLDMDE algorithm performs worse than the CoDE algorithm and the CoBiDE algorithm on the two test functions, but better than the DE, jDE, JADE, SinDE, and SHADE algorithms. In comparison with the SaDE algorithm and the MPEDE algorithm, the RLDMDE algorithm performs superiorly on only one of the test functions. On the 50D, except for the MPEDE algorithm, the RLDMDE algorithm outperforms other algorithms.

Simple multimodal functions $f_{b_4} - - f_{b_{10}}$: In the comparison of 30D, the number of RLDMDE algorithm winning most of the algorithms exceeds half of the total number of the multimodal functions. Similarly, the RLDMDE algorithm performs better than most algorithms on 50D. In particular, the RLDMDE algorithm outperforms the MPEDE algorithm on both 30D and 50D. However, in the comparison of 30D and 50D, the RLDMDE algorithm is weaker than the SHADE algorithm.

Hybrid functions $f_{b_{11}} - - f_{b_{20}}$: The optimization ability of the RLDMDE algorithm on 30D is weaker than the CoBiDE algorithm. It is only superior to the CoBiDE algorithm on three functions. However, on 50D, the RLDMDE algorithm outperforms the CoBiDE algorithm in most functions. The

Table 2 The experimental results of the RLDMDE algorithm and the other nine algorithms on CEC2013 (30D)

Function	RLDMDE		DE		jDE		SaDE		JADE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
F2	7673.869	7073.605	390740.6	274103.9	95411.6	75617.53	40788.06	21540.35	3624.609	3480.904
F3	27.67604	98.40216	**0.06382**	0.258592	827680.4	2091189	2860109	3897534	136526.2	745665.1
F4	**1E-07**	4.02E-07	910.7417	348.3019	17.35061	17.17037	392.1764	401.047	5291.348	12095
F5	9.85E-14	3.93E-14	1.1E-13	2.08E-14	1.1E-13	2.08E-14	**0**	0	1.06E-13	2.88E-14
F6	4.046662	5.272733	7.666626	3.904423	11.69455	3.051303	5.609019	5.264872	0.880247	4.821313
F7	0.199348	0.211917	**0.18318**	0.283144	1.641253	1.547428	20.43662	12.58891	5.195523	3.311599
F8	20.9367	0.062454	20.95234	0.051797	20.95642	0.050415	20.94578	0.062536	20.88087	0.135108
F9	25.90942	4.065292	27.79463	13.04951	21.90296	7.308399	16.51053	2.519177	26.49767	1.560485
F10	0.055529	0.033376	**0.00649**	0.006316	0.041949	0.017707	0.142407	0.080463	0.033012	0.012465
F11	**0**	0	126.0051	25.10501	**0**	0	0.165827	0.377138	**0**	0
F12	24.32468	5.153781	177.4473	10.50752	54.58408	12.55739	41.15808	9.28395	22.23457	3.283567
F13	45.43827	9.290076	182.4221	8.737509	92.98914	16.889	87.78075	20.82758	52.75832	14.56415
F14	**0.00167**	0.005278	6298.105	542.6208	0.00694	0.009982	0.974171	1.691969	0.024983	0.021456
F15	3087.171	359.3086	7079.573	294.9724	5275.904	507.0884	4407.927	1030.849	3260.403	263.3511
F16	1.186742	0.458968	2.480306	0.263506	2.501818	0.363763	2.270451	0.293685	1.745097	0.638779
F17	30.43395	0.000579	185.9644	16.48258	30.43375	7.61E-14	30.45723	0.055757	**30.4337**	0
F18	81.28568	6.664406	206.8887	15.64162	159.5518	14.10107	134.3916	16.66673	76.22419	7.447294
F19	1.33465	0.121062	15.28872	1.010605	1.641445	0.126569	3.49323	0.567216	1.45207	0.13504
F20	10.51679	0.44277	12.12501	0.227848	11.71045	0.259351	10.689	0.473408	10.21971	0.477315
F21	280	40.68381	303.4936	89.80214	**278.278**	106.7844	312.0422	76.40164	284.7848	50.39836
F22	105.5127	0.766665	6208.041	540.2598	107.6725	8.586577	129.7722	74.03304	**84.8555**	37.2131
F23	3185.148	399.8391	7010.851	303.0305	5247.679	377.7204	4591.232	1091.616	3502.229	393.0923
F24	202.9385	6.042102	200.0245	0.022894	208.3509	6.356945	220.4147	5.894894	210.144	11.10682
F25	263.5795	21.94022	**238.474**	4.656375	254.9956	10.91614	259.7121	14.54754	278.5524	11.87141
F26	200.0006	0.000471	203.3793	18.39728	200.0092	0.003418	200.0024	0.001226	212.5167	38.17981
F27	315.3996	25.88633	**300.955**	0.945039	394.616	62.0858	555.3975	69.22266	588.4253	255.1039
F28	**260**	81.36762	300	2.43E-13	300	0	300	0	300	0
Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	**0**	0	**0**	0	1.52E-14	5.77E-14	2.27E-14	6.94E-14	**0**	0
F2	71875.5	39752.42	56271.08	33044.79	2580807	1134643	5990.004	4051.652	**82.754**	441.5255
F3	601709.1	1164567	45.45126	203.2603	4079.478	7921.421	1974.811	9442.099	25.10008	93.15958
F4	0.160704	0.444244	6.71E-05	0.000124	4455.091	1817.776	7.57E-07	1.2E-06	7.83E-05	0.000225
F5	**0**	0	1.14E-13	0	1.14E-13	0	1.17E-13	2.08E-14	1.14E-13	0
F6	3.93937	8.968667	**0.51609**	0.478407	14.44744	2.260814	1.760495	6.699777	0.895355	4.818722
F7	10.65091	9.477041	1.984157	1.636435	0.345799	0.442877	3.630477	3.478337	2.465793	2.35653
F8	**20.7211**	0.1545	20.94014	0.053225	20.95551	0.03669	20.81879	0.214308	20.92633	0.055718
F9	13.69185	3.014962	**10.0857**	2.500623	14.71684	2.948727	28.35642	1.562629	14.8139	6.975509
F10	0.038763	0.02618	0.020765	0.016509	0.021771	0.018146	0.022578	0.013502	0.024615	0.015526
F11	**0**	0	1.14E-14	2.31E-14	0.165893	0.377108	**0**	0	9.47E-15	2.15E-14
F12	35.15518	10.37724	37.07877	10.6743	30.86524	6.461968	**14.8243**	2.344374	23.01671	6.926531
F13	74.85099	22.93048	86.7672	27.7236	73.51881	16.84434	**32.152**	11.05267	38.63935	19.59777
F14	3.570567	3.064675	237.561	54.54546	47.10418	16.26202	0.013879	0.016703	9.562838	3.879246
F15	3563.735	576.0579	2934.392	411.9892	**2802.02**	608.8984	3166.672	213.7834	4412.939	431.2478

Table 2 continued

Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F16	**0.36819**	0.210332	0.578972	0.918298	1.725899	0.284056	0.8603	0.189053	2.477974	0.267783
F17	30.43712	0.007119	37.36129	1.191426	33.76568	0.958072	30.43375	7E-14	30.5366	0.05091
F18	66.06858	10.04513	66.59657	8.254508	77.84062	13.27777	**62.9819**	4.098177	97.20676	13.57519
F19	1.59736	0.333454	2.654827	1.301702	2.362344	0.457468	**1.16453**	0.111146	2.004328	0.251021
F20	10.59559	0.655127	10.55245	0.666715	**10.0708**	0.508763	10.703	0.92558	10.55986	0.405444
F21	306.3966	100.7244	353.2235	104.171	282.9205	62.24356	295.8059	72.37164	314.9451	88.69527
F22	114.2693	20.82253	424.6076	221.4301	161.6233	57.80699	88.54706	35.46943	125.2957	10.45095
F23	3479.598	726.5187	3096.548	613.0512	**2940.05**	753.4862	3249.458	474.5509	4490.764	559.0816
F24	221.7645	9.702429	205.6482	6.806114	**200.011**	0.016639	212.7734	5.683281	207.0634	5.824742
F25	255.168	8.58041	247.6735	6.278686	242.8651	4.973513	247.5858	13.62807	245.6016	7.272399
F26	200.0075	0.004156	200.0048	0.002237	210.1466	30.58867	203.4816	19.06754	**200**	1.69E-05
F27	604.0956	106.6581	420.2322	102.2965	317.274	62.46495	474.1778	204.1555	376.1325	31.54364
F28	300	0	300	2.49E-13	300	8.44E-14	300	0	300	0

The bold data is the optimal value in the same set of data

Table 3 The experimental results of the RLDMDE algorithm and the other nine algorithms on CEC2013 (50D)

Function	RLDMDE		DE		jDE		SaDE		JADE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	8.34E-14	1.11E-13	1.74E-13	9.78E-14	1.06E-13	1.15E-13	**0**	0	1.89E-13	8.62E-14
F2	37067.98	18210.81	3206951	1186604	491430.7	164895.4	184851.5	69368.95	24789.48	9404.973
F3	95706.09	251994.4	321227.3	706572.9	3817030	4670680	37385146	47973333	1417827	2039412
F4	**5.9E-05**	0.00016	19234.77	3554.67	201.5526	141.8568	1127.942	819.4609	9001.071	20632.34
F5	1.14E-13	0	1.29E-13	3.93E-14	1.14E-13	0	**0**	0	1.14E-13	0
F6	**42.4675**	5.36694	43.4555	0.044755	43.82136	0.358402	50.43107	17.08356	43.82599	1.441048
F7	3.565763	2.899001	1.026805	1.221371	14.33187	6.694541	45.93501	8.735878	21.5977	8.038448
F8	21.12626	0.040369	21.12853	0.031642	21.11506	0.041049	21.13327	0.044618	21.13728	0.022919
F9	50.43636	8.015244	68.86412	9.879944	51.13736	5.850104	35.60872	4.11354	54.25578	1.611857
F10	0.083911	0.054527	0.030389	0.014388	0.049886	0.028349	0.177276	0.090121	0.037836	0.022349
F11	0.397984	1.032009	191.5249	33.62418	3.79E-14	2.73E-14	1.989918	1.652537	**1.9E-15**	1.04E-14
F12	45.62631	8.287562	351.252	14.38765	96.53432	21.44732	115.6471	21.0603	52.28026	5.93498
F13	102.735	18.54445	351.0821	13.61053	187.4478	24.51676	236.5825	37.36882	122.5903	25.74932
F14	0.002498	0.005082	11582.85	1151.227	**0.00084**	0.003169	7.610209	4.715629	0.037891	0.026343
F15	6746.537	429.3029	13798.24	399.3441	9820.898	536.0974	8838.988	2324.388	7026.022	428.2464
F16	1.419008	0.160576	3.378045	0.27767	3.256904	0.314726	2.899641	0.340581	2.513471	0.965069
F17	50.78576	1.7E-11	331.1733	30.34035	50.78576	8.65E-14	52.6008	1.688565	**50.7858**	4.09E-14
F18	143.6193	9.68078	400.1115	15.56463	284.3129	18.09915	203.7071	80.69916	141.1826	8.925575
F19	**2.12534**	0.324963	29.85726	2.002755	2.794239	0.253273	10.53654	1.958494	2.793087	0.185793
F20	19.84179	0.588482	22.09652	0.325563	21.46366	0.300042	19.93112	0.764533	19.98151	0.401997
F21	622.1298	334.3058	476.6557	429.823	608.2746	376.5632	913.5271	312.4098	930.3946	346.1719
F22	**11.1563**	1.707721	11506.14	961.0052	14.48145	6.450672	22.234	4.871282	19.96234	41.98481
F23	7057.177	729.1178	13618.43	404.6993	9345.139	786.8686	8518.721	1940.866	7045.086	407.4315
F24	215.8044	12.7491	207.5828	8.034687	243.7677	10.45406	263.7753	8.099607	252.2239	21.8932
F25	327.7288	33.18483	**274.586**	4.424191	322.2591	26.45286	337.2823	10.95782	356.8061	20.86362
F26	**229.218**	54.01213	258.0217	59.07067	239.0606	72.70003	249.9724	77.82441	360.0283	97.21795
F27	655.4646	199.305	**553.18**	149.5564	926.5177	115.7745	1110.384	102.2657	1356.809	284.9259
F28	**400**	2.84E-13	**400**	2.89E-13	**400**	2.89E-13	**400**	4.22E-14	**400**	2.89E-13

Table 3 continued

Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	std	Mean	Std
F1	**0**	0	**0**	0	2.27E-13	0	2.27E-13	0	1.74E-13	9.78E-14
F2	224159	88041.03	338644.2	144838.7	4102908	1495132	**20106.5**	9624.316	104026.2	58327.05
F3	12060527	14055667	2095187	2467282	**19798.9**	51135.04	830577	1948957	636164.1	1780484
F4	0.199979	0.309894	0.114252	0.076406	5926.13	1736.01	0.000112	0.000211	0.938127	2.760689
F5	4.55E-14	5.66E-14	1.14E-13	0	1.17E-13	2.08E-14	2.35E-13	5.11E-14	1.36E-13	4.63E-14
F6	43.8261	1.441444	43.44733	4.49E-10	43.44733	5.59E-12	43.44733	8.39E-14	43.44733	5.37E-14
F7	38.02451	11.69559	16.48454	7.060128	**0.97728**	0.808868	16.45129	6.872083	16.54625	7.120854
F8	**20.988**	0.08715	21.122	0.035879	21.1272	0.035948	20.99226	0.18164	21.14749	0.024485
F9	30.88897	4.544436	**24.0379**	4.341081	33.4806	5.565598	56.07823	2.651113	33.01781	5.824215
F10	0.055661	0.030691	0.052548	0.026063	0.073175	0.039699	0.029549	0.020298	**0.02708**	0.017301
F11	0.762802	0.854033	9.29E-09	2.72E-08	7.073014	3.794868	6.44E-14	4.15E-14	5.31E-14	1.44E-14
F12	91.03858	15.91445	81.95131	19.11064	53.47449	12.12316	**39.3046**	6.43189	57.87338	13.15057
F13	206.9988	49.72409	162.6014	41.61449	148.719	39.76145	**99.8633**	25.03972	134.096	36.58498
F14	28.95758	15.05483	822.8368	170.6573	192.0782	76.45419	0.01957	0.01418	6.085847	2.972347
F15	7031.15	767.5291	**6242.25**	568.5572	6308.569	766.9879	6618.572	513.9885	8859.387	655.4692
F16	0.946313	0.443166	**0.40358**	0.829022	2.026602	0.368148	1.219966	0.230242	3.138469	0.56508
F17	52.476	0.617487	86.0181	3.747059	65.82929	2.261638	50.78576	3.91E-14	50.8831	0.045083
F18	121.5596	19.34598	**117.357**	16.1975	134.6988	22.6857	119.3813	7.196147	169.1758	34.28728
F19	3.06036	0.473179	4.544706	1.08218	4.997338	0.675455	2.171231	0.160561	3.475293	0.416819
F20	19.9641	0.739611	19.9283	0.551016	19.57987	0.841133	19.43375	0.683273	**19.2333**	0.560365
F21	880.6	363.9602	**459.771**	382.9555	642.0433	454.9951	859.3908	384.591	795.7466	434.3321
F22	40.64752	34.44765	934.6788	269.5724	321.8909	240.6339	11.90889	4.123601	20.07802	4.594437
F23	7548.167	848.9299	6520.255	967.7547	**6090.77**	853.7984	7161.589	690.7698	8517.021	1406.732
F24	260.3064	13.57365	229.0554	9.054142	**206.262**	11.4651	245.506	12.62557	242.0547	7.655961
F25	314.745	9.244415	295.9417	11.59258	279.4169	10.74243	308.6763	27.49584	304.7915	12.54552
F26	273.2557	92.0006	267.3666	78.74911	280.5174	58.30331	285.1457	80.31712	254.3807	68.05572
F27	1101.932	114.5084	881.5218	124.2249	643.0968	193.3202	1002.756	313.5365	802.626	105.6984
F28	**400**	1.88E-13	**400**	3.1E-13	**400**	2.89E-13	499.0294	542.4064	**400**	2.89E-13

The bold data is the optimal value in the same set of data

Table 4 The statistical results of mean values on CEC2013 (30D)

RLDMDE vs	DE	jDE	SaDE	JADE	CoDE	CoBiDE	SinDE	SHADE	MPEDE
Unimodal	3 (1)	4 (1)	3 (1)	3 (1)	3 (1)	4 (1)	5	4	2 (1)
Basic multimodal	13	11 (1)	14	7 (1)	8 (1)	9	10	5 (1)	9
Composition	5	6	7	7	7	6	5	6	6
sum	21 (1)	21 (2)	24 (1)	17 (2)	18 (2)	19 (1)	20	15 (1)	17 (1)

Table 5 The statistical results of mean values on CEC2013 (50D)

RLDMDE vs	DE	jDE	SaDE	JADE	CoDE	CoBiDE	SinDE	SHADE	MPEDE
Unimodal	5	4 (1)	3	3 (1)	3	3 (1)	4	4	5
Basic multimodal	13	10	14	11	10	8	9	5	11
Composition	3 (1)	5 (1)	7	6 (1)	6	4 (1)	3 (1)	7	6 (1)
sum	21 (1)	19 (2)	24	20 (2)	19	15 (2)	16 (1)	16	22 (1)

Table 6 The experimental results of the RLDMDE algorithm and the other nine algorithms on CEC2017 (30D)

Function	RLDMDE		DE		jDE		SaDE		JADE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	4.26E-15	6.62E-15	9.47E-15	6.81E-15	1.09E-14	6.11E-15	5.07E-14	1.53E-13	8.53E-15	7.08E-15
F3	5.31E-14	1.44E-14	17.93882	18.42589	0.000606	0.000611	1.89E-15	1.04E-14	5234.379	11283.62
F4	14.51479	25.221	59.48753	2.105935	56.99459	15.54124	**2.42414**	1.995273	57.19787	15.26023
F5	20.4145	5.794316	179.5415	7.409426	38.98813	5.251978	30.92203	7.05414	25.6433	4.582354
F6	1.14E-13	0	3.38E-08	3.65E-08	1.14E-13	0	**8.7E-14**	4.89E-14	1.74E-13	5.77E-14
F7	52.85506	2.794852	207.3403	12.91756	78.49453	6.207572	61.19361	6.865	56.17405	2.275234
F8	23.19015	2.751217	178.9345	10.64146	42.63938	4.978507	33.86174	7.429435	25.85883	3.237398
F9	**0**	0	3.79E-15	2.08E-14	**0**	0	0.989674	1.571606	**0**	0
F10	1879.882	271.6145	6646.474	316.1159	2879.47	208.4449	3517.193	656.8246	**1616.22**	343.7277
F11	20.6737	24.60777	57.10735	28.0157	28.56138	24.52173	79.59794	31.3367	19.5039	23.30677
F12	1055.105	365.1219	6809.779	6864.008	9831.729	6522.028	6468.399	4798.446	1501.186	2147.618
F13	24.38026	8.7402	78.38903	7.532278	36.35192	20.8261	225.3976	158.8638	50.70673	22.78708
F14	22.93292	7.427426	60.07233	8.408954	31.71	7.748896	80.40932	26.38417	4402.741	7609.502
F15	11.00512	7.149868	37.74187	5.728167	11.66355	3.731885	68.98668	39.08961	261.715	847.3528
F16	290.3947	129.0164	646.2456	405.3919	470.2168	125.4402	411.1706	122.647	407.0076	130.6341
F17	51.84013	11.405	72.32743	7.486749	90.7954	15.6813	44.74792	21.44724	70.47896	16.67958
F18	37.89808	27.20851	35.88533	4.472894	48.62916	25.90379	580.5203	667.2484	30794.32	60162.73
F19	7.09262	3.12034	18.34499	6.732649	11.10395	2.216765	43.83959	34.819	530.6327	1569.698
F20	71.57564	36.75192	**26.8287**	18.34397	71.20601	36.99076	78.97638	65.11419	75.84672	32.74989
F21	220.6597	3.03816	364.9582	8.595357	244.5085	6.681304	233.4456	7.238559	227.3514	3.478761
F22	**100**	0	100	1.39E-13	**100**	0	100.082	0.44909	**100**	0
F23	364.7897	4.290136	517.8914	13.02284	386.8538	4.249006	380.9298	10.36412	371.9069	5.241864
F24	437.2594	2.351445	590.8731	8.40078	456.5141	6.516837	452.1695	11.00605	440.6769	3.156612
F25	**386.624**	0.870194	386.7437	0.029664	386.9847	0.123739	387.687	2.313583	387.0331	0.140319
F26	**387.09**	288.8126	2470.584	128.8341	1329.203	83.37738	1171.985	474.2232	1228.881	55.51944
F27	508.2077	7.224896	**486.441**	9.943677	501.7613	4.193101	514.7472	11.58371	503.0079	4.891757
F28	**310.329**	31.51541	311.0412	33.72882	332.0546	49.87986	324.9816	46.1553	322.4387	45.68058
F29	464.8752	26.67748	535.4911	109.1123	489.9126	25.08944	449.6064	41.25202	472.1489	43.62622
F30	2020.582	100.1925	1999.114	42.93951	2199.494	153.5556	2785.05	519.495	2127.271	188.0938
Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	1.89E-15	6.17E-15	**0**	0	0.000689	0.002833	1.23E-14	4.91E-15	2.84E-15	5.78E-15
F3	**0**	0	3.79E-15	1.44E-14	668.4231	342.4947	9.47E-14	4.8E-14	7.28E-11	3.98E-10
F4	36.2241	29.04988	21.3537	27.22855	84.58248	5.929721	59.30234	1.920908	55.58343	15.25323
F5	39.43349	13.5101	36.31596	14.81704	29.58758	8.884748	**15.3605**	2.592417	21.96631	7.315782
F6	4.79E-07	2.62E-06	7.56E-09	1.19E-08	1.14E-13	0	3.39E-05	2.62E-05	4.56E-09	2.5E-08
F7	67.13924	8.881164	62.57794	11.45812	61.95948	7.395511	**43.2501**	3.167063	57.82457	5.279546
F8	38.4717	13.37003	37.90788	10.08025	31.706	9.602956	**14.9052**	2.46451	27.14392	6.877392
F9	**0**	0	**0**	0	1.14E-14	3.47E-14	0.026858	0.047896	0.002984	0.016346
F10	2066.186	434.9267	1849.364	440.8042	1984.379	348.4697	1712.773	214.004	2741.63	525.4591
F11	20.02259	17.18177	**15.2368**	13.02387	19.72949	22.64677	31.1383	24.3056	20.92864	4.981519
F12	8953.585	9540.348	4419.081	3182.124	68992.67	71750.37	1473.305	556.6526	**838.684**	364.3554
F13	44.52133	66.24442	24.47026	5.194729	3861.231	5252.671	40.86555	18.67087	**23.5495**	4.138144

Table 6 continued

Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F14	14.04727	9.393586	**11.2796**	4.543256	26.61127	8.002817	28.4168	4.992913	13.04785	8.471267
F15	10.30525	4.688705	9.213777	2.540469	13.64196	2.91268	36.41821	18.56999	**8.82035**	5.982645
F16	517.8784	222.3698	402.178	110.5252	156.8088	149.4665	**156.667**	141.0896	362.1006	179.7319
F17	58.03971	62.39935	38.33347	31.73579	**28.2086**	13.75443	39.32818	9.567678	59.36625	24.22483
F18	81.73716	64.36783	**18.8078**	8.049053	46508.28	27837.43	87.7451	52.83068	24.09752	6.347772
F19	5.254418	1.798514	**5.16786**	1.415021	10.06753	7.795437	9.862089	4.823316	7.869423	2.578664
F20	93.92713	75.02646	28.25631	46.07853	43.13958	51.24525	62.87866	47.076	104.164	61.4346
F21	240.9141	10.89958	239.0405	8.635263	231.7173	6.492594	**215.27**	3.401629	226.8224	7.519953
F22	332.9373	733.8145	100	2.12E-13	174.6617	408.9389	100	1.15E-13	**100**	0
F23	388.5997	9.488985	383.741	6.760113	377.8108	8.648974	**363.968**	4.082627	373.5985	8.448559
F24	459.4519	14.50289	462.5115	10.47602	447.0229	6.501837	**436.296**	3.492505	442.8974	6.600032
F25	386.8818	0.109993	386.7849	0.039334	386.9387	0.10147	386.8458	0.084509	386.8103	0.096998
F26	1356.227	241.3359	1408.681	106.8844	1182.481	77.91864	1074.148	49.39711	1223.05	58.75878
F27	499.6198	8.464133	498.5315	11.08121	499.8424	5.485087	506.8852	4.616257	500.5683	5.538704
F28	334.9082	55.60037	370.0464	54.35742	347.4247	54.06531	334.0991	53.64156	322.3453	46.2582
F29	437.1435	65.38029	**424.265**	57.88758	430.101	15.10531	465.4686	19.122	454.6343	35.03227
F30	2082.91	109.6357	2041.313	83.02162	4070.288	1314.933	2100.499	113.9355	**1986.11**	37.29502

The bold data is the optimal value in the same set of data

Table 7 The experimental results of the RLDMDE algorithm and the other nine algorithms on CEC2017 (50D)

Function	RLDMDE		DE		jDE		SaDE		JADE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	3.6E-14	8.12E-15	28.11261	100.8006	2.05E-08	3.74E-08	18.85343	44.25269	4.55E-14	1.42E-14
F3	**1.9E-13**	5.24E-14	61309.29	8419.1	136.2856	180.8423	0.000569	0.001094	40155.62	44005.93
F4	**27.6647**	27.4105	64.91599	50.90048	88.20806	55.73712	56.04679	52.35068	74.08637	48.83123
F5	37.4207	6.037492	349.4809	14.85604	89.35658	8.487933	81.71919	13.69143	55.1158	5.9507
F6	6.44E-07	2.56E-06	8.67E-08	3.11E-07	**1.2E-13**	2.08E-14	0.001172	0.003708	1.78E-13	6.46E-14
F7	94.44616	7.833686	404.3199	11.8471	142.9865	11.40162	138.6818	18.89137	101.9361	7.849223
F8	41.5999	7.278953	352.6003	10.2823	85.57021	8.692207	91.13808	15.13351	55.72724	6.476966
F9	0.014921	0.033936	0.015144	0.082948	0.125557	0.34023	44.5097	60.00282	0.970505	1.22996
F10	3418.755	284.6303	12949.1	298.6591	4922.219	444.2855	5867.963	1747.47	3714.753	285.9286
F11	61.4412	16.42016	139.4023	17.18496	49.68454	12.82307	109.2619	35.27331	129.0307	33.11919
F12	7120.36	4474	63711.53	41652.65	48797.76	26379.9	25292.31	17913.16	4510.273	2271.873
F13	87.3076	49.03502	287.5948	113.4034	1463.051	1653.283	1583.076	1565.902	190.7489	91.36678
F14	137.0403	47.76403	126.0374	9.130635	63.587	10.86839	337.4601	291.3848	18590.07	56217.45
F15	106.1136	63.00766	109.0196	7.138596	108.9201	107.5782	1821.894	2092.542	334.7743	146.5046
F16	797.8246	119.5761	2012.108	922.6336	905.2115	139.5935	751.1414	210.9196	810.983	182.8447
F17	516.3341	102.6961	1063.009	448.3275	541.344	251.9419	533.7098	164.6793	674.4007	150.3499
F18	158.8883	114.5996	357.5446	288.1741	2775.185	2162.538	4414.025	3822.37	42003.35	132808.2
F19	48.92319	30.89182	58.0738	9.495558	29.21662	9.038508	96.61498	98.67339	838.1947	3850.512
F20	302.3955	114.3085	1027.526	541.845	495.5481	102.044	327.6611	152.316	490.6934	100.1695
F21	244.5757	9.463037	551.5664	17.01802	290.794	10.80103	274.9775	13.34228	255.3933	6.21859
F22	**390.362**	1080.242	10051.33	5590.225	4852.158	1916.911	4120.06	3132.227	3562.568	1407.676
F23	464.8014	6.872193	774.9606	16.34502	506.1684	11.02595	517.1131	18.19424	470.0512	9.438129
F24	**529.345**	5.08476	836.4264	13.67718	572.7392	7.987079	581.224	19.72451	544.1432	9.463718

Table 7 continued

Function	RLDMDE		DE		jDE		SaDE		JADE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F25	525.1202	33.17963	498.0479	31.8203	512.5055	36.06414	559.5492	29.93527	521.8151	33.39645
F26	**521.271**	452.1432	4144.599	500.1131	1898.171	145.1256	2343.21	267.9206	1602.698	72.79597
F27	545.8584	40.47316	**508.184**	9.281189	527.0946	8.979278	668.4756	52.30999	554.1357	19.30296
F28	485.5056	21.86595	462.0959	12.35758	479.3722	23.98695	493.8845	33.32323	481.5108	24.34646
F29	409.3386	38.01841	689.4259	425.2362	553.1817	89.25343	494.8773	117.396	437.7078	63.08334
F30	621531.7	32665.49	**585043**	13388.58	603441.8	29197.42	792658.8	101029.1	683534.8	67302.62
Function	CoDE		CoBiDE		SinDE		SHADE		MPEDE	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
F1	50.40768	92.43353	7.41911	39.00561	3759.243	4096.995	3.74E-14	1.26E-14	**2.9E-14**	5.88E-15
F3	1.51E-09	3.42E-09	1.36E-07	1.83E-07	15045.23	3291.645	3.07E-13	6.78E-14	0.000344	0.001098
F4	51.67492	47.7825	46.74822	37.0942	74.35058	51.64027	74.11088	36.62761	38.33505	35.4356
F5	76.57359	18.01057	92.46468	15.89123	60.50654	13.51114	**31.29**	4.784637	48.48762	16.03454
F6	1.95E-06	9.7E-06	1.22E-08	1.63E-08	1.44E-13	5.11E-14	0.000619	0.000987	0.000769	0.002142
F7	133.9227	21.46185	130.7368	15.48493	114.2928	11.07362	**80.5793**	3.198864	113.1539	11.41751
F8	79.05973	14.82012	91.93241	14.09511	63.8006	12.9684	**35.1775**	4.709164	53.30316	10.53948
F9	3.243011	4.992361	0.048417	0.182864	**1.1E-13**	2.08E-14	1.222689	1.023696	0.7943	1.064735
F10	4343.258	748.6193	4052.016	747.7449	3993.231	658.3093	**3401.14**	307.6668	4620.835	467.34
F11	57.16136	13.08909	52.51977	7.907488	**40.4142**	5.169811	127.6557	23.96425	102.1064	23.8673
F12	43216.52	35091.16	64834.49	33815.24	886696.4	589123.6	**4414.97**	4320.074	8141.905	5405.832
F13	3986.19	4105.648	902.2821	1638.53	2032.255	2516.998	304.7215	189.4963	**65.9486**	31.93455
F14	127.2183	327.5055	**43.4116**	9.474833	9495.774	10160.16	230.348	62.89703	54.89233	11.43724
F15	213.3266	425.2419	**59.5664**	34.60444	709.5079	938.7836	303.7877	118.9208	64.27842	21.71624
F16	1129.757	299.9913	901.8153	253.2272	**731.159**	220.8543	752.4204	142.6397	1009.448	280.8078
F17	672.9963	237.8833	557.7212	156.0198	**394.88**	170.6898	438.089	140.2959	627.7105	183.4088
F18	2687.514	1872.949	1248.057	723.2764	218683.2	196048.4	162.7607	57.00214	**155.641**	113.0315
F19	43.36731	49.3806	**17.5534**	8.217429	7017.606	6124.53	152.433	43.70661	36.67746	16.9332
F20	467.818	226.0714	423.635	198.1754	**287.069**	148.8479	335.7998	113.0359	435.4211	158.4682
F21	275.572	15.37629	276.5383	16.87864	264.1342	10.97803	**231.798**	5.245361	263.9047	17.0246
F22	4754.322	1514.712	4052.211	1473.301	4444.209	1023.968	3527.553	1377.129	3390.489	2560.793
F23	509.4284	21.17428	500.2422	20.3887	476.3939	10.64451	**457.045**	5.874695	485.476	10.58032
F24	571.1511	20.25455	578.2409	14.48838	551.5469	9.769824	531.6262	5.007641	546.1819	17.64234
F25	523.69	38.92651	511.2471	26.84189	**485.829**	20.92222	512.222	37.51369	536.8478	35.1114
F26	1967.427	152.1698	1970.293	215.1482	1688.55	143.7725	1382.337	87.76138	1519.557	124.9289
F27	544.3163	17.82313	534.1889	17.05278	509.3984	10.41134	552.6595	27.06436	546.4978	25.72347
F28	487.6983	22.4913	465.1783	16.9909	**461.83**	11.39576	494.026	21.71314	493.0993	21.35125
F29	510.054	182.4092	580.6698	161.4164	**343.698**	21.48804	449.8259	82.85957	411.8922	70.66085
F30	597535.5	28488.32	601588.1	22451.29	662750	49258.38	683398.5	79366.03	698743.2	98640.09

The bold data is the optimal value in the same set of data

performance of the RLDMDE algorithm and the MPEDE algorithm on 30D and 50D is relatively close. Compared with the remaining algorithms, the RLDMDE algorithm demonstrates superior performance overall, winning most of the functions on both 30D and 50D.

Composition function $f_{b21} - - f_{b30}$: In the comparison of 30D and 50D, the number of RLDMDE algorithm winning the other algorithms exceeds half of the total number of combination functions. In solving all the combination functions on 50D, the RLDMDE algorithm outperforms the SaDE algorithm and the MPEDE algorithm.

In summary, the RLDMDE algorithm exhibits superior performance in solving the CEC2017 compared to other algorithms.

Table 8 The statistical results of mean values on CEC2017 (30D)

RLDMDE vs	DE	jDE	SaDE	JADE	CoDE	CoBiDE	SinDE	SHADE	MPEDE
Unimodal	2	2	1	2	0	0	2	2	1
Simple multimodal	7	5 (2)	5	5 (1)	6 (1)	5 (1)	6 (1)	3	7
Hybrid	8	9	9	9	6	3	6	7	5
Composition	8	8 (1)	9	8 (1)	8	8	8	6	6 (1)
sum	25	24 (3)	24	24 (2)	20 (1)	16 (1)	22 (1)	18	19 (1)

Table 9 The statistical results of mean values on CEC2017 (50D)

RLDMDE vs	DE	jDE	SaDE	JADE	CoDE	CoBiDE	SinDE	SHADE	MPEDE
Unimodal	2	2	2	2	2	2	2	2	1
Simple multimodal	6	6	7	6	7	6	5	3	7
Hybrid	9	7	9	9	7	6	6	7	5
Composition	6	6	10	8	7	6	6	7	10
sum	23	21	28	25	23	20	19	19	23

Wilcoxon rank-sum and Friedman tests

In this paper, two non-parametric test methods, namely Wilcoxon rank-sum test and Friedman test, are used to make further statistical analysis of the experimental results in the previous two subsections. Significance level α in Wilcoxon rank-sum test is set to 0.05. When the obtained $pvalue$ is less than α, it implies that there is a significant difference between the compared data. The statistical results of the two tests are shown in Tables 10 and 11. The symbols $+, -, \approx$ in Table 10 indicate that the compared algorithm is superior to, worse than, or similar to the RLDMDE algorithm, respectively. From the statistical results in Tables 10 and 11, the performance of the RLDMDE algorithm is consistent with the conclusions analyzed in the previous two subsections. The RLDMDE algorithm can achieve superior solutions on two different benchmark function sets, CEC2013 and CEC2017, and is superior to the original DE algorithm and its variants.

It is worth mentioning that although both the RLDMDE algorithm and the MPEDE algorithm are multi-population algorithms, the RLDMDE algorithm shows superior performance compared to the MPEDE algorithm, especially on the multi-extreme problems. The main reason is that the mutation strategy of each subpopulation in the RLDMDE algorithm is not fixed. With the help of the Q-learning algorithm, each subpopulation can choose an appropriate mutation strategy based on the environmental state, thus improving the capability to escape from a local optimum. And thanks to the establishment of the population state monitoring mechanism, the judgment of population stagnation will be more accurate, which will help to adjust the algorithm strategy in time.

Table 10 The statistical results of Wilcoxon rank-sum test

vs RLDMDE	Symbol	CEC2013		CEC2017	
		30D	50D	30D	50D
DE	$\approx$	5	4	3	3
	$-$	19	20	23	21
	$+$	4	4	3	5
jDE	$\approx$	8	8	5	7
	$-$	18	17	23	18
	$+$	2	3	1	4
SaDE	$\approx$	5	4	4	6
	$-$	21	20	20	23
	$+$	2	4	5	0
JADE	$\approx$	9	8	7	7
	$-$	13	16	20	22
	$+$	6	4	2	0
CoDE	$\approx$	8	3	8	7
	$-$	15	16	15	19
	$+$	5	9	6	3
CoBiDE	$\approx$	6	5	5	4
	$-$	17	15	15	19
	$+$	5	8	9	6
SinDE	$\approx$	6	8	7	3
	$-$	15	13	17	19
	$+$	7	7	5	7
SHADE	$\approx$	10	6	7	7
	$-$	8	13	14	15
	$+$	10	9	8	7
MPEDE	$\approx$	5	2	14	9
	$-$	15	21	11	16
	$+$	8	5	4	4

Table 11 The statistical results of Friedman test

Algorithm	CEC2013				CEC2017			
	30D		50D		30D		50D	
	Mean ranking	Actual ranking	Mean ranking	Actual ranking	Mean ranking	Actual ranking	Mean ranking	Actual ranking
RLDMDE	3.660714286	1	3.696428571	1	3.224137931	1	3.068965517	1
DE	7.178571429	10	7.196428571	10	7.275862069	10	7.206896552	9
jDE	6.392857143	8	5.696428571	6	6.810344828	9	6.034482759	7
SaDE	7	9	7.107142857	9	6.724137931	8	7.448275862	10
JADE	5.017857143	3	5.946428571	8	5.775862069	5	5.689655172	6
CoDE	5.321428571	6	5.785714286	7	6	6	6.413793103	8
CoBiDE	5.267857143	5	4.696428571	3	4.24137931	3	5.172413793	5
SinDE	5.571428571	7	5.267857143	5	6.068965517	7	5.068965517	4
SHADE	4.571428571	2	4.446428571	2	4.724137931	4	4.275862069	2
MPEDE	5.017857143	3	5.160714286	4	4.155172414	2	4.620689655	3

Convergence analysis

In this subsection, this paper verifies and analyzes the convergence performance (speed and effect) of the RLDMDE algorithm. Figures 6, 7, 8 show the convergence curves of all algorithms on the unimodal functions, basic multimodal functions, and composition functions in CEC2013, respectively. The experimental test dimension is 30D.

Unimodal functions $f_{a_1} - - f_{a_5}$: Compared to the original DE algorithm and the jDE algorithm, the RLDMDE algorithm exhibits superior convergence speed and effect. In most functions, the JADE algorithm and the SHADE algorithm demonstrate superior convergence speed to the RLDMDE algorithm. Compared with the SaDE algorithm, on functions $f_{a_2} - - f_{a_4}$, the convergence speed of the RLDMDE algorithm is slower, but the late convergence effect is superior. Compared with the CoDE algorithm and the CoBiDE algorithm, the RLDMDE algorithm is only weak in the early convergence on function f_{a_2}. The SinDE algorithm outperforms the RLDMDE algorithm in the convergence speed on functions f_{a_1} and f_{a_5}. However, on the residual functions, the RLDMDE algorithm exhibits more dominant convergence speed and effect. In comparison to the MPEDE algorithm, the RLDMDE algorithm only performs poorly on the late convergence effect of the function f_{a_2}.

Basic multimodal functions $f_{a_6} - - f_{a_{20}}$: On function f_{a_6}, the RLDMDE algorithm is roughly consistent with other algorithms in the early convergence speed. In the later convergence effect, the JADE algorithm and the SHADE algorithm perform superiorly. Similarly, on functions f_{a_7}, $f_{a_{11}} - - f_{a_{13}}$, $f_{a_{16}} - - f_{a_{19}}$, most of the algorithms keep the same convergence speed in the early stage. In the comparison of the late convergence effect, the RLDMDE algorithm is only weaker than the SHADE algorithm on function $f_{a_{13}}$. On function f_{a_8}, except for the JADE algorithm, the remaining algorithms remain roughly the same in the early convergence speed. In the late convergence effect, the CoDE algorithm performs best. On function f_{a_9}, the CoDE algorithm performs best in convergence speed and effect. On function $f_{a_{10}}$, the SHADE algorithm performs superiorly in the convergence speed, while the DE algorithm performs superiorly in the late convergence effect. The performance of the RLDMDE algorithm on these three functions is not outstanding. On functions $f_{a_{14}}$, $f_{a_{15}}$ and $f_{a_{20}}$, the RLDMDE algorithm is not outstanding in the convergence speed, but the late convergence effect is superior to most algorithms.

Composition functions $f_{a_{21}} - - f_{a_{28}}$: The RLDMDE algorithm is consistent with most algorithms in the early convergence speed. In the late convergence effect, the RLDMDE algorithm outperforms most algorithms. For example, on function $f_{a_{25}}$, the RLDMDE algorithm shows similar convergence speed to other algorithms in the early stage, but its convergence effect in the later stage outperforms other algorithms.

In summary, the performance of the RLDMDE algorithm is not outstanding in the comparison of the early convergence speed, and there is no obvious gap with most algorithms. However, in the comparison of the late convergence effect, the RLDMDE algorithm has certain advantages and can achieve higher solution accuracy.

Parameter analysis

This subsection analyzes the effect of different values of the population number ps and the population state threshold $state\_max$ on the performance of the RLDMDE algorithm. The experimental results are shown in Tables 12 and 13,

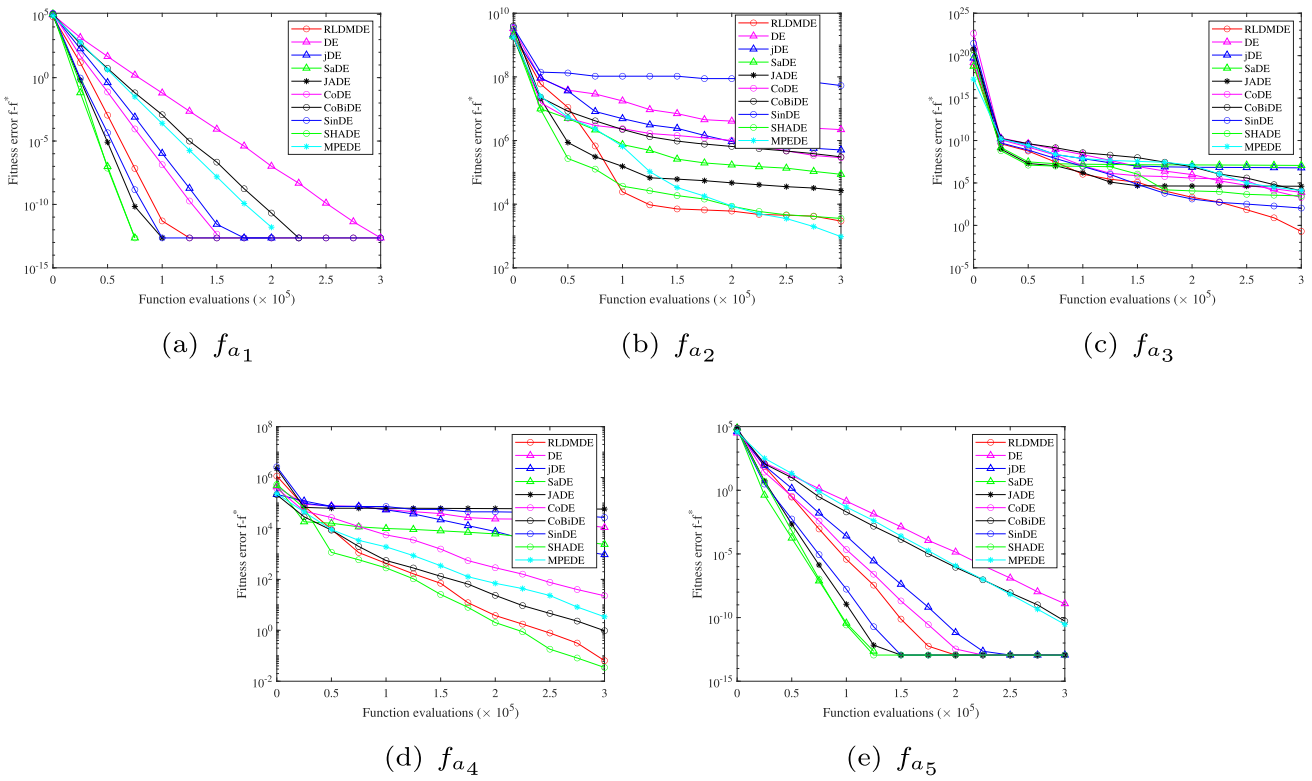


Fig. 6 The convergence curves of the RLDMDE and other algorithms solving unimodal functions $f_{a_1} - - f_{a_5}$

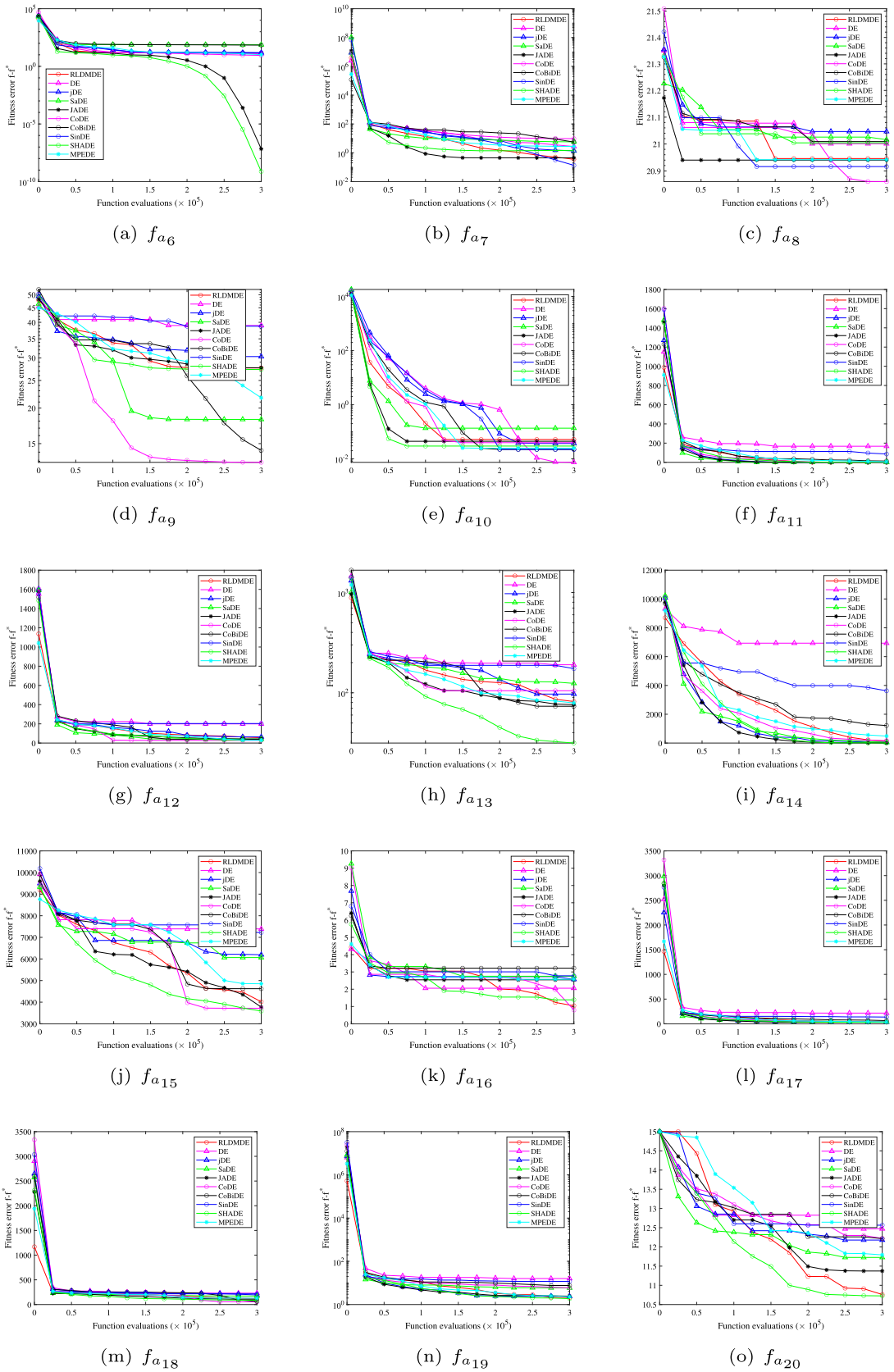
respectively. The data in the tables are the mean values of the algorithm after 30 consecutive runs.

First of all, the experimental comparison of different ps is analyzed. In the performance of the unimodal functions, it generally shows that the larger the ps, the better the performance. Therefore, it can be seen that the RLDMDE algorithm with $ps = 120$ outperforms algorithms with values of 30, 60, and 90, but weaker than algorithms with values of 150, 180 and 210. However, on the basic multimodal functions, the larger the ps, the performance of the algorithm decreases. On the composition functions, the RLDMDE algorithm performs poorly when the ps is smaller (less than or equal to 90). When the ps is larger (greater than or equal to 120), the performance of the RLDMDE algorithm is similar. On the whole, the RLDMDE algorithm with the $ps = 120$ has the best overall performance.

Next, the experimental comparison of different $state\_max$ is analyzed. On the unimodal functions, the performance of the $state\_max$ with value of 3 is weaker than that with values of 5 and 6, and is close to the performance of other values. However, on basic multimodal functions, the $state\_max$ with value of 3 performs better overall than other values. The performance of values 4 and 5 is relatively close to that of 3. On the composition functions, the performance of the RLDMDE algorithm with $state\_max$ value of 3 is close to that with value of 9 and superior to the remaining values. On the whole, the RLDMDE algorithm with $state\_max$ value of 3 performs best, followed by 4 and 5. Therefore, the value of $state\_max$ should not be too small (less than 2) or too large (greater than 6). This will cause the performance of the algorithm to degrade due to frequent individual migration or the lack of execution of migration strategy for a long time.

Engineering design problems

In this section, six real-world engineering design problems are selected to test the ability of the RLDMDE algorithm in solving practical application problems. The parameters of the RLDMDE algorithm are the default parameter settings, and the experimental environment is the same as in the section "Experimental settings". The algorithm executed 30 times continuously. In addition to the mean, standard deviation, and optimal solution during the running process, the optimal and worst values are also recorded. The optimal value can show the best performance of the algorithm. The worst value can reflect the feasibility and acceptability of the algorithm in solving practical application problems in the worst case. The following is a mathematical model description of these six engineering design problems. The best values are bolded.



(a) f_{a_6} (b) f_{a_7} (c) f_{a_8}

(d) f_{a_9} (e) $f_{a_{10}}$ (f) $f_{a_{11}}$

(g) $f_{a_{12}}$ (h) $f_{a_{13}}$ (i) $f_{a_{14}}$

(j) $f_{a_{15}}$ (k) $f_{a_{16}}$ (l) $f_{a_{17}}$

(m) $f_{a_{18}}$ (n) $f_{a_{19}}$ (o) $f_{a_{20}}$

Fig. 7 The convergence curves of the RLDMDE and other algorithms solving basic multimodal functions $f_{a_6} - - f_{a_{20}}$

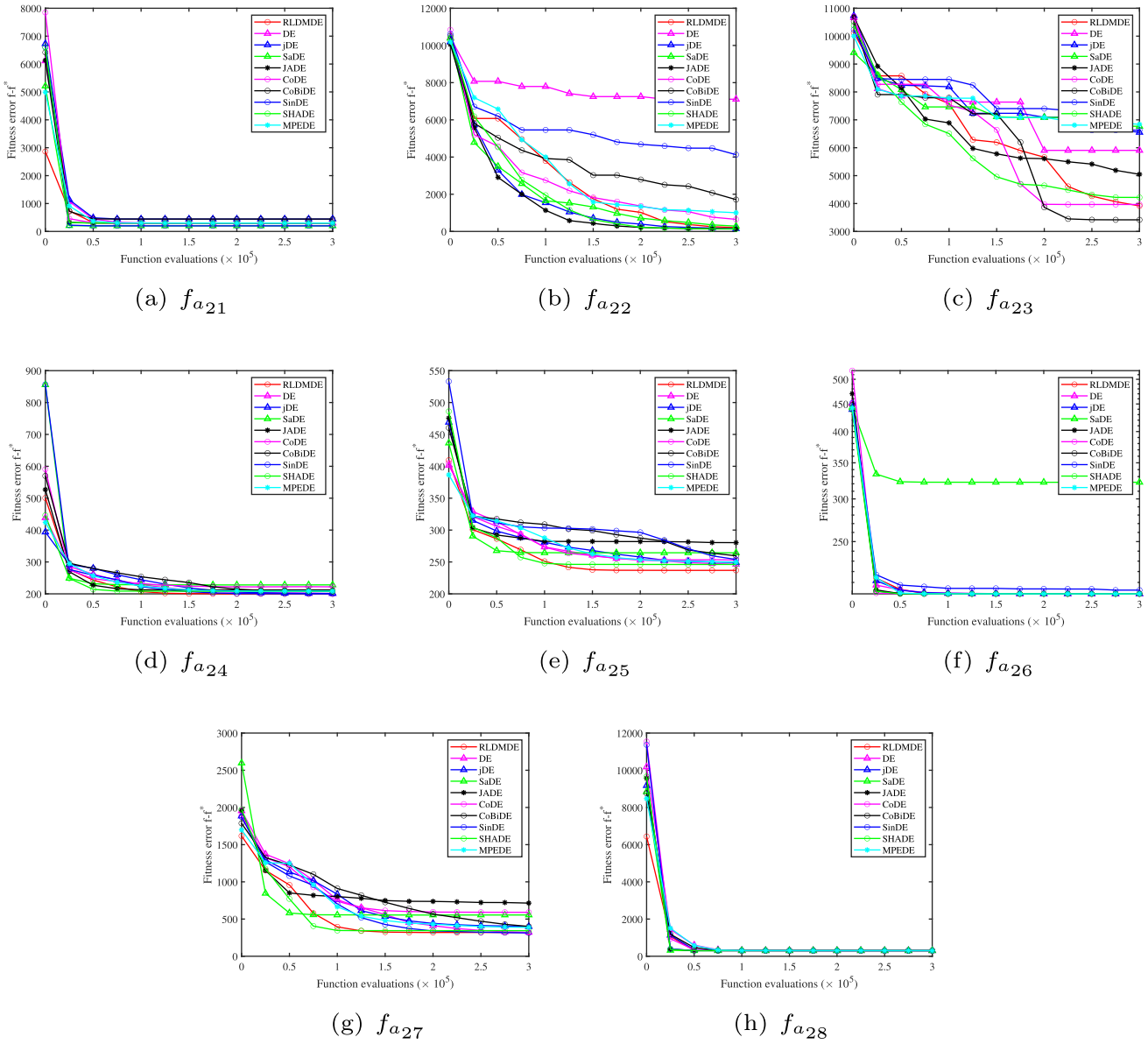


Fig. 8 The convergence curves of the RLDMDE and other algorithms solving composition functions $f_{a_{21}} - - f_{a_{28}}$

Tension/compression spring design

The objective is to minimize its weight while satisfying the minimum deflection, surge frequency, shear stress, and outer diameter constraints. The schematic diagram is shown in Fig. 9. This problem mainly requires optimization of three design variables: wire diameter d, mean spring coil diameter D, and effective number of coils N. The mathematical model is described as follows:

Objective function:

$$min\ f = (N+2)Dd^2. \tag{28}$$

Constraints:

$$\begin{aligned} g_1 &= 1 - \frac{D^3 N}{71785d^4} \le 0 \\ g_2 &= 1 - \frac{140.45d}{D^2 N} \le 0 \\ g_3 &= \frac{4D^2 - dD}{12566(Dd^3 - d^4)} + \frac{1}{5108d^2} - 1 \le 0 \\ g_4 &= \frac{D+d}{1.5} - 1 \le 0, \end{aligned} \tag{29}$$

Table 12 The experimental results of the RLDMDE algorithm with different ps on CEC2013 (30D)

Function	ps=120	30	60	90	150	180	210	240
F1	0	3.33E-13	5.31E-14	0	0	0	0	0
F2	7673.869	12073.65	9356.324	8296.627	5058.742	2825.811	2397.759	2017.124
F3	27.67604	5723094	811861.4	1499.573	65.07769	0.673204	5.362357	50898.89
F4	1.01E-07	3090.441	0.318136	3.61E-06	8.76E-09	3.03E-09	4.48E-09	3.54E-09
F5	9.85E-14	2.35E-13	1.25E-13	1.06E-13	9.85E-14	9.47E-14	9.85E-14	1.14E-13
F6	4.046662	8.551667	2.671113	4.394624	6.472284	7.994651	10.94405	12.42329
F7	0.199348	31.75934	5.234707	0.591891	0.200675	0.12152	0.057213	0.21829
F8	20.9367	20.88336	20.83934	20.94048	20.95254	20.94651	20.94512	20.96033
F9	25.90942	28.5627	27.77203	27.14886	26.50251	26.15462	26.70547	28.72651
F10	0.055529	0.126291	0.069722	0.073089	0.056836	0.059048	0.064071	0.049528
F11	0	4.808959	0.132661	0.066331	3.22E-14	7.92E-06	2.288551	11.76009
F12	24.32468	38.99115	25.47895	20.05832	31.66755	45.52781	56.71436	73.10011
F13	45.43827	97.40279	58.50155	46.37127	50.96264	67.71473	77.75837	88.99261
F14	0.001674	4.227679	0.018737	0.004858	12.07116	90.58861	277.0421	694.3202
F15	3087.171	3355.441	2966.422	2920.624	3416.82	3876.041	4462.849	4929.799
F16	1.186742	0.877902	1.083188	1.107674	1.416627	1.647731	1.918272	2.163677
F17	30.43395	30.70048	30.43375	30.43375	31.22558	34.69487	41.18407	50.55778
F18	81.28568	71.14958	64.89464	73.65934	92.69768	111.4935	128.2589	144.9128
F19	1.33465	1.678452	1.020622	1.057627	1.674629	2.128573	2.775829	3.67849
F20	10.51679	11.45419	10.78467	10.49692	10.45115	10.64558	10.89223	11.27608
F21	280	324.9451	323.4936	251.4515	246.6667	270	276.6667	293.3333
F22	105.5127	108.9223	106.3176	103.8254	123.5042	195.6575	351.1318	716.4172
F23	3185.148	3717.118	3359.601	3310.17	3552.651	3937.966	4705.591	4986.637
F24	202.9385	242.9736	226.3996	209.9694	200.4874	200.2758	200.223	200.0149
F25	263.5795	276.7335	271.0711	268.8502	258.827	258.1605	246.1478	252.1338
F26	200.0006	256.0774	217.9118	203.6992	200.0003	200.0002	200.0001	200.0002
F27	315.3996	811.3905	573.4838	342.3611	326.7682	301.7384	302.4141	300.3733
F28	260	300	266.6667	280	273.3333	286.6667	286.6667	300
Unimodal	–	5	5	4(1)	1(2)	0(1)	0(2)	2(1)
Basic Multimodal	–	12	8	8	14	14	14	14
Composition	–	8	8	6	4	3	3	4
Sum	–	25	21	18(1)	19(2)	17(1)	17(2)	20(1)

where the value ranges of variables D, d, and N are [0.25, 1.30], [0.05, 2.00], [2, 15], respectively. $g_1 - -g_4$ correspond to minimum deflection, surge frequency, shear stress, and outer diameter constraints, respectively. The experimental results comparing the RLDMDE algorithm with the existing literature are presented in Table 14. As shown in the table, the RLDMDE algorithm has a good solution performance for this problem. It is weaker than the most algorithms on the optimal value, but is superior to most algorithms on other items.

Three-bar truss design

The objective is to minimize the volume while ensuring that the stress constraints on each side of the truss members are satisfied. Figure 10 shows the schematic diagram. The problem requires optimizing the cross-sectional area design variables A_1 and A_2. The mathematical model for this problem is described as follows:

Objective function:

$$min\ f = \left(2\sqrt{2}A_1 + A_2\right) \times H. \tag{30}$$

Table 13 The experimental results of the RLDMDE algorithm with different *state_max* on CEC2013 (30D)

Function	*state_max*=3	2	4	5	6	7	8	9
F1	0	0	0	0	0	0	0	0
F2	7673.869	7608.352	5650.532	6840.754	6466.573	6419.709	6669.922	6468.454
F3	27.67604	32.90749	233.0078	1.99091	100735.2	13.01548	48.32535	22.18732
F4	1.01E-07	2.81E-07	1.72E-07	2.15E-07	4.31E-08	2.13E-07	1.06E-07	4.39E-07
F5	9.85E-14	9.47E-14	9.85E-14	8.72E-14	8.72E-14	9.85E-14	9.09E-14	1.06E-13
F6	4.046662	2.751116	4.54646	5.407278	6.62631	5.41411	4.589515	6.98344
F7	0.199348	0.48216	0.41616	0.236289	0.404141	0.322943	0.240827	0.364262
F8	20.9367	20.95121	20.93977	20.94718	20.95345	20.94344	20.9494	20.93068
F9	25.90942	26.25345	26.82827	26.49179	26.00289	27.2944	26.28161	26.99744
F10	0.055529	0.061115	0.049361	0.079919	0.080564	0.074013	0.069243	0.071217
F11	0	0.033165	0	0	0	0	3.79E-15	0.033165
F12	24.32468	25.11427	23.91841	24.07644	23.48663	24.48713	24.29811	24.18102
F13	45.43827	42.22118	43.6898	45.46596	43.70213	42.79168	43.51075	42.35953
F14	0.001674	0.005107	0.002175	0.002921	0.008946	0.001853	0.002118	0.004723
F15	3087.171	3097.797	3176.909	3169.116	3162.837	3142.009	3144.061	3174.844
F16	1.186742	1.248036	1.254097	1.185951	1.07697	1.318792	1.174702	1.186384
F17	30.43395	30.43401	30.43446	30.43452	30.43445	30.43421	30.43478	30.43482
F18	81.28568	82.00003	78.96629	80.47219	81.7167	81.23121	83.14946	81.486
F19	1.33465	1.376167	1.333893	1.334599	1.367389	1.355104	1.345065	1.369298
F20	10.51679	10.53247	10.2965	10.51088	10.41445	10.51337	10.57799	10.25783
F21	280	278.1181	282.9029	258.1181	251.4515	276.6667	270	266.6667
F22	105.5127	106.3901	106.6766	105.5135	105.6247	106.2268	105.9236	105.7394
F23	3185.148	3306.932	3253.478	3320.075	3276.98	3288.728	3285.381	3395.355
F24	202.9385	206.7764	202.2934	201.3341	203.5805	205.362	204.5691	201.2754
F25	263.5795	261.2713	262.5786	265.0111	260.7244	254.6559	260.1467	260.5757
F26	200.0006	203.3693	203.335	200.0006	203.7709	200.0006	206.8265	210.5714
F27	315.3996	371.2179	331.1932	353.1264	345.2457	348.6477	314.5393	331.1402
F28	260	273.3333	293.3333	286.6667	253.3333	286.6667	273.3333	260
Unimodal	–	2 (1)	2 (2)	1 (1)	1 (1)	1 (2)	2 (1)	2 (1)
Basic Multimodal	–	13	8 (1)	9 (1)	10 (1)	11 (1)	12	10
Composition	–	6	6	6	5	6	5	4
Sum	–	21 (1)	16 (3)	16 (2)	16 (2)	18 (3)	19 (1)	16 (2)

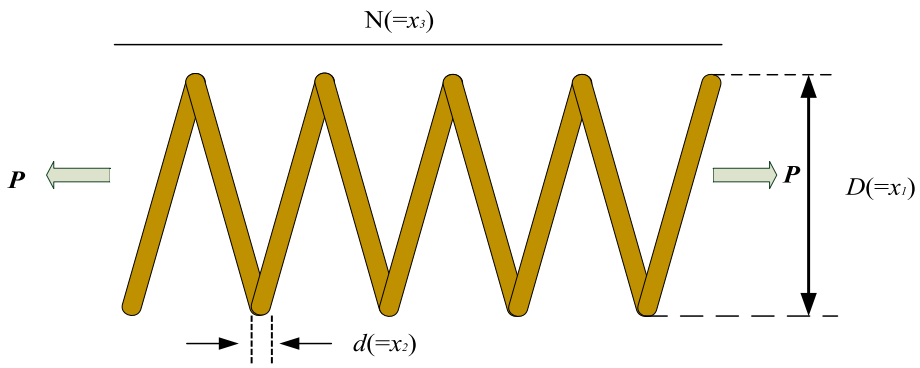


Fig. 9 The schematic diagram of tension/compression spring design

Table 14 Comparison results of the algorithms in solving tension/compression spring design

Algorithm	Statistical results				Optimal solution		
	Worst	Mean	Best	Std	*D*	*d*	*N*
RLDMDE	**0.012679791**	**0.012666993**	0.012665384	2.64E-06	0.051773	0.358728	11.17386
SNS [72]	0.012765873	0.012684717	0.012665246	2.39E-05	0.051587	0.354268	11.43406
Society and civilization [73]	0.016717272	0.012922669	0.012922669	5.92E-04	0.05216	0.368159	10.64844
NDE [74]	0.012687092	0.012668899	0.012665232	5.38E-06	0.051689	0.356718	11.28897
MVDE [65]	0.012719055	0.012667324	0.012665272	**2.45E-06**	0.051681	0.35653	11.3
WCA [75]	0.012952	0.012746	0.012665	8.06E-05	0.05168	0.3565	11.3004
IAPSO [76]	0.01782864	0.013676527	**0.01266523**	1.57E-03	0.051685	0.356629	11.29418
CGO [77]	0.012719055	0.012670085	0.012665246	1.09E-05	0.051662	0.356078	11.32657

The bold data is the optimal value in the same set of data

Constraints:

$$\begin{aligned} g_1 &= \frac{\sqrt{2}A_1 + A_2}{\sqrt{2}A_1^2 + 2A_1A_2} P - \sigma \le 0 \\ g_2 &= \frac{A_2}{\sqrt{2}A_1^2 + 2A_1A_2} P - \sigma \le 0 \\ g_3 &= \frac{1}{A_1 + \sqrt{2}A_2} P - \sigma \le 0, \end{aligned} \tag{31}$$

where the variables A_1 and A_2 are both in the range of $[0, 1.0]$. $g_1 - g_3$ correspond to the stress constraints on each side of the truss member, respectively. The parameter H is $100cm$, and the parameters P and σ are both $2KN/cm^2$. The comparison results with the other literature are shown in Table 15. In solving this problem, the RLDMDE algorithm also has good performance and is more stable.

Gear train design

The problem objective is to minimize the gear ratio. The ratio is described as the ratio of the output angular velocity to the input angular velocity. The schematic diagram is shown in Fig. 11. This problem requires the optimization of the number of teeth n_A, n_B, n_C, and n_D of the four gears. The following presents the establishment of the mathematical model:

Objective function:

$$min\ f = \left(\frac{1}{6.931} - \frac{n_C n_B}{n_A n_D}\right)^2, \tag{32}$$

where the four variables are in the range of $[12, 60]$. Table 16 records the experimental results of the RLDMDE algorithm and the existing literature. The RLDMDE algorithm realizes the solution of the known optimal value, and the stability is superior to other algorithms.

Cantilever beam design

The cantilever beam consists of five hollow square blocks with fixed thickness and different sizes in this problem. The schematic diagram is shown in Fig. 12. Under the condition of satisfying the relevant constraints, the overall weight of the cantilever beam is minimized by optimizing the width (or height) x_i of each hollow block. The following is the mathematical model for this problem:

Objective function:

$$min\ f = 0.0624(x_1 + x_2 + x_3 + x_4 + x_5). \tag{33}$$

Constraints:

$$g = \frac{61}{x_1^3} + \frac{37}{x_2^3} + \frac{19}{x_3^3} + \frac{7}{x_4^3} + \frac{1}{x_5^3} - 1 \le 0, \tag{34}$$

where the range of each variable is $[0.01, 100]$. g is the length constraint of each hollow square block. The experimental results are shown in Table 17. The results demonstrate that the RLDMDE algorithm is superior to the results in the existing literature in terms of the mean value and optimal value. The stability (standard deviation) is also superior to most algorithms.

I-beam design

The problem is to minimize the vertical deflection of the I-beam under meeting the constraints. The schematic diagram is shown in Fig. 13. The problem mainly optimizes four design variables: flange width b, section height h, flange thickness t_f, and web thickness t_w. The mathematical model is established as follows:

$$min\ f = \frac{5000}{\frac{t_w(h-2t_f)^3}{12} + \frac{bt_f^3}{6} + 2bt_f\left(\frac{h-t_f}{2}\right)^2}. \tag{35}$$

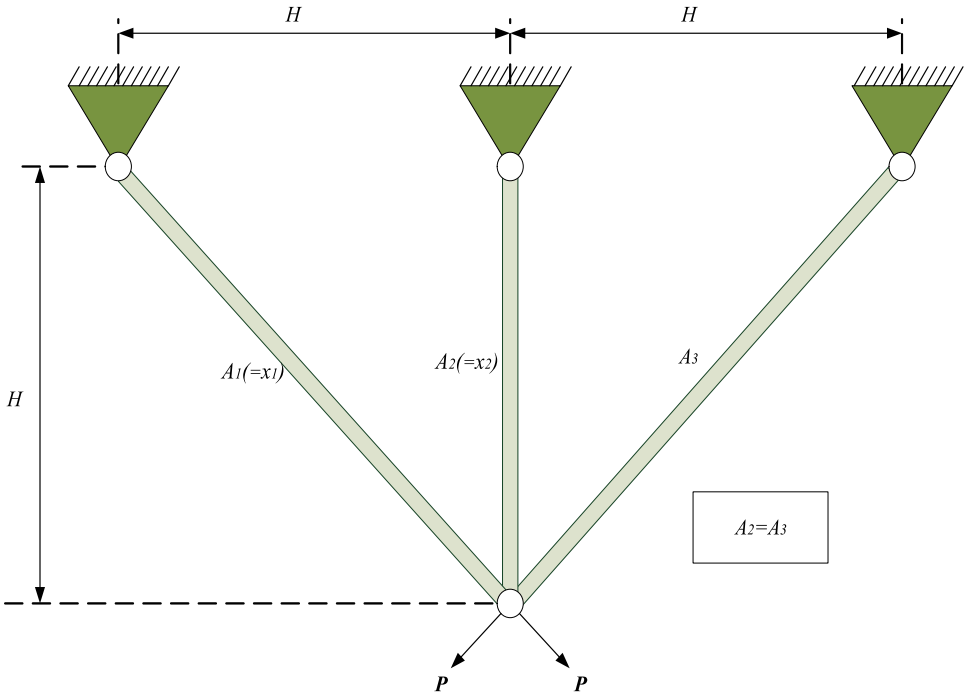


Fig. 10 The schematic diagram of three-bar truss design

Table 15 Comparison results of the algorithms in solving three-bar truss design

Algorithm	Statistical results				Optimal solution	
	Worst	Mean	Best	Std	A_1	A_2
RLDMDE	**263.8958434**	**263.8958434**	263.8958434	2.49E-10	0.788675	0.408248
SNS [72]	263.8958561	263.8958462	263.8958434	3.31E-06	0.788685	0.408221
NDE [74]	**263.8958434**	**263.8958434**	263.8958434	**0.00E+00**	0.788675	0.408248
MVDE [65]	263.8958548	**263.8958434**	263.8958434	2.58E-07	0.788675	0.408248
WCA [75]	263.896201	263.895903	263.8958434	8.71E-05	0.788651	0.408316
WSA [78]	263.8974322	263.8960669	263.8958434	3.12E-04	0.788683	0.408227
PSO-DE [79]	**263.8958434**	**263.8958434**	263.8958434	4.50E-10	0.788675	0.408248
AOS [80]	263.8958453	263.8958435	**263.8958433**	8.26E-09	0.788675	0.40824
CGO [77]	263.8960068	263.8958511	**263.8958433**	2.51E-05	0.788674	0.408252

The bold data is the optimal value in the same set of data

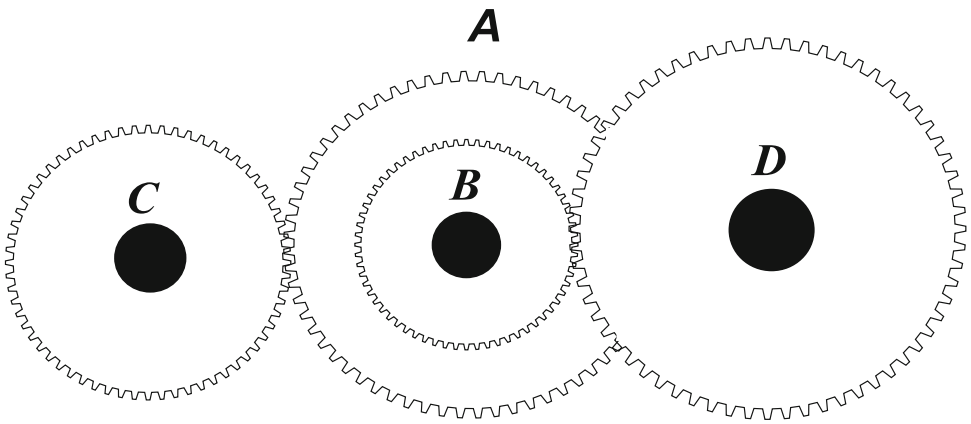


Fig. 11 The schematic diagram of gear train design

Table 16 Comparison results of the algorithms in solving gear train design

Algorithm	Statistical results				Optimal solution			
	Worst	Mean	Best	Std	n_A	n_B	n_C	n_D
RLDMDE	**2.31E-11**	**6.78E-12**	**2.70E-12**	**8.29E-12**	43	16	19	49
SNS [72]	1.36E-09	1.68E-10	**2.70E-12**	3.75E-10	43	19	16	49
WSA [78]	1.36E-09	1.68E-10	**2.70E-12**	3.83E-10	43	16	19	49
MBA [81]	2.06E-08	2.47E-09	**2.70E-12**	3.94E-09	43	16	19	49
CS [82]	2.36E-09	1.98E-09	**2.70E-12**	3.55E-09	43	16	19	49
IAPSO [76]	1.83E-08	5.49E-09	**2.70E-12**	6.36E-09	43	16	19	49

The bold data is the optimal value in the same set of data

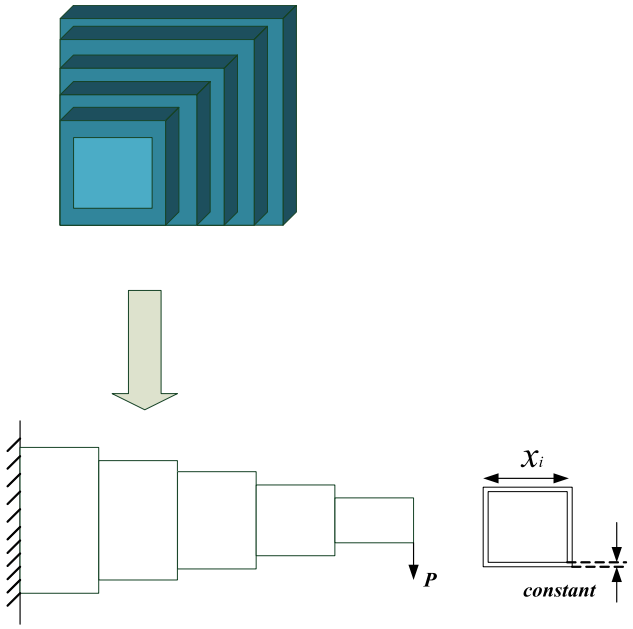


Fig. 12 The schematic diagram of cantilever beam design

Constraints:

$$\begin{aligned} g_1 =& 2bt_w + t_w(h - 2t_f) \le 300 \\ g_2 =& \frac{18h \times 10^4}{t_w(h-2t_f)^3 + 2bt_w(4t_f^2 + 3h(h-2t_f))} \\ &+ \frac{15b \times 10^3}{(h-2t_f)t_w^3 + 2t_w b^3} \le 56, \end{aligned} \tag{36}$$

where the range of variable b is $[10, 50]$, variable h is $[10, 80]$, and variables t_w and t_f are both $[0.9, 5.0]$. g_1 and g_2 are the cross-sectional area and stress constraint of the I-beam, respectively. Table 18 shows the experimental results. The RLDMDE algorithm achieves the known optimal value and outperforms other algorithms in performance.

Tubular column design

The goal of the problem is to obtain a more uniform column by minimizing the cost under the condition of meeting the relevant constraints. The schematic diagram is shown in Fig. 14. This problem mainly optimizes two design variables, the mean diameter d of column and the thickness t of the tube. The mathematical model is established as follows:

$$min\ f = 9.8dt + 2d. \tag{37}$$

Constraints:

$$\begin{aligned} g_1 &= \frac{P}{\pi dt\sigma_y} - 1 \le 0 \\ g_2 &= \frac{8PL^2}{\pi^3 Edt(d^2 + t^2)} - 1 \le 0 \\ g_3 &= \frac{2.0}{d} - 1 \le 0 \\ g_4 &= \frac{d}{14} - 1 \le 0 \\ g_5 &= \frac{0.2}{t} - 1 \le 0 \\ g_6 &= \frac{t}{0.8} - 1 \le 0, \end{aligned} \tag{38}$$

where the range of variable d is $[2, 14]$, and t is $[0.2, 0.8]$. g_1 and g_2 are buckling and yield stresses constraints. $g_3 - -g_6$ are the range constraints of variables. P represents the compressive load that the tubular column can withstand, and the value is $2500kgf$. L is the length of the cylinder and its value is $250cm$. The parametric yield stress σ_y in the column material is $500kgf/cm^2$ and the elastic modulus E is $0.85\times10^6 kgf/cm^2$. As recorded in Table 19, the RLDMDE algorithm achieves the same known optimal value as the the SNS algorithm, but the RLDMDE algorithm is significantly more stable than other algorithms.

Table 17 Comparison results of the algorithms in solving cantilever beam design

Algorithm	Statistical results				Optimal solution				
	Worst	Mean	Best	Std	x_1	x_2	x_3	x_4	x_5
RLDMDE	1.339956565	**1.339956397**	**1.339956361**	4.27E-08	6.015501	5.309147	4.495198	3.500744	2.153071
SNS [72]	1.3399576	1.3399576	1.3399576	**1.11E-15**	6.01545	5.31066	4.488	3.50528	2.15428
SOS [83]	N/A	1.33997	1.33996	1.10E-05	6.01878	5.30344	4.49587	3.49896	2.15564
AOS [80]	**1.339956366**	1.351954573	1.339956366	0.02499743	6.016165	5.308903	4.494578	3.501151	2.152508
CGO [77]	1.340602747	1.340052419	1.339970289	0.000122452	6.024656	5.303313	4.484319	3.511757	2.149838

The bold data is the optimal value in the same set of data

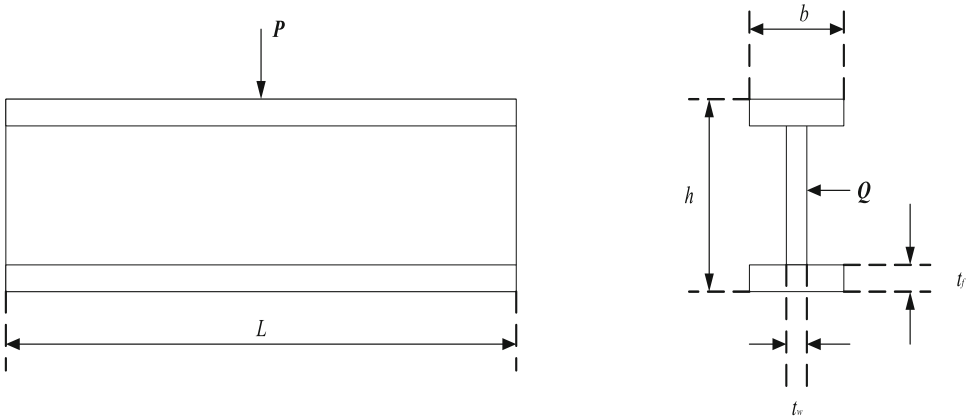


Fig. 13 The schematic diagram of I-beam design

Table 18 Comparison results of the algorithms in solving I-beam design

Algorithm	Statistical results				Optimal solution			
	Worst	Mean	Best	Std	h	b	t_w	t_f
RLDMDE	0.01307412	**0.013074117**	**0.0130741**	5.80E-09	80	50	0.9	2.321792
SNS [72]	0.0130764	0.0130743	**0.0130741**	4.31E-07	80	50	0.9	2.3217
SOS [83]	N/A	0.0130884	**0.0130741**	4.00E-05	80	50	0.9	2.3217
CS [82]	0.01353646	0.0132165	0.0130747	0.0001345	80	50	0.9	2.321672
CGO [77]	**0.013074119**	0.013074119	**0.0130741**	**1.28E-11**	80	50	0.9	2.321792

The bold data is the optimal value in the same set of data

Table 19 Comparison results of the algorithms in solving tubular column design

Algorithm	Statistical results				Optimal solution	
	Worst	Mean	Best	Std	d	t
RLDMDE	**26.48636147**	**26.4863615**	**26.48636147**	**7.23E-15**	5.452181	0.291626
SNS [72]	26.48637095	26.48636249	**26.48636147**	2.22E-06	5.451156	0.291965
AOS [80]	26.60821361	26.53161399	26.53137828	0.00193	5.451153	0.291967
CS [82]	26.53972	26.53504	26.53217	0.00193	5.45139	0.29196
HSO-GA [84]	29.22785442	26.69081931	26.53132788	5.12E-01	5.451156	0.291965
EO [84]	26.53178332	26.53135981	26.53132788	7.25E-05	5.451156	0.291965

The bold data is the optimal value in the same set of data

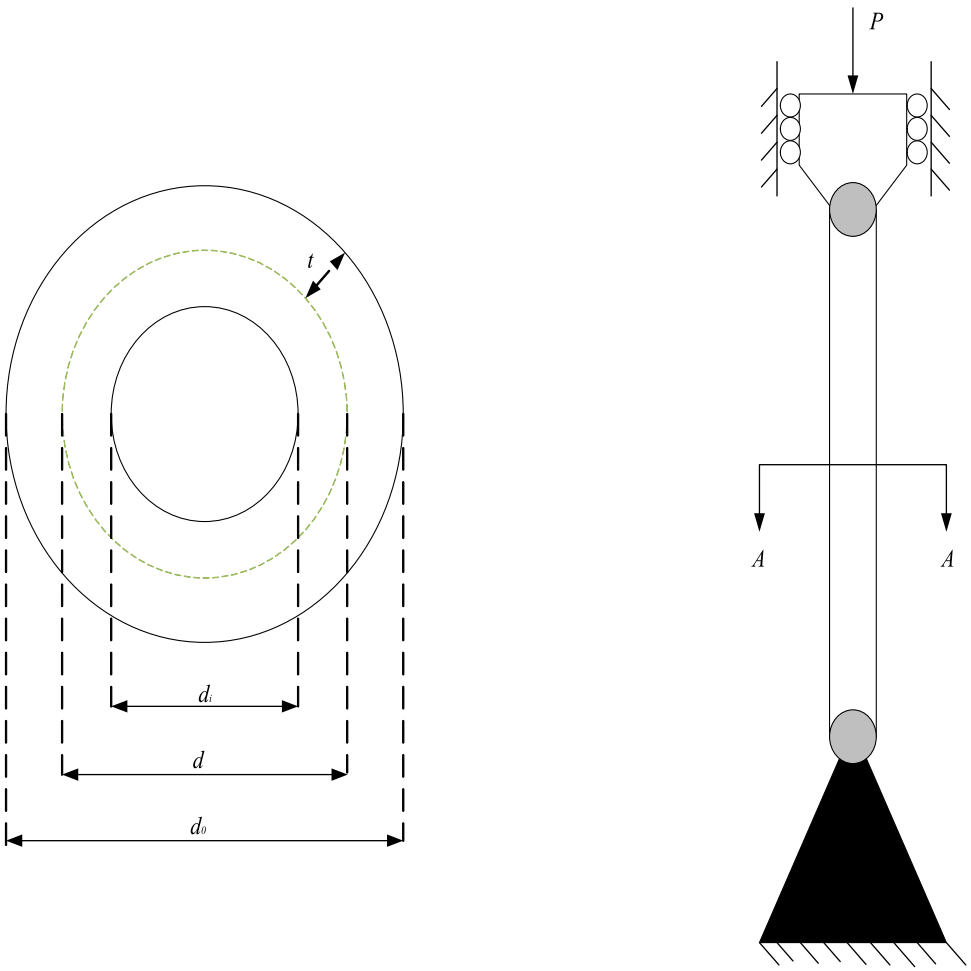


Fig. 14 The schematic diagram of tubular column design

Conclusions

A dynamic multi-strategy integrated differential evolution algorithm based on reinforcement learning (RLDMDE) is proposed in this paper. The RLDMDE algorithm adopts a multi-population structure. Each subpopulation has the same data structure, but remains independent during execution. To improve the quality of the initial solutions, this paper adopts the initialization method of combining the good point set and the random opposition-based learning. By introducing the Q-learning algorithm, the mutation strategy of each subpopulation is no longer fixed, but adaptively selected according to the current environmental state. At the same time, this paper proposes an individual dynamic migration strategy based on the population state to avoid the waste of individual computing resources. To evaluate the optimization performance of the RLDMDE algorithm, two benchmark function sets, CEC2013 and CEC2017, are used for testing. The results demonstrate that the RLDMDE algorithm outperforms the original DE algorithm and eight state-of-art DE variants. In addition, this paper also selects six real-world engineering design problems to further verify the RLDMDE algorithm in solving practical problems. Through the test, the RLDMDE algorithm shows excellent solution performance in solving these problems, and the obtained results are close to or even superior to the optimal results in the existing literature.

The RLDMDE algorithm also has some drawbacks. Since the algorithm contains many strategies, the complexity is relatively high. When dealing with large-scale high-dimensional complex optimization problems, the RLDMDE algorithm may face the problem of long solution time. It may also face deployment difficulties when dealing with resource-constrained optimization problems. In addition, the RLDMDE algorithm itself has randomness, which may affect the agent to make correct decisions in reinforcement learning, thereby reducing the solution performance. In future work, we will focus on the research and improvement of the RLDMDE algorithm. The efficiency and robustness of the RLDMDE algorithm can be further improved by introducing other techniques (such as surrogate-assisted [7, 11], data reduction, and compact mechanism [66]). We also plan to apply it tomore fields to solve more practical problems.

Data Availability The data used to support the findings of this study are included within the article.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

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

References

1. Abualigah L, Elaziz MA, Khasawneh AM et al (2022) Metaheuristic optimization algorithms for solving real-world mechanical engineering design problems: a comprehensive survey, applications, comparative analysis, and results. Neural Comput Appl 34:1–30
2. Xiong H, Shi S, Ren D et al (2022) A survey of job shop scheduling problem: the types and models. Comput Operat Res 142:105731
3. Djordjevic V, Tao H, Song X et al (2023) Data-driven control of hydraulic servo actuator: an event-triggered adaptive dynamic programming approach. Math Biosci Eng 20(5):8561–8582
4. Alipour MM, Razavi SN, Feizi Derakhshi MR et al (2018) A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. Neural Comput Appl 30:2935–2951
5. Zhou C, Tao H, Chen Y et al (2022) Robust point-to-point iterative learning control for constrained systems: a minimum energy approach. Int J Robust Nonlinear Control 32(18):10139–10161
6. Fu Z, Chu SC, Watada J et al (2022) Software and hardware co-design and implementation of intelligent optimization algorithms. Appl Soft Comput 129:109639
7. Hu P, Pan JS, Chu SC et al (2022) Multi-surrogate assisted binary particle swarm optimization algorithm and its application for feature selection. Appl Soft Comput 121:108736
8. Pant M, Zaheer H, Garcia-Hernandez L et al (2020) Differential evolution: a review of more than two decades of research. Eng Appl Artif Intell 90:103479
9. Mitchell M (1998) An introduction to genetic algorithms. MIT Press
10. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341
11. Pan JS, Liu N, Chu SC et al (2021) An efficient surrogate-assisted hybrid optimization algorithm for expensive optimization problems. Inf Sci 561:304–325
12. Yang XS, Karamanoglu M (2020) Nature-inspired computation and swarm intelligence: a state-of-the-art overview. Academic Press, pp 3–18
13. Song PC, Chu SC, Pan JS et al (2022) Simplified phasmatodea population evolution algorithm for optimization. Complex Intell Syst 8(4):2749–2767
14. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39
15. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, IEEE, pp 1942–1948
16. Chu SC, Du ZG, Peng YJ et al (2021) Fuzzy hierarchical surrogate assists probabilistic particle swarm optimization for expensive high dimensional problem. Knowl-Based Syst 220:106939
17. Karaboga D, Basturk B (2008) On the performance of artificial bee colony (abc) algorithm. Appl Soft Comput 8(1):687–697
18. Yang XS, Hossein Gandomi A (2012) Bat algorithm: a novel approach for global engineering optimization. Eng Comput 29(5):464–483
19. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
20. Pan JS, Zhang LG, Wang RB et al (2022) Gannet optimization algorithm: a new metaheuristic algorithm for solving engineering optimization problems. Math Comput Simul 202:343–373
21. Das S, Suganthan PN (2010) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
22. Li J, Gao Y, Zhang H et al (2022) Self-adaptive opposition-based differential evolution with subpopulation strategy for numerical and engineering optimization problems. Complex Intell Syst 8(3):2051–2089
23. Brest J, Greiner S, Boskovic B et al (2006) Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans Evol Comput 10(6):646–657
24. Zhang J, Sanderson AC (2009) Jade: adaptive differential evolution with optional external archive. IEEE Trans Evol Comput 13(5):945–958
25. Qin AK, Huang VL, Suganthan PN (2008) Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput 13(2):398–417
26. Wang Y, Cai Z, Zhang Q (2011) Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans Evol Comput 15(1):55–66
27. Wu G, Mallipeddi R, Suganthan PN et al (2016) Differential evolution with multi-population based ensemble of mutation strategies. Inf Sci 329:329–345
28. Meng Z, Yang C (2021) Hip-de: historical population based mutation strategy in differential evolution with parameter adaptive mechanism. Inf Sci 562:44–77
29. Li X, Wang L, Jiang Q et al (2021) Differential evolution algorithm with multi-population cooperation and multi-strategy integration. Neurocomputing 421:285–302
30. Tong L, Dong M, Jing C (2018) An improved multi-population ensemble differential evolution. Neurocomputing 290:130–147
31. Ma Y, Bai Y (2020) A multi-population differential evolution with best-random mutation strategy for large-scale global optimization. Appl Intell 50(5):1510–1526
32. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
33. Naeem M, Rizvi STH, Coronato A (2020) A gentle introduction to reinforcement learning and its application in different fields. IEEE Access 8:209320–209344
34. Zhuang Z, Tao H, Chen Y et al (2022) An optimal iterative learning control approach for linear systems with nonuniform trial lengths under input constraints. IEEE Trans Syst Man Cybern Syst 53(6):3461–3473
35. Wang YC, Usher JM (2005) Application of reinforcement learning for agent-based production scheduling. Eng Appl Artif Intell 18(1):73–82

36. Chen Q, Jin Y, Song Y (2022) Fault-tolerant adaptive tracking control of Euler-Lagrange systems-an echo state network approach driven by reinforcement learning. Neurocomputing 484:109–116
37. Huang L, Bi S, Zhang YJA (2019) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans Mob Comput 19(11):2581–2593
38. Wang FY, Zhang JJ, Zheng X et al (2016) Where does AlphaGo go: from church-Turing thesis to AlphaGo thesis and beyond. IEEE/CAA J Autom Sin 3(2):113–120
39. Yu L, Xie W, Xie D et al (2019) Deep reinforcement learning for smart home energy management. IEEE Internet Things J 7(4):2751–2762
40. Tutsoy O (2021) Pharmacological, non-pharmacological policies and mutation: an artificial intelligence based multi-dimensional policy making algorithm for controlling the casualties of the pandemic diseases. IEEE Trans Pattern Anal Mach Intell 44(12):9477–9488
41. Xu Z, Han G, Liu L et al (2021) Multi-energy scheduling of an industrial integrated energy system by reinforcement learning-based differential evolution. IEEE Trans Green Commun Netw 5(3):1077–1090
42. Zhao M, Li G, Li H et al (2022) Reliable scheduling algorithm for space debris monitoring resources using adaptive multipopulation differential evolutionary optimization with reinforcement learning. IEEE Trans Reliab 71(2):687–697
43. Xu Y, Pi D (2020) A reinforcement learning-based communication topology in particle swarm optimization. Neural Comput Appl 32:10007–10032
44. Seyyedabbasi A, Aliyev R, Kiani F et al (2021) Hybrid algorithms based on combining reinforcement learning and metaheuristic methods to solve global optimization problems. Knowl-Based Syst 223:107044
45. Wu D, Wang S, Liu Q et al (2022) An improved teaching-learning-based optimization algorithm with reinforcement learning strategy for solving optimization problems. Comput Intell Neurosci 2022. https://doi.org/10.1155/2022/1535957
46. Hamad QS, Samma H, Suandi SA et al (2022) Q-learning embedded sine cosine algorithm (qlesca). Expert Syst Appl 193:116417
47. Samma H, Mohamad-Saleh J, Suandi SA et al (2020) Q-learning-based simulated annealing algorithm for constrained engineering design problems. Neural Comput Appl 32:5147–5161
48. Huynh TN, Do DT, Lee J (2021) Q-learning-based parameter control in differential evolution for structural optimization. Appl Soft Comput 107:107464
49. Tan Z, Li K (2021) Differential evolution with mixed mutation strategy based on deep reinforcement learning. Appl Soft Comput 111:107678
50. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8:279–292
51. Luogeng H, Yuan W (1978) Application of number theory in modern analysis. Springer
52. Long W, Jiao J, Liang X et al (2019) A random opposition-based learning grey wolf optimizer. IEEE Access 7:113810–113825
53. Liang JJ, Qu B, Suganthan PN et al (2013) Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212(34):281–295
54. Wu G, Mallipeddi R, Suganthan PN (2017) Problem definitions and evaluation criteria for the cec 2017 competition on constrained real-parameter optimization. National University of Defense Technology, Changsha, Hunan, PR China and Kyungpook National University, Daegu, South Korea and Nanyang Technological University, Singapore, Technical Report
55. Pan JS, Liu N, Chu SC (2022) A competitive mechanism based multi-objective differential evolution algorithm and its application in feature selection. Knowl-Based Syst 245:108582
56. Fan HY, Lampinen J (2003) A trigonometric mutation operation to differential evolution. J Global Optim 27:105–129
57. Piotrowski AP (2017) Review of differential evolution population size. Swarm Evol Comput 32:1–24
58. Omran MG, Salman A, Engelbrecht AP (2005) Self-adaptive differential evolution. In: Computational Intelligence and Security: International Conference, CIS 2005, Xi'an, China, December 15-19, 2005, Proceedings Part I, Springer, pp 192–199
59. Tanabe R, Fukunaga A (2013) Success-history based parameter adaptation for differential evolution. In: 2013 IEEE congress on evolutionary computation, IEEE, pp 71–78
60. Wang Y, Li HX, Huang T et al (2014) Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. Appl Soft Comput 18:232–247
61. Sun J, Zhang Q, Tsang EP (2005) De/eda: a new evolutionary algorithm for global optimization. Inf Sci 169(3–4):249–262
62. Rahnamayan S, Tizhoosh HR, Salama MM (2008) Opposition-based differential evolution. IEEE Trans Evol Comput 12(1):64–79
63. Das S, Abraham A, Chakraborty UK et al (2009) Differential evolution using a neighborhood-based mutation operator. IEEE Trans Evol Comput 13(3):526–553
64. Sun G, Cai Y (2017) A novel neighborhood-dependent mutation operator for differential evolution. In: 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), IEEE, pp 837–841
65. De Melo VV, Carosio GL (2013) Investigating multi-view differential evolution for solving constrained engineering design problems. Expert Syst Appl 40(9):3370–3377
66. Khalfi S, Draa A, Iacca G (2021) A compact compound sinusoidal differential evolution algorithm for solving optimisation problems in memory-constrained environments. Expert Syst Appl 186:115705
67. Li C, Deng L, Qiao L et al (2022) An efficient differential evolution algorithm based on orthogonal learning and elites local search mechanisms for numerical optimization. Knowl-Based Syst 235:107636
68. Mallipeddi R, Suganthan PN, Pan QK et al (2011) Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput 11(2):1679–1696
69. Olorunda O, Engelbrecht AP (2008) Measuring exploration/exploitation in particle swarms using swarm diversity. In: 2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence), IEEE, pp 1128–1134
70. Ursem RK (2002) Diversity-guided evolutionary algorithms. In: International Conference on Parallel Problem Solving from Nature, Springer, pp 462–471
71. Draa A, Bouzoubia S, Boukhalfa I (2015) A sinusoidal differential evolution algorithm for numerical optimisation. Appl Soft Comput 27:99–126
72. Bayzidi H, Talatahari S, Saraee M et al (2021) Social network search for solving engineering optimization problems. Comput Intell Neurosci 2021:1–32
73. Ray T, Liew KM (2003) Society and civilization: an optimization algorithm based on the simulation of social behavior. IEEE Trans Evol Comput 7(4):386–396
74. Mohamed AW (2018) A novel differential evolution algorithm for solving constrained engineering optimization problems. J Intell Manuf 29:659–692
75. Eskandar H, Sadollah A, Bahreininejad A et al (2012) Water cycle algorithm-a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput Struct 110:151-166

76. Guedria NB (2016) Improved accelerated PSO algorithm for mechanical engineering optimization problems. Appl Soft Comput 40:455–467
77. Talatahari S, Azizi M (2020) Optimization of constrained mathematical and engineering design problems using chaos game optimization. Comput Ind Eng 145:106560
78. Kaveh A, Eslamlou AD (2020) Water strider algorithm: a new metaheuristic and applications. Structures 25:520–541
79. Liu H, Cai Z, Wang Y (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10(2):629–640
80. Azizi M, Talatahari S, Giaralis A (2021) Optimization of engineering design problems using atomic orbital search algorithm. IEEE Access 9:102497–102519
81. Sadollah A, Bahreininejad A, Eskandar H et al (2013) Mine blast algorithm: a new population based algorithm for solving constrained engineering optimization problems. Appl Soft Comput 13(5):2592-2612
82. Gandomi AH, Yang XS, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 29:17–35
83. Cheng MY, Prayogo D (2014) Symbiotic organisms search: a new metaheuristic optimization algorithm. Comput Struct 139:98–112
84. Gupta S, Abderazek H, Yıldız BS et al (2021) Comparison of metaheuristic optimization algorithms for solving constrained mechanical design optimization problems. Expert Syst Appl 183:115351

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