SCIENCE CHINA
Mathematics

AI Methods for Optimization Problems https://doi.org/10.1007/s11425-023-2252-9
• ARTICLES •

Learning to select the recombination operator for derivative-free optimization

Haotian Zhang[1], Jianyong Sun[1,*], Thomas Bäck[2] & Zongben Xu[1]

[1]*School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an* 710049*, China;*
[2]*Leiden Institute of Advanced Computer Science, Leiden University, Leiden* 2333 *CA, Netherland*

*Email: zht*570795275*@stu.xjtu.edu.cn, jy.sun@xjtu.edu.cn, t.h.w.baeck@liacs.leidenuniv.nl, zbxu@mail.xjtu.edu.cn*

Received July 1, 2023; accepted January 11, 2024; published online February 22, 2024

Abstract Extensive studies on selecting recombination operators adaptively, namely, adaptive operator selection (AOS), during the search process of an evolutionary algorithm (EA), have shown that AOS is promising for improving EA's performance. A variety of heuristic mechanisms for AOS have been proposed in recent decades, which usually contain two main components: the feature extraction and the policy setting. The feature extraction refers to as extracting relevant features from the information collected during the search process. The policy setting means to set a strategy (or policy) on how to select an operator from a pool of operators based on the extracted feature. Both components are designed by hand in existing studies, which may not be efficient for adapting optimization problems. In this paper, a generalized framework is proposed for learning the components of AOS for one of the main streams of EAs, namely, differential evolution (DE). In the framework, the feature extraction is parameterized as a deep neural network (DNN), while a Dirichlet distribution is considered to be the policy. A reinforcement learning method, named policy gradient, is used to train the DNN. As case studies, the proposed framework is applied to two DEs including the classic DE and a recently-proposed DE, which result in two new algorithms named PG-DE and PG-MPEDE, respectively. Experiments on the Congress of Evolutionary Computation (CEC) 2018 test suite show that the proposed new algorithms perform significantly better than their counterparts. Finally, we prove theoretically that the considered classic methods are the special cases of the proposed framework.

Keywords evolutionary algorithm, differential evolution, adaptive operator selection, reinforcement learning, deep learning

MSC(2020) 68T05, 68W01, 90C40

Citation: Zhang H T, Sun J Y, Bäck T, et al. Learning to select the recombination operator for derivative-free optimization. Sci China Math, 2024, 67, https://doi.org/10.1007/s11425-023-2252-9

1 Introduction

Optimization for derivative-free objective functions is an important research avenue. Such optimization can be seen in a variety of scenarios, such as hyper-parameter tuning of algorithms [17] and automatic machine learning [51]. A variety of optimization algorithms have been proposed for derivative-free

* Corresponding author

optimization. These algorithms can be roughly categorized as surrogate model-assisted algorithms [10,18] or evolutionary algorithms. In this paper, we focus on evolutionary algorithms.

Evolutionary computation has been a promising paradigm for derivative-free optimization due to its advantages such as inherit parallel computing, self-organization and without requiring gradient information and explicit forms of the objective function. In the past several decades, many evolutionary algorithms (EAs) [11] have been proposed, such as differential evolution (DE) [6, 26], evolutionary strategies (ES) [14, 15], particle swarm optimization (PSO) [8], estimation of distribution algorithm (EDA) [33] and others.

Adaptive EAs have become the mainstream research direction and attracted tremendous attentions in recent years. The adaptation in EA can be roughly categorized as either controlling the algorithm's parameters or selecting adaptively the recombination operators during the search process. A variety of adaptive EAs, such as adaptive DE (e.g., JADE [50]), adaptive ES [29], hybrid ES (e.g., HSES [44]), adaptive PSO [43] and others have been proposed. It has been well recognized that adaptive DE can deliver competitive rankings as seen in various CEC competitions[1)] among these methods.

In the two categories of adaptive EA, adaptive parameter control (APC) usually refers to the adaptive generation of EA's parameters (such as the scaling factor F and the crossover rate CR in DE) at each generation. Adaptive operator selection (AOS) means adaptively selecting one recombination operator from a pool of candidate operators at each generation. This paper focuses on AOS for DE.

Existing AOS mechanisms mainly contain two components, which we name the feature extraction and the policy setting. The feature extraction refers to as collecting relevant information during the optimization process and extracting features from the information. The policy setting means to develop a policy to determine the probability of selecting each operator based on the extracted feature. Taking SaDE [28] as an example, we see that SaDE first calculates the so-called "success rate" which is the ratio between the number of each operator used in previous generations and the number of each operator which can improve the quality of solutions in previous generations. The probability of selecting one operator is the ratio between the success rate of the operator and the summation of the success rates of all the operators. In our opinion, the number of each operator used in previous generations and the number of each operator, which can improve the quality of solutions in the previous several generations is the relevant information collected during the optimization process. The calculation of the success rate is the feature extraction process, while the calculation of selection probability is the policy being set.

To the best of our knowledge, existing AOS mechanisms are all designed by hand. In this paper, inspired by the advancement of deep learning, we propose to use the so-called learning-to-optimize technology to *learn* an AOS mechanism. The underlying idea is to secure knowledge (represented by the deep neural network, DNN) from optimizing related problems and use the knowledge to design the *optimal* selection mechanism. The contributions of this paper can be summarized as follows:

• A framework based on learning-to-optimize is proposed to adaptively select recombination operators.

• A Dirichlet-based policy gradient algorithm is proposed to train the network used in the framework.

• As case studies, the framework is applied to two classical DEs, namely, classic DE and MPEDE. Two resultant algorithms, named PG-DE and PG-MPEDE, are established, respectively.

• We prove theoretically that the newly proposed algorithm (i.e., PG-DE and PG-MPEDE) is asymptotically identical to the classical DE (i.e., SaDE and MPEDE).

• Experiments on the CEC 2018 test suite show that the developed algorithms perform significantly better than their counterparts and PG-MPEDE performs the best among many developed EAs.

The rest of this paper is organized as follows. In Section 2, we review the studies on AOS and the application of learning-to-optimize on solving optimization problems. In Section 3, we briefly introduce some preliminaries. Section 4 presents the details of the application of the proposed framework to the two developed DEs. Experiments on the CEC 2018 test suite are presented in Section 5. In Section 6, we conclude the paper.

[1)] https://2024.ieeewcci.org/program/competitions

2 Related work

Studies on adaptive operator selection can be divided into three categories. In the first category, at each generation, an operator is selected from a pool of operators by a fixed policy. For example, in $j\mathrm{DE}_{\mathrm{MM}}$ [42], the selection of a mutation operator is determined by both the population size and a random variable. Silva et al. [4] proposed a refined strategy for operator selection, in which with different proportions of maximum generations, different strategies are applied. The methods in this category consider little information from the optimization process.

In the second category, operators used in algorithms, such as CoDE and EPSDE, are selected by trial and error. For example, CoDE [39] first generates three trial vectors by three mutation operators respectively and the best vector survives. EPSDE [23] first assigns the mutation operators randomly to each individual and compares the trial vectors generated by the mutation operators with their parents. If the trial vector is better than its parents, the mutation operator that generates the trial vector will be inherited; otherwise will be re-initialized. Methods in this category only consider the information from the last two populations, but ignore previous information.

In the third category, operators are selected based on the information collected during the optimization process. For example, SaDE [28] proposes to select an operator based on the success rate, which characterizes how well each operator performs. cDE [36] selects an operator based on how well each operator performs. MPEDE [40] selects from a pool of three DE mutation operators. It divides the population into four sub-populations and the first three sub-populations are assigned to the three operators, respectively. The last sub-population is assigned to the operator with the best performance in the previous several generations. DE-AOPS [9] first sets a candidate pool of operators and then assesses each operator with a few budgets. The success rate is used to select half of the best operators and the operators will survive in the candidate pool. LSAOS-DE [30] also proposes a mechanism to select operators based on the success rate. Different from the methods in this category, it not only considers the success rate of each operator but also considers the landscape of the objective function.

The idea of learning-to-optimize was first proposed by Andrychowicz et al. [1]. Its core idea is to extract knowledge from existing optimization experiences to aid the optimization of new problems. The knowledge extraction is implemented by using an intelligent agent which is represented as a DNN. In [1], a recurrent neural network named long short-term memory (LSTM) [16] is applied to decide the descent direction of a commonly-used stochastic gradient descent (SGD) algorithm at each iteration. It is claimed that the learned LSTM can result in a better performance than SGD and the adaptive moment (ADAM) algorithm [19] on training DNNs. Some studies based on learning-to-optimize are proposed afterward. For example, Li and Malik [20] applied a reinforcement learning algorithm to learn a DNN on deciding the direction of SGD. Wang et al. [38] used LSTM to learn the hyper-parameters of ADAM. Chen et al. [3] applied LSTM to learn the promising sampling point for a derivative-free optimization algorithm.

The learning-to-optimize technology has also been devoted to solving combinatorial optimization problems (COPs). In [2, 5], the solution to a COP is constructed sequentially based on the DNN from scratch. At each constructive step, an element to be filled into the solution is decided by the output of a trained DNN. The training of the DNN is done by reinforcement learning. Besides the DNN, the graph convolutional network (GCN [7]) and the attention model [37] have also been used in the technology for solving COPs, especially traveling salesman problem, such as [12, 21, 41]. Interested readers please refer to [13, 24] for more details on learning-to-optimize for COPs.

The learning-to-optimize technology has also been studied for modifying or generating parameters in the area of EA. These studies can be roughly grouped into two categories based on parameter type (either structural or numerical). In the first category, the DNN is used to control the structural parameters of EAs. For example, in DQ-HSES proposed by Zhang et al. [45, 47], Q-table and deep Q-network are applied to control the switching time for HSES [44]. In the second category, agents are used to control the numerical parameters of EAs. For example, Sun et al. [32] and Zhang et al. [46] proposed to use the DNN to control the algorithmic parameters of DE. Liu et al. [22] used the DNN to learn the mutation operator and the parameters for DE. Zhang et al. [48, 49] proposed a hybrid parametric model of the mutation operator for DE, in which neural networks decide the parameters of the adaptive mutation operators.

3 Preliminaries

The training of neural networks in existing learning-to-optimize literature for EAs is normally realized by modeling the evolutionary search process of EA as a Markov decision process and is carried out by reinforcement learning (RL) algorithms, such as Q-learning and policy gradient [34] and others.

RL aims to train an agent so that it can intelligently interact with the environment. In the game of GO [31], an agent plays intelligently according to the environmental change caused by its rival. The interaction of agent and environment can be modeled by MDP [27], which is denoted by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{R}, \pi, p, \mu_0, T\}$, where $\mathcal{S} \subseteq \mathbb{R}^{D_1}$ represents the state space, $\mathcal{A} \subseteq \mathbb{R}^{D_2}$ represents the action space, $\mathcal{R} \subseteq \mathbb{R}$ is the reward, π is the policy, p is the transition probability, μ_0 is the distribution of the initial state, and T is the time horizon for MDP. Here, D_1 and D_2 represent the dimensionality of the state and the action space, respectively.

Figure 1 shows the flowchart of a T-step MDP. At time step t, the agent observes state s_t and implements action $a_t \sim \pi(a_t|s_t, \boldsymbol{w})$, where $\boldsymbol{w}$ is the parameter of the policy. After the action, the environments return the next state based on the transition probability $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ and the reward r_{t+1}. A trajectory can be generated until the time limit, which is denoted by $\tau = \{s_0, a_0, s_1, r_1, \ldots, s_T, r_T\}$ in the sequel.

Mathematically, RL aims to find an optimal policy that maximizes the expectation of the cumulative reward $R(\tau) = \sum_{t=0}^{T-1} \gamma_t r_{t+1}$, where $\gamma_t \in (0, 1]$ is the time decay coefficient.

Many RL algorithms have been proposed in recent decades, such as Q-learning, DQN [25], policy gradient, actor-critic and others [34]. In this section, we only introduce the policy gradient (PG). The objective function of RL can be written as follows:

$$\max_{\boldsymbol{w}} \mathbb{E}_\tau[R(\tau)] = \max_{\boldsymbol{w}} \sum_{\tau} \mathrm{P}(\tau; \boldsymbol{w}) R(\tau), \tag{3.1}$$

where $\mathrm{P}(\tau; \boldsymbol{w})$ is the probability of the trajectory τ which is defined as

$$\mathrm{P}(\tau; \boldsymbol{w}) = \mu_0 \prod_{t=0}^{T-1} p(s_{t+1}|a_t, s_t)\pi(a_t|s_t, \boldsymbol{w}).$$

Policy gradient, as its name suggested, uses the gradient ascent method to optimize (3.1), i.e.,

$$\begin{aligned}\nabla_{\boldsymbol{w}} \sum_{\tau} \mathrm{P}(\tau; \boldsymbol{w}) R(\tau) &= \sum_{\tau} \nabla_{\boldsymbol{w}} \mathrm{P}(\tau; \boldsymbol{w}) R(\tau) = \sum_{\tau} \nabla_{\boldsymbol{w}} \ln[\mathrm{P}(\tau; \boldsymbol{w})] \mathrm{P}(\tau; \boldsymbol{w}) R(\tau) \\ &\approx \sum_{i=1}^{N_s} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{w}} \ln \pi(a_t^i|s_t^i, \boldsymbol{w}) R(\tau_i).\end{aligned} \tag{3.2}$$

In (3.2), the expectation is approximated by Monte Carlo sampling, where $\{\tau_i\}_{i=1}^{N_s}$ is the set of trajectories simulated by the policy π. In general, π is set as a Gaussian distribution

$$\pi(a|s, \boldsymbol{w}) = \frac{1}{(2\pi)^{\frac{D_2}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2}(a - \phi(s, \boldsymbol{w}))^{\mathrm{T}} \Sigma^{-1} (a - \phi(s, \boldsymbol{w})) \right\}, \tag{3.3}$$

where $\phi(\cdot, \boldsymbol{w})$ can be any smooth function such as a linear function or a neural network.

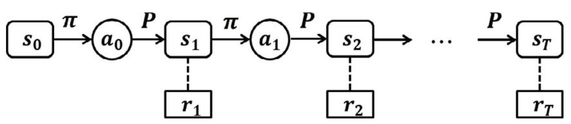


Figure 1 The illustration of a finite-time MDP

4 The method

This paper considers the black-box optimization problem $\min f(\boldsymbol{x})$, $\boldsymbol{x} \in \Omega \subset \mathbb{R}^n$, where the derivative information of f is not available.

4.1 A brief introduction of DE

DE works by iteratively carrying out mutation, crossover and selection. At the t-th generation, the current population is $X^t = \{\boldsymbol{x}_1^t, \dots, \boldsymbol{x}_N^t\}$. For each individual $\boldsymbol{x}_i^t$, a trial vector $\boldsymbol{v}_i^t$ is firstly obtained by using mutation. Commonly-used mutation operators include

$$\begin{aligned}
&\boldsymbol{v}_i^t = \boldsymbol{x}_{r_1^i}^t + F \cdot (\boldsymbol{x}_{r_2^i}^t - \boldsymbol{x}_{r_3^i}^t) \text{ (rand/1)},\\
&\boldsymbol{v}_i^t = \boldsymbol{x}_i^t + F \cdot (\boldsymbol{x}_{r_1^i}^t - \boldsymbol{x}_i^t) + F \cdot (\boldsymbol{x}_{r_2^i}^t - \boldsymbol{x}_{r_3^i}^t) \text{ (current-to-rand/1)},\\
&\boldsymbol{v}_i^t = \boldsymbol{x}_i^t + F \cdot (\boldsymbol{x}_{\text{best}}^t - \boldsymbol{x}_i^t) + F \cdot (\boldsymbol{x}_{r_1^i}^t - \boldsymbol{x}_{r_2^i}^t) \text{ (current-to-best/1)},\\
&\boldsymbol{v}_i^t = \boldsymbol{x}_{r_1^i}^t + F \cdot (\boldsymbol{x}_{\text{best}}^t - \boldsymbol{x}_{r_1^i}^t) + F \cdot (\boldsymbol{x}_{r_2^i}^t - \boldsymbol{x}_{r_3^i}^t) + F \cdot (\boldsymbol{x}_{r_4^i}^t - \boldsymbol{x}_{r_5^i}^t) \text{ (rand-to-best/2)},
\end{aligned} \tag{4.1}$$

where $\boldsymbol{x}_{\text{best}}^t$ is the best solution in X^t, $\boldsymbol{x}_{r_1^i}^t, \boldsymbol{x}_{r_2^i}^t$, $\boldsymbol{x}_{r_3^i}^t$, $\boldsymbol{x}_{r_4^i}^t$ and $\boldsymbol{x}_{r_5^i}^t$ are randomly selected individuals from X^t, and F is the scaling factor.

After mutation, the crossover operator is applied to the trial vectors. Commonly-used crossover is the binomial crossover [6]. The j-th component of the obtained vector $\boldsymbol{u}_i^t$ by crossover at the t-th generation is written as

$$\boldsymbol{u}_i^{j,t} = \begin{cases} \boldsymbol{v}_i^{j,t}, & \text{if } \text{rand}_j(0,1) \leqslant \text{CR or } j = j_{\text{rand}},\\ \boldsymbol{x}_i^{j,t}, & \text{otherwise}, \end{cases} \tag{4.2}$$

where $\boldsymbol{x}_i^{j,t}$ is the j-th component of $\boldsymbol{x}_i^t$, $\text{rand}_j(0,1)$ is a random value sampled from the uniform distribution $\mathcal{U}(0,1)$ for each j, and j_{rand} is a random integer sampled from $[1,n]$. The control parameter CR is called the crossover rate.

After crossover, the obtained individuals $\boldsymbol{u}_1^t, \dots, \boldsymbol{u}_N^t$ are selected as follows:

$$\boldsymbol{x}_i^{t+1} = \begin{cases} \boldsymbol{u}_i^t, & \text{if } f(\boldsymbol{u}_i^t) < f(\boldsymbol{x}_i^t),\\ \boldsymbol{x}_i^t, & \text{otherwise}. \end{cases} \tag{4.3}$$

For each individual $\boldsymbol{u}_i^t$, it is selected if its function values $f(\boldsymbol{u}_i^t)$ is smaller than that of the current individual $f(\boldsymbol{x}_i^t)$.

4.2 The generalized framework

The proposed generalized framework for operator selection is summarized in Algorithm 1. In the framework, at each generation, some observations are collected by using function **ObsCollection** (Line 4). Features are then extracted from these observations by a DNN, which is named ϕ with the parameter $\boldsymbol{w}$ (Line 5). The features are used as parameters for the policy π which is a K-dimensional probability distribution (Line 6). π represents the distribution of the selection probability of operators, which takes the output of ϕ as its parameter. A selection probability vector is then sampled from π (Line 6). The mutation operator is implemented based on the sampled probability vector (Line 7), while crossover (**Crossover**) and selection (**Selection**) are implemented sequentially (Lines 8 and 9, respectively). The algorithm terminates when the maximum number of generations has been reached, and returns the best solution found.

4.2.1 *Model into MDP*

Under the proposed generalized framework, its evolutionary process can be modeled as an MDP with the following tuple:

- State: observations collected (i.e., ξ_t).

- Action: the probability vector of the operator selection (i.e., $\boldsymbol{p}_t$).
- Policy: the probability distribution of the operator selection (i.e., π).
- Reward: if $t = T$, the reward is the negative logarithm of the best function value obtained by the policy (i.e., $-\log(f^*)$); otherwise is zero.

As mentioned above, the policy is often set to be a Gaussian in reinforcement learning simply because its gradient in (3.2) can be easily calculated. However, in our task, the dimension of action should be equal to the number of candidate operators as the action is a probability vector. A natural constraint is the summation of components of $\boldsymbol{p}_t$ is 1. To accommodate the constraint, Dirichlet distribution is used instead.

Algorithm 1 The generalized framework for operator selection for EA

```
Require: An objective function $f$, the maximum number of generations $T$, the parameters of the neural network $\boldsymbol{w}$, the
    population size $N$ and $K$ candidate operators $\mathcal{O}_K = \{\mathrm{OP}_k\}_{k=1}^K$;
Ensure: $f^*$.
 1: Initialize $X^0 \in \mathbb{R}^{n\times N}$ randomly and evaluate the population;
 2: Set $t \leftarrow 0$ and $f^* = \min\{f(\boldsymbol{x}_1^t), \dots, f(\boldsymbol{x}_N^t)\}$;
 3: while $t < T$ do
 4:     $\xi_t \leftarrow$ ObsCollection($X^t$);
 5:     $\kappa_t \leftarrow \phi(\xi_t, \boldsymbol{w})$;
 6:     Sample a $K$-dimensional probability vector $\boldsymbol{p}_t$ from the policy $\pi(\boldsymbol{p}|\kappa_t)$;
 7:     $V^t \leftarrow$ Mutation($X^t, \boldsymbol{p}_t, \mathcal{O}_K$);
 8:     $U^t \leftarrow$ Crossover($\mathrm{CR}, V^t, X^t$);
 9:     $X^{t+1} \leftarrow$ Selection($U^t, X^t, f$);
10:     $t \leftarrow t + 1$;
11:     $f^* \leftarrow \min\{f^*, \{f(\boldsymbol{x}_i^t)\}_{i=1}^N\}$;
12: end while
13: return $f^*$.
```

4.2.2 *Policy gradient with Dirichlet distribution*

In this subsection, the policy gradient algorithm for the Dirichlet policy is presented. We choose the policy to be a Dirichlet distribution simply because the selection vector of the operators should be a probability vector, i.e., the random variable $\boldsymbol{p} = (p_1, \dots, p_K)^{\mathrm{T}} \in \mathbb{R}^K$ in policy $\pi(\boldsymbol{p}|\kappa_t)$ should satisfy $\sum p_i = 1$ and $p_i \geqslant 0$, $1 \leqslant i \leqslant K$. Normally-used Gaussian distribution for the policy is not applicable.

The probability density function (PDF) of a Dirichlet random variable $X \in \mathbb{R}^K$ is written as

$$\mathrm{P}(X) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K x_i^{\alpha_i - 1}, \tag{4.4}$$

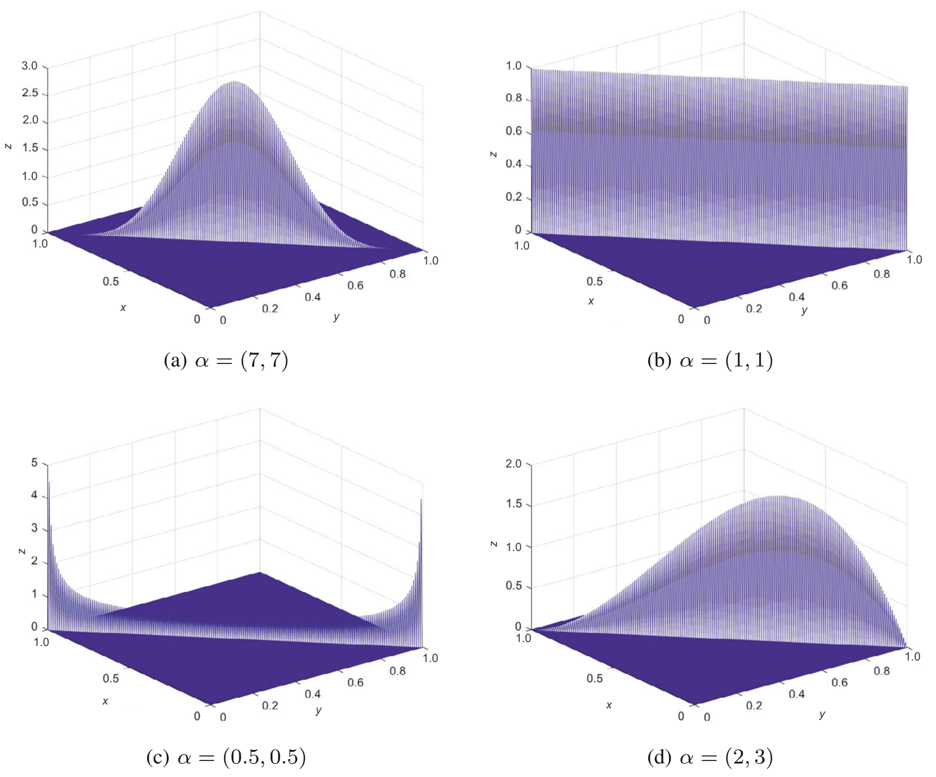
where $\alpha_0 = \sum_{i=1}^K \alpha_i$ and $\Gamma(\cdot)$ is the Gamma function. The Dirichlet random variable is denoted by $X \sim \mathrm{Dir}(X|\alpha_1, \dots, \alpha_K)$. Notice that the Dirichlet distribution is defined on a simplex

$$S = \left\{ X \,\middle|\, x_i \in [0,1],\ 1 \leqslant i \leqslant K \text{ and } \sum_{i=1}^K x_i = 1 \right\}.$$

Figure 2 shows four 2-D Dirichlet distributions with different parameters. From the figures, we see that when all the α_i's are larger than one, it is concentrated; when all the α_i's are equal to one, it is similar to a uniform distribution but constrained on a simplex; when all the α_i's are larger than zero and smaller than one, it is dispersed; when all the α_i's are not the same, it is asymmetrical.

Since there is no explicit variance parameter in the Dirichlet distribution like in Gaussian, we propose to introduce an extra parameter. It is known that the variance and covariance of X are respectively

$$\mathrm{var}(x_i) = \frac{\alpha_i(\alpha_0 - \alpha_i)}{\alpha_0^2(\alpha_0 + 1)}, \quad \mathrm{cov}(x_i, x_j) = \frac{\alpha_i(\alpha_0 - \alpha_j)}{\alpha_0^2(\alpha_0 + 1)}.$$


(a) $\alpha=(7,7)$ (b) $\alpha=(1,1)$

(c) $\alpha=(0.5,0.5)$ (d) $\alpha=(2,3)$

Figure 2 (Color online) The probability density function of 2-D Dirichlet distribution with different parameters

By multiplying all α_i's with a positive number M, we see that the corresponding variance and covariance become

$$\frac{\alpha_i(\alpha_0-\alpha_i)}{\alpha_0^2(M\alpha_0+1)},\quad \frac{\alpha_i(\alpha_0-\alpha_j)}{\alpha_0^2(M\alpha_0+1)},$$

respectively. When $M\to+\infty$, $\mathrm{var}(X_i)\to 0$ and $\mathrm{cov}(X_i,X_j)\to 0$. This implies that M can control the variance-covariance of the Dirichlet distribution. Figure 3 shows the Dirichlet distribution with different M.

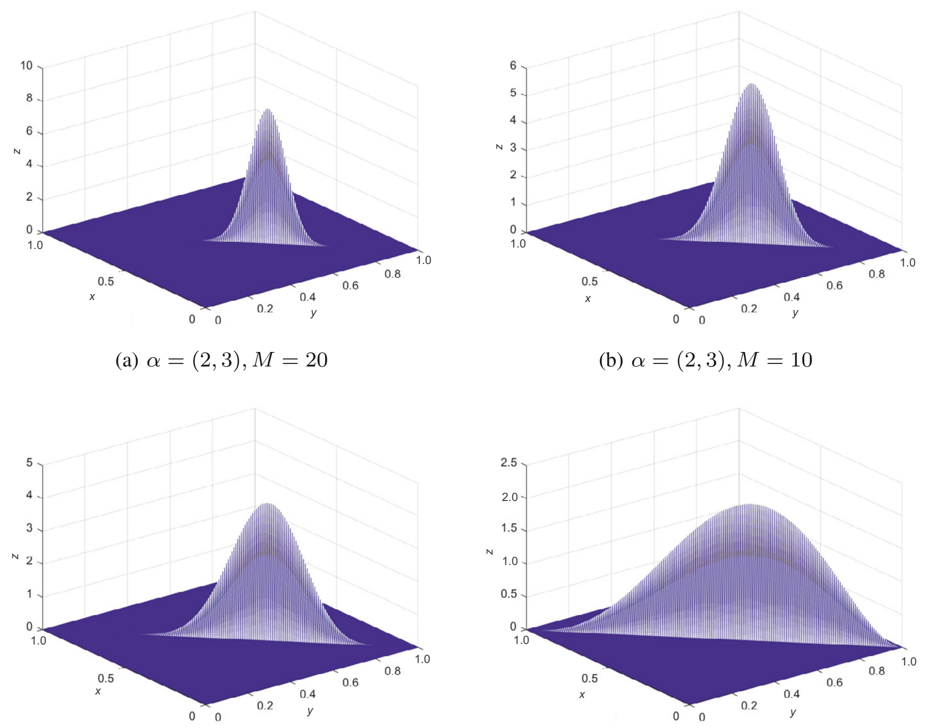
In the classical Gaussian policy gradient method, the output of the network ϕ is the mean of Gaussian policy (see (3.3)). Similarly, the Dirichlet policy can be written as

$$\pi(\boldsymbol{p}|s,\boldsymbol{w})=\frac{\Gamma(\sum_{i=1}^K\phi(\boldsymbol{s},\boldsymbol{w})_i)}{\prod_{i=1}^K\Gamma(\phi(\boldsymbol{s},\boldsymbol{w})_i)}\prod_{i=1}^K p_i^{\phi(s,\boldsymbol{w})_i-1}, \tag{4.5}$$

where $\boldsymbol{s}$ is the input of the network, $\phi(\boldsymbol{s},\boldsymbol{w})\in\mathbb{R}^K$, and $\phi(\boldsymbol{s},\boldsymbol{w})_i$ is its i-th element. We hence propose to modify (4.5) as follows:

$$\pi(\boldsymbol{p}|\boldsymbol{s},\boldsymbol{w})=\frac{\Gamma(\sum_{i=1}^K M\phi(\boldsymbol{s},\boldsymbol{w})_i+K)}{\prod_{i=1}^K\Gamma(M\phi(s,\boldsymbol{w})_i+1)}\prod_{i=1}^K p_i^{M\phi(s,\boldsymbol{w})_i}. \tag{4.6}$$

Combining (4.6) with (3.2), we can obtain the policy gradient method for the Dirichlet policy, which is summarized in Algorithm 2. In Algorithm 2, at each epoch, N_s trajectories are first generated by the Dirichlet policy with the parameter $\boldsymbol{w}_e$ (Line 3). The policy gradient with respect to the network parameter $\boldsymbol{w}$ is calculated based on the trajectories (Line 4), which is then updated (Line 5). The iteration continues until the maximum number of epochs has been reached.



(a) $\alpha = (2, 3), M = 20$ (b) $\alpha = (2, 3), M = 10$

(c) $\alpha = (2, 3), M = 5$ (d) $\alpha = (2, 3), M = 1$

Figure 3 (Color online) The probability density function of 2-D Dirichlet distribution with different scale parameters

Algorithm 2 Policy gradient with Dirichlet distribution

```
Require: The initial parameters of the neural network $\boldsymbol{w}_0$, $M > 0$, the learning rate $\alpha$, the sampling number $N_s$ and the
    maximum training epoch maxEpoch.
Ensure: $\boldsymbol{w}^*$.
1: Set $e \leftarrow 0$ ;
2: while $e <$ maxEpoch do
3:     Generate $N_s$ trajectories $\{\tau_i\}_{i=1}^{N_s}$ ($\tau_i = \{s_0^i, a_0^i, s_1^i, r_1^i, \ldots, s_T^i, r_T^i\}$) by the Dirichlet policy with parameter $\boldsymbol{w}_e$;
4:     $\Delta_{\boldsymbol{w}} \leftarrow \sum_{i=1}^{N_s} \sum_{t=0}^{T-1} \nabla_{\boldsymbol{w}} \ln \pi(a_t^i | s_t^i, \boldsymbol{w}) R(\tau_i)$;
5:     $\boldsymbol{w}_{e+1} \leftarrow \boldsymbol{w}_e + \alpha \Delta_{\boldsymbol{w}}$;
6:     $e \leftarrow e + 1$;
7: end while
8: return $\boldsymbol{w}^* \leftarrow \boldsymbol{w}_{\mathsf{maxEpoch}}$.
```

4.3 Case studies

As case studies, the framework is applied to classic DE and MPEDE [40], i.e., the selection mechanism proposed in the framework is embedded within classic DE and replaced that in MPEDE. The resultant algorithms, named PG-DE and PG-MPEDE, are summarized in Algorithms 3 and 4, respectively.

In PG-DE, in the first L generations, the selection probability vector $\boldsymbol{p}_t = [p_t^1, \ldots, p_t^K] \in \mathbb{R}^K$ is set to be $1/K$ for each operator and $\boldsymbol{n}_t = [n_t^1, \ldots, n_t^K]$ records the number of the operators used in the t-th generation (Line 9). Candidate operators are then assigned to the individuals randomly according to the selection probability $\boldsymbol{p}_t$ (Line 11), while the control parameters for the operators are fixed to be $F = 0.5$ and $\mathrm{CR} = 0.9$. After the assignment, the mutation and crossover are sequentially implemented to generate a set of trial vectors $U^t = \{\boldsymbol{u}_i^t\}_{i=1}^N$ (Line 13). Here, we use s_t^k to record the number of the k-th operator that succeeds in improving the function value, which is increased by one if it is successful (Line 17). After L generations, the information in recent L generations is collected (Line 6) and the probability for operator selection is calculated by (4.6) (Line 7). The algorithm terminates until the maximal number of generations are reached.

In PG-MPEDE, the population is divided into four sub-populations based on a set of the parameter $\{\lambda_k\}_{k=1}^4$ (Line 3). The first three sub-populations use the first two mutation operators as shown in (4.1), and the following mutation operator, named current-to-pbest/1:

$$\boldsymbol{v}_i^t = \boldsymbol{x}_i^t + F \cdot (\boldsymbol{x}_{\text{pbest}}^t - \boldsymbol{x}_i^t) + F \cdot (\boldsymbol{x}_{r_1}^t - \boldsymbol{x}_{r_2}^t) \text{ (current-to-pbest/1)}, \tag{4.7}$$

where $\boldsymbol{x}_{\text{pbest}}^t$ is one of the top $100p\%$ best solutions in the current population, and $\boldsymbol{x}_{r_1}^t$ and $\boldsymbol{x}_{r_2}^t$ are randomly selected individuals from the current population.

In this paper we set $\lambda_k = 0.2$, $k = 1, \ldots, 3$ and $\lambda_4 = 0.4$. Individuals in the fourth sub-population are assigned to the three operators according to an initial probability vector $\boldsymbol{p}_0 = [0.3, 0.3, 0.4]$ (Line 4). When one individual is assigned with the k-th operator, the individual can be regarded as merging into the k-th sub-population (Line 5), the size of the k-th sub-population is then increased by one (Line 6). For each sub-population, PG-MPEDE uses the following adaptive parameter control mechanism (Line 12) to control F and CR, i.e., F is sampled from a Cauchy distribution and CR is sampled from a Gaussian distribution. $U_k^t = \{\boldsymbol{u}_{i,k}^t\}_{i=1}^{N_k}$ is obtained by implementing the k-th mutation operator and then crossover (see (4.2)) on $X_k^t = \{\boldsymbol{x}_{i,k}^t\}_{i=1}^{N_k}$ (Line 14). The selection operation is implemented from Line 17 to Line 26. Like in PG-DE, s_t^k records the number of the k-th operator that succeeds in improving the function value (Line 22). Meanwhile, $S_{F,k}$ and $S_{\text{CR},k}$ are used to record the F and CR values that succeed in improving the function value (Line 23). For every L generations, the information will be collected (Line 28) and (4.6) will be used to sample the probability vector $\boldsymbol{p}_t$ (Line 30). Finally, the median of the Cauchy distribution and mean of the Gaussian distribution are updated in Lines 32 and 33, where mean$L$ is the Lehmer mean which is defined as

$$\text{mean}_L(\boldsymbol{x}) = \frac{\sum_i x_i^2}{\sum_i x_i}.$$

Algorithm 3 PG-DE

```
Require: An objective function $f$, the maximum number of generations $T$, the learning period $L$, the parameters of neural
    network $\boldsymbol{w}$, the candidate operators $\{\text{OP}_k\}_{k=1}^K$ and the population size $N$;
Ensure: $f^*$;
 1: Initialize $X^0$ randomly; Set $t \leftarrow 0$;
 2: $f^* \leftarrow \min\{f(\boldsymbol{x}_i^t) \mid i = 1, \ldots, N\}$;
 3: while $t < T$ do
 4:     Set $\boldsymbol{s}_t \in \mathbb{R}^{2K} \leftarrow 0$;
 5:     if $t \geqslant L$ then
 6:         Set $\boldsymbol{s}_t \leftarrow (\sum_{j=t-L}^{t-1} s_j^k, \sum_{j=t-L}^{t-1} n_j^k, 1 \leqslant k \leqslant K)$;
 7:         Sample $\boldsymbol{p}_t = [p_t^1, \ldots, p_t^K]$ from $\pi(\boldsymbol{p} \mid \boldsymbol{s}_t, \boldsymbol{w})$ and compute $n_t^k \leftarrow \lfloor N p_t^k \rfloor, k = 1, \ldots, K$;
 8:     else
 9:         Set $p_t^k = 1/K$ and $n_t^k \leftarrow \lfloor N p_t^k \rfloor$, $k = 1, \ldots, K$;
10:     end if
11:     For each operator $\text{OP}_k, 1 \leqslant k \leqslant K$, assign $\lfloor N p_t^k \rfloor$ individuals to it randomly;
12:     for $i = 1 \to N$ do
13:         Generate a trial vector $\boldsymbol{u}_i^t$ by using $\text{OP}_{k_i}$ where $\text{OP}_{k_i}$ is the operator assigned to $\boldsymbol{x}_i^t$ and the binomial crossover
            stated in (4.2);
14:         if $f(\boldsymbol{x}_i^t) < f(\boldsymbol{u}_i^t)$ then
15:             $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{x}_i^t$;
16:         else
17:             $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{u}_i^t$, $s_t^{k_i} \leftarrow s_t^{k_i} + 1$;
18:         end if
19:     end for
20:     $t \leftarrow t + 1$;
21:     $f^* \leftarrow \min\{f^*, f(\boldsymbol{x}_i^t) \mid i = 1, \ldots, N\}$;
22: end while
23: return $f^*$.
```

Algorithm 4 PG-MPEDE

```
Require: An objective function $f$, the maximum number of generations $T$, the learning period $L$, the network parameter
    $\boldsymbol{w}$, the candidate operators $\mathcal{O}_3 = \{\mathrm{OP}_k\}_{k=1}^3$ and the population size $N$;
Ensure: $f^*$;
 1: Initialize $X^0$ randomly; Set $t \leftarrow 0$ and $f^* \leftarrow \min\{f(\boldsymbol{x}_i^0) \mid i = 1, \ldots, N\}$;
 2: Initialize $\lambda_k$, set $N_k \leftarrow \lfloor N\lambda_k \rfloor$, $k = 1, \ldots, 4$;
 3: Randomly divide $X^t$ into four sub-populations as $X_k^t = \{\boldsymbol{x}_{i,k}^t\}_{i=1}^{N_k}$;
 4: Initialize $\boldsymbol{p}_t = \{p_t^k\}_{k=1}^3$ and assign individuals to the candidate operators according to $\boldsymbol{p}_t$;
 5: Merge individuals in $X_4^t$ to the $k$-th sub-population if an individual is assigned with $\mathrm{OP}_k$;
 6: Set $N_k \leftarrow N_k + \lfloor p_t^k \times N_4 \rfloor$, $k = 1, \ldots, 3$;
 7: Set $\mu_{\mathrm{CR}_k} \leftarrow 0.5$, $\mu_{F_k} \leftarrow 0.5$, $k = 1, \ldots, 3$;
 8: while $t < T$ do
 9:     $s_t^k \leftarrow 0$, $k = 1, \ldots, 3$;
        % Generate Offsprings
10:     for $k = 1 \rightarrow 3$ do
11:         for $i = 1 \rightarrow N_k$ do
12:             $F_{i,k} \leftarrow \mathcal{C}(\mu_{F_k}, 0.1)$, $\mathrm{CR}_{i,k} \leftarrow \mathcal{N}(\mu_{\mathrm{CR}_k}, 0.1)$;
13:         end for
14:         Obtain $U_k^t = \{\boldsymbol{u}_{i,k}^t\}_{i=1}^{N_k}$ by using $\mathrm{OP}_k$ and binomial crossover operator on $X_k^t = \{\boldsymbol{x}_{i,k}^t\}_{i=1}^{N_k}$;
15:         Set $S_{F,k} \leftarrow \emptyset$,$S_{\mathrm{CR},k} \leftarrow \emptyset$;
16:     end for
        % Selection
17:     for $k = 1 \rightarrow 3$ do
18:         for $i = 1 \rightarrow N_k$ do
19:             if $f(\boldsymbol{x}_{i,k}^t) < f(\boldsymbol{u}_{i,k}^t)$ then
20:                 $\boldsymbol{x}_{i,k}^{t+1} \leftarrow \boldsymbol{x}_{i,k}^t$;
21:             else
22:                 $\boldsymbol{x}_{i,k}^{t+1} \leftarrow \boldsymbol{u}_{i,k}^t$, $s_t^k \leftarrow s_t^k + 1$;
23:                 $S_{\mathrm{CR},k} \leftarrow S_{\mathrm{CR},k} \cup \{\mathrm{CR}_{i,k}\}$, $S_{F,k} \leftarrow S_{F,k} \cup \{F_{i,k}\}$;
24:             end if
25:         end for
26:     end for
        % Calculate Selection Probability and Assign Individuals
27:     if $\mathrm{mod}(t, L) == 0$ and $t > 0$ then
28:         $\boldsymbol{s}_t \leftarrow \{\sum_{j=t-L+1}^t s_j^k, L \times N_k\}_{k=1}^3$;
29:         Randomly divide the whole population $X^t$ into $X_k^t$ according to $\lfloor N \times \lambda_k \rfloor, 1 \leqslant k \leqslant 4$
30:         Sample $\boldsymbol{p}_t$ from $\pi(\boldsymbol{p} \mid \boldsymbol{s}_t, \boldsymbol{w})$ and assign individuals of $X_4^t$ to mutation operators randomly according to $\boldsymbol{p}_t$;
31:         Merge the individuals of $X_4^t$ into the first three sub-populations;
32:         Set $N_k \leftarrow N \times \lambda_k + \lfloor p_t^k \times N_4 \rfloor, k = 1, \ldots, 3$;
33:     end if
34:     Set $\mu_{F_k} \leftarrow (1-c)\mu_{F_k} + c \cdot \mathrm{mean}L(S_{F,k}), k = 1, 2, 3$;
35:     Set $\mu_{\mathrm{CR}_k} \leftarrow (1-c)\mu_{\mathrm{CR}_k} + c \cdot \mathrm{mean}(S_{\mathrm{CR},k}), k = 1, 2, 3$;
36:     $f^* \leftarrow \min\{f^*, f(\boldsymbol{x}_i^t) \mid i = 1, \ldots, N\}$, $t \leftarrow t + 1$;
37: end while
38: return $f^*$.
```

4.4 Relationship between the proposed algorithms and classical DEs

We first present the following theorem.

Theorem 4.1. *Assume $p(X)$ is a d-dimensional Dirichlet distribution with parameters $M\alpha_1 + 1$, $\ldots, M\alpha_d + 1$, where $\alpha_i > 0$, $i = 1, \ldots, d$. For all $\delta > 0$, we have*

$$\lim_{M \rightarrow +\infty} \mathrm{P}\{\|X - m(X)\|_2 > \delta\} = 0,$$

where $m(X)$ is the mode of $p(X)$.

Proof. Please see Appendices A–C. □

In SaDE, the success rate of each operator $S_i, i = 1, \ldots, K$ is computed at each generation by taking the ratio between the success number of the i-th operator (denoted by s_i) and the number of the operator that has been used in the previous several generations (denoted by n_i). The success rate is then used to construct the selection probability as

$$p_i = \frac{S_i}{\sum_i S_i}, \quad i = 1, \ldots, K.$$

Note that the mode of a Dirichlet distribution $X \sim \mathrm{Dir}(X \mid \alpha_1, \ldots, \alpha_K)$ is

$$\left[\frac{\alpha_1 - 1}{\sum_{i=1}^K \alpha_i - K}, \ldots, \frac{\alpha_i - 1}{\sum_{i=1}^K \alpha_i - K}, \ldots, \frac{\alpha_K - 1}{\sum_{i=1}^K \alpha_i - K}\right].$$

This indicates that the selection probability of the candidate operators in SaDE is the mode of the Dirichlet distribution with parameters $S_1 + 1, \ldots, S_K + 1$.

Notice that in PG-DE, after obtaining the parameters of the Dirichlet distribution (i.e., $\phi(\boldsymbol{s}, \boldsymbol{w})$), we see that the selection probability is sampled from the Dirichlet distribution. Theorem 4.1 shows that when M is large, the sampled point is close to its mode. Note that the mode of the Dirichlet distribution used in PG-DE (i.e., (4.6)) is

$$m(X) = \left[\frac{\phi(\boldsymbol{s}, \boldsymbol{w})_1}{\sum_j \phi(\boldsymbol{s}, \boldsymbol{w})_j}, \ldots, \frac{\phi(\boldsymbol{s}, \boldsymbol{w})_K}{\sum_j \phi(\boldsymbol{s}, \boldsymbol{w})_j}\right]^{\mathrm{T}}. \tag{4.8}$$

Therefore, taking $\phi(s, \boldsymbol{w})_i$ to be S_i indicates that PG-DE degenerates to SaDE when M is large enough.

In consideration of MPEDE, the improvement of each operator $\tilde{S}_i = \frac{\Delta f_i}{n_i}$, $i = 1, \ldots, K$ is computed at each generation, where n_i denotes the number of each operator that has been used in the previous several generations and Δf_i denotes the function value improvement. The improvement is then used to construct the selection probability as

$$p_i = \frac{\tilde{S}_i}{\sum_i \tilde{S}_i}, \quad i = 1, \ldots, K.$$

The operator with the largest p_i will be assigned to the fourth sub-population. This indicates that it is the index, which corresponds to the maximum component of the mode of $\mathrm{Dir}(\tilde{S}_1 + 1, \ldots, \tilde{S}_K + 1)$ that is assigned to the selected operator.

Different from MPEDE, individuals in the fourth sub-population are assigned according to the sampled probability from $\mathrm{Dir}(M\phi(\boldsymbol{s}, \boldsymbol{w})_1 + 1, \ldots, M\phi(\boldsymbol{s}, \boldsymbol{w})_K + 1)$ in PG-MPEDE. Thus we can conclude that if we take $\phi(s, \boldsymbol{w})_i$ to be $\tilde{S}_i$, the selection probability $p_i, i = 1, \ldots, K$ of PG-MPEDE is the same as MPEDE when M is large enough.

4.5 Training the neural network

It is proved that SaDE is a specific case of PG-DE. Specifically, taking $\phi(s, \boldsymbol{w})_i$ to be $S_i, i = 1, \ldots, K$, we see that PG-DE degenerates to SaDE when the scale parameter $M \to +\infty$. SaDE can thus be regarded as *a prior* knowledge, or expert experience, for the learning of the neural network ϕ.

Since the selection mechanism of SaDE is considered to be *a prior* knowledge, it should be made usable. In other words, the output of the network $\phi(s, \boldsymbol{w})$ shall be close to the success rate $S = [S_1, \ldots, S_K]$ in SaDE. Each element S_i in S is a ratio, which can be denoted by $\frac{a}{b}$. Algorithm 5 summarizes the training procedure. The procedure is composed of two phases including the supervised learning phase and the reinforcement learning phase. In the supervised learning phase, the ratio, i.e., the division between two positive numbers is to be learned by ϕ (Line 4 to Line 10). To learn the division operation, we randomly generate a set of positive number $b_i \in (0, 1]$ and $a_i \in (0, b_i]$, $i = 1, \ldots, N_D$ (Line 3). The generated data, i.e., $\boldsymbol{x}_i = [a_i, b_i]^{\mathrm{T}}$ and $\boldsymbol{y}_i = a_i/b_i$, $i = 1, \ldots, N_D$, are used as the training data (Line 4). With these data, the network parameter $\boldsymbol{w}$ is updated in Line 8. The learned parameter $\boldsymbol{w}$ is used as initialization for the reinforcement learning phase. The algorithm with neural network is applied to generate a set of trajectories in Line 16. According to the definition, the cumulative reward $R(\tau_i)$ is calculated in Line 17, where we minus the known global minima f_m^*. Finally, the policy gradient is used to update the parameter in Line 19.

Algorithm 5 The training algorithm for the parameter of ϕ

```
Require: A set of training functions $\{f_m\}_{m=1}^M$ and its optimal values $\{f_m^*\}_{m=1}^M$, the learning rates $\alpha$ and $\beta$, the batch size $B$, the maximum number of epochs maxEpoch, the number of sampling $n_{RL}$, the maximum number of epochs of supervised learning $L^{sp}$, the number of data in supervised learning $N_D$ and an optimization algorithm $\mathcal{A}$.
Ensure: $\boldsymbol{w}^*$.
 1: Initialize randomly the network weight $\boldsymbol{w}_0$;
 2: Set $l \leftarrow 0$;
 3: Randomly generate a set of positive number $b_i \in (0,1]$ and $a_i \in (0,b_i]$, $i=1,\ldots,N_D$;
 4: Generate data $\{\boldsymbol{x}_i=[a_i,b_i]^{\mathrm{T}}, \boldsymbol{y}_i=a_i/b_i\}_{i=1}^{N_D}$;
 5: % The supervised-learning phase
 6: while $l < L^{sp}$ do
 7:     Randomly select $B$ data pairs $\{\boldsymbol{x}_{j_s},\boldsymbol{y}_{j_s}\}_{j_s=1}^B$;
 8:     Update $\boldsymbol{w}_{l+1} \leftarrow \boldsymbol{w}_l - \frac{\alpha}{B}\sum_{j_s=1}^B \nabla_{\boldsymbol{w}}\|\phi(\boldsymbol{x}_{j_s},\boldsymbol{w}_l)-\boldsymbol{y}_{j_s}\|^2$;
 9:     $l \leftarrow l+1$;
10: end while
11: % The reinforcement-learning phase
12: Set $e \leftarrow 0$, $\boldsymbol{w}_0 \leftarrow \boldsymbol{w}_l$;
13: while $e <$ maxEpoch do
14:     for $m = 1 \to M$ do
15:         for $i = 1 \to n_{RL}$ do
16:             Run the algorithm $\mathcal{A}$ on the function $f_m$ for $T$ steps to obtain the best solution $f_{i,m}^b$ and a trajectory $\tau_{i,m}=\{s_{0,m}^i,a_{0,m}^i,\ldots,a_{T-1,m}^i,s_{T,m}^i\}$;
17:             Set $R(\tau_{i,m}) \leftarrow -\log(f_{i,m}^b - f_m^*)$;
18:         end for
19:         Update $\boldsymbol{w}_{e+1} \leftarrow \boldsymbol{w}_e + \beta\sum_{i=1}^{n_{RL}}\sum_{t=0}^{T-1}\nabla_{\boldsymbol{w}}\ln\pi(a_{t,m}^i|s_{t,m}^i,\boldsymbol{w})R(\tau_{i,m})$;
20:         $e \leftarrow e+1$;
21:     end for
22: end while
23: Return $\boldsymbol{w}^* \leftarrow \boldsymbol{w}_{\mathrm{maxEpoch}}$.
```

5 Experimental results

The CEC 2018 test suite[2)] is used to evaluate the proposed algorithms. The test suite contains 29 test functions with 10D, 30D, 50D and 100D. These functions are divided into four categories: F_1 and F_3 are uni-modal functions, F_4–F_{10} are multi-modal functions, F_{11}–F_{20} are hybrid functions, and F_{21}–F_{30} are composition functions. In this paper, the test functions are roughly divided into three groups by merging the first two categories into one, since uni-modal functions and multi-modal functions are relatively simple functions[3)]. We name the three categories as

$$S_1=\{F_1,F_3-F_{10}\},\quad S_2=\{F_{11}-F_{20}\}\quad\text{and}\quad S_3=\{F_{21}-F_{30}\}.$$

5.1 Training details

In our experiments, an MLP with two hidden layers is used for PG-DE. The numbers of hidden units are 36 and 100, respectively, and the activation functions are the hyperbolic tangent function and sigmoid function, respectively. For PG-MPEDE, an MLP with four hidden layers is used, the numbers of hidden units are 16, 32, 16 and 8, respectively; the activation function for the first three layers is tangent, while the last layer is sigmoid. Here, we show the hyper-parameters used in Algorithm 5. $T=\frac{10{,}000D}{N}$, where D is the dimension of the test problems, and N is the population size. Learning rates $\alpha=0.01$ and $\beta=0.01$, the batch size $B=64$, the maximum number of epochs maxEpoch $=100$, the number of sampling $n_{RL}=10$, the maximum number of epochs of supervised learning $L^{sp}=50{,}000$ and the number of data in supervised learning $N_D=10{,}000$.

2) https://github.com/P-N-Suganthan/CEC2018

3) According to the results of CEC competitions, most algorithms can find the global optimum of the functions in the first two categories.

5.2 PG-DE

Four candidate operators are used in PG-DE, including rand/1, current-to-rand/1, rand-to-best/2 and current-to-best/1. The first three operators are used in SaDE. The operator current-to-best/1 is a widely used operator in DE studies.

We train the two neural networks for 10D and 30D functions, respectively. For each dimension, the first neural network is trained on F_5, and the network is tested on S_1 except F_5. The second neural network is trained on F_{15}, and the network is tested on S_2 except F_{15}. We compare PG-DE with SaDE, and the two algorithms are run 51 times. The medians and standard deviations of error values (i.e., the difference between the obtained optimum and the known global optimum) averaged over 51 runs are summarized in Table 1. The Wilcoxon rank sum hypothesis test is also taken between PG-DE and SaDE at a significance level of 5%, where †/§ means SaDE performs significantly better/worse than PG-DE, and $\approx$ means that there is no significant difference between the two algorithms. The results of the hypothesis test are summarized in the last line of Table 1. Meanwhile, BM is used as a metric, which records the number of functions for which an algorithm obtains the best median value. The larger BM is, the better performance an algorithm has.

From the table, we can see that for 10D test functions, PG-DE performs better on seven functions and worse on four functions than SaDE. For 30D test functions, PG-DE performs better on 11 functions and worse on eight functions than SaDE. PG-DE has larger BM values than SaDE. Note that the parameters in PG-DE are fixed ($F = 0.5$, $\mathrm{CR} = 0.9$), while SaDE adaptively controls these parameters. Thus we may conclude that the trained networks can help select better operators than the classic heuristic design.

5.3 PG-MPEDE

For PG-MPEDE, all the test functions in 10D and 30D of the CEC 2018 test suite are considered. The training functions are similar to that in PG-DE. The first network is trained on F_5, and tested on other functions in S_1. The second network is trained on F_{15}, and tested on other functions in S_2. The third network is trained on F_{25}, and tested on other functions in S_3. The statistical results are summarized in Table 2. The table shows that for 10D test functions, PG-MPEDE performs significantly better on 23 functions and worse on only one function than MPEDE. For 30D test functions, PG-MPEDE performs significantly better on nine functions and worse on seven functions than MPEDE. From the observations, we may conclude that the trained networks can significantly improve the performance of MPEDE.

5.4 Comparison results with developed EAs

To further verify the performance of PG-MPEDE, it is compared with some developed EAs. The compared algorithms are grouped into three categories.

- **DEs with AOS**, including SaDE [28], cDE [36], CoDE [39] and MPEDE [40]. These algorithms all use the AOS mechanism.
- **DEs with APC**, including JADE [50], which is a DE with an adaptive parameter control mechanism.
- **Other EA**, including HSES [44], which is the winning algorithm in the CEC 2018 competition.

The average performance score (APS) [35] is used as a metric for algorithm comparison. APS is defined as follows. Assuming the comparison of M algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_M$ on L test functions $F_1, \ldots, F_L$ based on the error values obtained by them for several runs. For each algorithm $\mathcal{A}_m$, $m \in \{1, \ldots, M\}$, if $\mathcal{A}_j$, $j \in \{1, \ldots, M\} \setminus \{m\}$ outperforms $\mathcal{A}_m$ on the l-th function F_l, $l \in \{1, \ldots, L\}$ with statistical significance, then $\delta^l_{m,j}$ is set to be 1, otherwise $\delta^l_{m,j}$ is set as 0. The performance score of the algorithm $\mathcal{A}_m$ on the function F_l is defined as

$$PS_l(\mathcal{A}_m) = \sum_{j \in \{1,\ldots,M\} \setminus \{m\}} \delta^l_{m,j}.$$

Table 1 Results obtained by SaDE and PG-DE on test functions over 51 runs

	10D			
	SaDE		PG-DE	
	Median	std	Median	std
F_1	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00
F_3	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00
F_4	**1.17E−07**†	5.15E−01	4.44E+00	1.35E+00
F_5	7.41E+00§	1.49E+00	**2.98E+00**	3.54E+00
F_6	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	2.81E−07
F_7	**2.04E+01**†	2.35E+00	2.32E+01	3.25E+00
F_8	6.93E+00§	1.70E+00	**2.98E+00**	3.66E+00
F_9	**0.00E+00**$^{\approx}$	0.00E+00	0.00E+00	0.00E+00
F_{10}	4.05E+02§	9.85E+01	**1.42E+01**	2.29E+02
F_{11}	4.09E−04§	4.60E−01	**0.00E+00**	7.86E−01
F_{12}	1.18E+02$^{\approx}$	5.98E+01	**1.13E+01**	6.91E+01
F_{13}	**9.94E−01**$^{\approx}$	2.73E+00	3.11E+00	2.73E+00
F_{14}	4.34E−01§	3.60E−01	**0.00E+00**	3.55E+00
F_{15}	**1.00E−02**†	1.11E−01	5.59E−02	3.10E−01
F_{16}	1.44E+00§	4.08E−01	**7.21E−01**	1.66E+01
F_{17}	2.49E+00§	1.07E+00	**4.33E−01**	2.85E+00
F_{18}	**4.05E−01**$^{\approx}$	4.72E−01	4.75E−01	7.30E+00
F_{19}	2.03E−02$^{\approx}$	1.02E−02	**1.97E−02**	4.27E−02
F_{20}	**0.00E+00**†	0.00E+00	3.12E−01	3.63E−01
BM	10		**12**	
†/ ≈ /§			4/8/7	

	30D			
	SaDE		PG-DE	
	Median	std	Median	std
F_1	**0.00E+00**†	5.75E−02	2.34E+01	3.70E+02
F_3	2.01E−04§	1.65E−02	**3.17E−07**	7.15E−03
F_4	**6.79E+01**†	3.65E+01	1.19E+02	2.22E+01
F_5	6.11E+01§	2.06E+01	**3.18E+01**	7.78E+00
F_6	**0.00E+00**†	0.00E+00	6.92E−02	2.39E−01
F_7	**1.12E+02**†	1.55E+01	1.73E+02	6.15E+01
F_8	6.52E+01§	1.76E+01	**2.70E+01**	3.19E+01
F_9	**0.00E+00**†	5.01E−01	9.15E+00	1.08E+01
F_{10}	**4.27E+03**†	2.77E+02	6.37E+03	1.39E+03
F_{11}	5.51E+01§	2.38E+01	**4.04E+01**	2.77E+01
F_{12}	9.76E+03§	7.36E+03	**2.06E+02**	4.04E+02
F_{13}	6.39E+01§	5.76E+01	**5.41E+01**	2.39E+01
F_{14}	**4.21E+01**†	1.47E+01	7.01E+01	8.12E+00
F_{15}	3.73E+01§	3.56E+01	**3.78E+00**	3.91E+00
F_{16}	**4.90E+02**†	1.15E+02	8.53E+02	4.65E+02
F_{17}	7.00E+01§	2.03E+01	**6.24E+01**	2.37E+01
F_{18}	7.29E+01§	7.91E+01	**2.53E+01**	4.16E+00
F_{19}	1.94E+01§	1.75E+01	**5.07E+00**	1.50E+00
F_{20}	8.01E+01§	4.81E+01	**2.11E+01**	2.10E+01
BM	8		**11**	
†/ ≈ /§			8/0/11	

Table 2 Results obtained by MPEDE and PG-MPEDE on test functions over 51 runs

	10D				30D			
	MPEDE		PG-MPEDE		MPEDE		PG-MPEDE	
	Median	std	Median	std	Median	std	Median	std
F_1	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00
F_3	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00§	0.00E+00	**0.00E+00**	0.00E+00
F_4	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00	5.85E+01§	1.81E+01	**5.85E+01**	3.09E+01
F_5	6.30E+00§	1.34E+00	**4.00E+00**	1.35E+00	**2.68E+01**†	6.64E+00	3.18E+01	9.24E+00
F_6	2.32E–05§	5.99E–06	**1.23E–06**	1.03E–06	0.00E+00§	2.01E–08	**0.00E+00**	0.00E+00
F_7	1.73E+01§	1.99E+00	**1.47E+01**	1.82E+00	**5.41E+01**†	6.10E+00	6.03E+01	1.10E+01
F_8	6.27E+00§	1.47E+00	**3.98E+00**	1.39E+00	**2.68E+01**$^{\approx}$	8.34E+00	3.08E+01	1.03E+01
F_9	**0.00E+00**$^{\approx}$	0.00E+00	**0.00E+00**	0.00E+00	**0.00E+00**$^{\approx}$	6.55E–02	**0.00E+00**	2.93E–02
F_{10}	2.87E+02§	9.99E+01	**1.00E+02**	9.66E+01	2.76E+03§	3.73E+02	**2.28E+03**	4.48E+02
F_{11}	2.39E+00§	5.40E–01	**1.43E+00**	9.05E–01	**1.89E+01**†	1.33E+01	3.23E+01	2.90E+01
F_{12}	7.01E+00§	1.34E+01	**6.04E–01**	6.48E+00	9.80E+02§	3.44E+02	**6.60E+02**	3.50E+02
F_{13}	5.00E+00§	1.97E+00	**2.03E+00**	2.20E+00	2.05E+01$^{\approx}$	8.71E+00	**1.91E+01**	8.58E+00
F_{14}	4.53E+00§	1.20E+00	**1.38E+00**	1.24E+00	**1.19E+01**†	1.02E+01	3.59E+01	1.28E+01
F_{15}	7.28E–01§	2.50E–01	**1.34E–01**	6.03E–02	7.26E+00$^{\approx}$	3.87E+00	**6.42E+00**	3.25E+00
F_{16}	2.20E+00§	8.94E–01	**6.43E–01**	3.92E–01	**3.43E+02**$^{\approx}$	2.21E+02	3.73E+02	2.14E+02
F_{17}	7.55E+00§	2.53E+00	**1.11E+00**	5.88E–01	5.42E+01§	1.67E+01	**4.11E+01**	2.48E+01
F_{18}	2.72E+00§	1.38E+00	**2.89E–01**	2.56E–01	**2.46E+01**†	7.07E+00	2.86E+01	1.16E+01
F_{19}	4.92E–01§	2.12E–01	**2.42E–02**	1.59E–02	7.41E+00$^{\approx}$	1.67E+00	**6.73E+00**	2.46E+00
F_{20}	8.01E–01§	5.65E–01	**2.34E–04**	4.38E–04	4.38E+01§	3.41E+01	**2.69E+01**	4.88E+01
F_{21}	1.00E+02§	3.93E+01	**1.00E+02**	3.20E+01	**2.28E+02**$^{\approx}$	7.38E+00	2.30E+02	1.02E+01
F_{22}	1.00E+02§	1.96E+01	**1.00E+02**	1.96E+01	1.00E+02§	0.00E+00	**1.00E+02**	3.47E–01
F_{23}	3.06E+02§	2.23E+00	**3.03E+02**	2.24E+00	**3.77E+02**$^{\approx}$	1.07E+01	3.82E+02	1.33E+01
F_{24}	3.28E+02§	1.17E+02	**1.00E+02**	1.18E+02	**4.41E+02**$^{\approx}$	8.85E+00	4.46E+02	9.82E+00
F_{25}	3.98E+02$^{\approx}$	1.23E+01	**3.98E+02**	1.24E+01	**3.86E+02**†	7.38E–02	3.86E+02	1.61E–01
F_{26}	3.00E+02§	1.69E–08	**3.00E+02**	0.00E+00	**1.20E+03**$^{\approx}$	1.20E+02	1.21E+03	3.98E+02
F_{27}	3.89E+02§	2.80E–01	**3.89E+02**	2.91E–01	**5.02E+02**$^{\approx}$	5.94E+00	5.04E+02	7.77E+00
F_{28}	3.00E+02§	7.40E+01	**3.00E+02**	5.56E–07	3.00E+02§	5.15E+01	**3.00E+02**	5.43E+01
F_{29}	2.50E+02$^{\approx}$	5.51E+00	**2.49E+02**	6.81E+00	**4.51E+02**†	2.66E+01	4.57E+02	4.29E+01
F_{30}	3.95E+02§	1.46E+00	**3.94E+02**	3.98E+00	**1.98E+03**$^{\approx}$	9.72E+01	**1.98E+03**	8.40E+01
BM	4		**29**		**17**		15	
$\dagger/\approx/\S$			1/5/23				7/13/9	

Table 3 The APS values obtained by PG-MPEDE, cDE, MPEDE, HSES, CoDE, JADE and SaDE

Dimensions	PG-MPEDE	cDE	MPEDE	HSES	CoDE	JADE	SaDE
10D	*1.2069*	2.4138	3.1379	2.8966	**0.7241**	2.7931	2.2414
30D	1.6207	3.5862	*1.2759*	**0.7931**	2.3448	2.4138	4.1379
Average	**1.4138**	3.0000	2.2069	1.8448	*1.5345*	2.6035	3.1897

$PS_l(\mathcal{A}_m)$ presents the number of algorithms which outperform $\mathcal{A}_m$ on function F_l. The APS of the algorithm $\mathcal{A}_m$ is taken as the average of $PS_l(\mathcal{A}_m)$ over l

$$\mathrm{APS}(\mathcal{A}_m) = \frac{1}{L}\sum_{l=1}^{L} PS_l(\mathcal{A}_m).$$

Thus the **smaller** the APS value is, the **better** performance an algorithm has.

The APS values obtained by PG-MPEDE, cDE, MPEDE, HSES, CoDE, JADE and SaDE are summarized in Table 3. The table shows that PG-MPEDE obtains the smallest average APS value, which implies that PG-MPEDE can perform competitively among these developed EAs.

5.5 Time complexity

The CPU time of MPEDE and PG-MPEDE for running F_1–F_{30} once is shown in Table 4. It is seen that PG-MPEDE requires a little bit more time than MPEDE, which implies that the trained network can adaptively select the operators without sacrificing too much complexity.

5.6 Ablation study

In the training algorithm (Algorithm 5), the supervised learning phase is used to learn from expert experiences. In this subsection, we study the effect of supervised learning. We train two networks for PG-DE on F_5 with 10D with and without supervised learning (SL), respectively. The two PG-DEs (with and without supervised learning) are tested on the test function group S_1 with 10D. The experimental results are summarized in Table 5. It is seen that PG-DE with supervised learning performs better on four functions and worse on one function. Thus we may conclude that supervised learning improves the efficiency of network training.

5.7 Optimization curves

In this subsection, we show the optimization curves, i.e., the obtained fitness value against the number of fitness evaluations, of the proposed methods and traditional methods on the CEC 2018 test suite. The curves obtained by PG-DE and SaDE on F_8, F_{11} in 10D and F_8, F_{15} in 30D, and PG-MPEDE and MPEDE on F_5, F_{11} in 10D and F_{10}, F_{17} in 30D averaged over five runs are shown in Figure 4. In the figure, the x-axis is the ratio between the number of fitness evaluations consumed so far (i.e., FEs) and the maximum number of fitness evaluations (i.e., maxNFEs), while the y-axis represents the function value obtained at the ratio. The sub-figures in the first row show the optimization curves of PG-DE and SaDE, while the sub-figures in the second row show those of PG-MPEDE and MPEDE. The names of sub-figures are defined by the rule "Algorithm-Dimension-Function index". For example, PG-DE (SaDE)-10D-F_8 represents the obtained optimization curves of F_8 in 10D optimized by PG-DE and SaDE.

Table 4 The CPU time costs of MPEDE and PG-MPEDE

Method	10D	30D
MPEDE	**17.44 s**	**217.24 s**
PG-MPEDE	18.36 s	247.72 s

Table 5 Results obtained by PG-DE without SL and PG-DE with SL on 10D test functions over 51 runs

	10D			
	PG-DE without SL		PG-DE with SL	
	Mean	std	Mean	std
F_1	**0.00**E+**00**$^{\approx}$	0.00E+00	**0.00**E+**00**	0.00E+00
F_3	**0.00**E+**00**$^{\approx}$	0.00E+00	**0.00**E+**00**	0.00E+00
F_4	**3.51**E+**00**†	1.28E+00	3.97E+00	1.45E+00
F_5	9.03E+00§	6.48E+00	**4.38**E+**00**	3.74E+00
F_6	**0.00**E+**00**$^{\approx}$	4.97E−08	**0.00**E+**00**	4.97E−08
F_7	2.45E+01§	2.55E+00	**2.25**E+**01**	2.84E+00
F_8	8.23E+00§	5.49E+00	**5.23**E+**00**	5.06E+00
F_9	**0.00**E+**00**$^{\approx}$	0.00E+00	**0.00**E+**00**	0.00E+00
F_{10}	3.51E+02§	3.69E+02	**1.99**E+**02**	2.93E+02
BM	5		**8**	
$\dagger/\approx/\S$			1/4/4	

From the figures, we see that on most problems, PG-DE can obtain better function values than SaDE, while PG-MPEDE can obtain better function values than MPEDE. Furthermore, it may be concluded that PG-DE and PG-MPEDE converge faster than their counterparts.

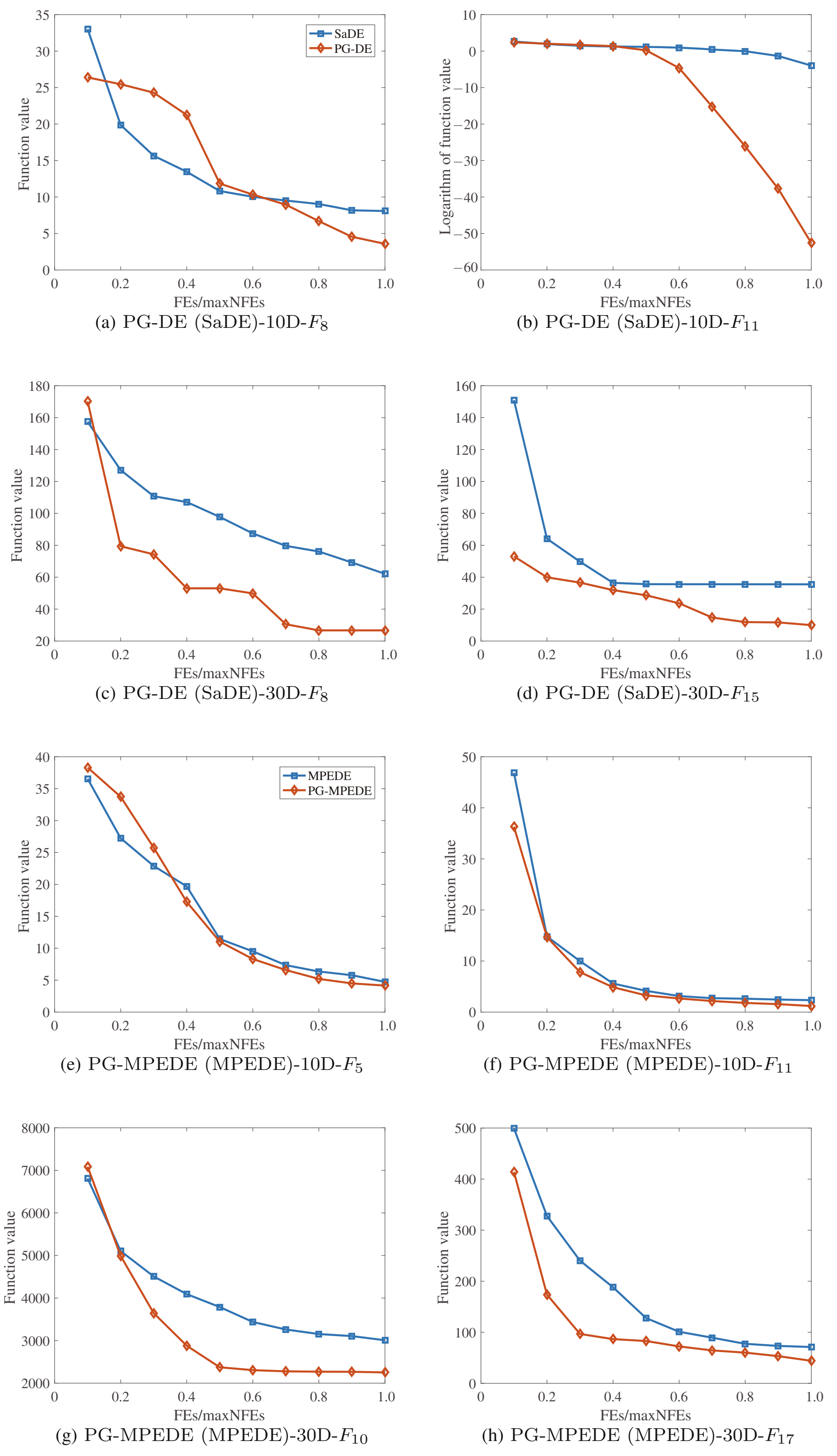
5.8 Visualizing the selection probabilities of the operators

In this subsection, the selection probability calculated by the trained deep neural networks for each operator is shown against the generations. We test PG-DE on F_1, F_{11}, F_{24} in 10D, on F_8 in 30D, and test PG-MPEDE on F_8, F_{11} in 10D. The selection probability of each operator at each generation is shown in Figure 5. (a)–(d) in Figure 5 show the selection probabilities of the four operators, including rand/1, current-to-rand/1, rand-to-best/2 and current-to-best/1, in PG-DE, while (e)–(f) in Figure 5 show the selection probabilities of the other three operators including rand/1, current-to-rand/1 and current-to-pbest/1, in PG-MPEDE. The name of the sub-figure is defined by the rule "Algorithm-Dimension-Function index". For example, PG-DE-10D-F_1 represents the operator selection probability of PG-DE on the optimizing function F_1 in 10D.

From the figures, we see that the predicted selection probabilities of the operators by the trained network are much different for different problems. Furthermore, it is observed that the selection probability varies at different generations, which implies that it is difficult to design the mechanism for selecting operators manually. The experiment shows that by employing deep neural networks, a complex mechanism for operator selection can be learned, which can successfully improve the performance of DEs.

6 Conclusion

In this paper, we develop a learning-based adaptive operator selection mechanism. A deep neural network is used to extract relevant features during the search process and a Dirichlet policy is learned to determine the probability of operator selection. Furthermore, a scale parameter is introduced to control the variance of the Dirichlet distribution. A reinforcement learning method taking the Dirichlet distribution as policy is proposed to train the deep neural network. Furthermore, expert knowledge is learned by a supervised learning phase, which has been verified to be capable of improving the performance of the learned neural network. After training, we see that the obtained algorithms are tested on the CEC 2018 test suite. In comparison with well-developed EAs, we find that it is competitive and has gained the best average performance. The experimental results show that the obtained algorithms (PG-DE and PG-MPEDE) can perform better than their counterparts. Theoretical analysis show that the proposed algorithms (i.e., PG-DE and PG-MPEDE) are identical to classical DEs (i.e., SaDE and MPEDE) asymptotically.



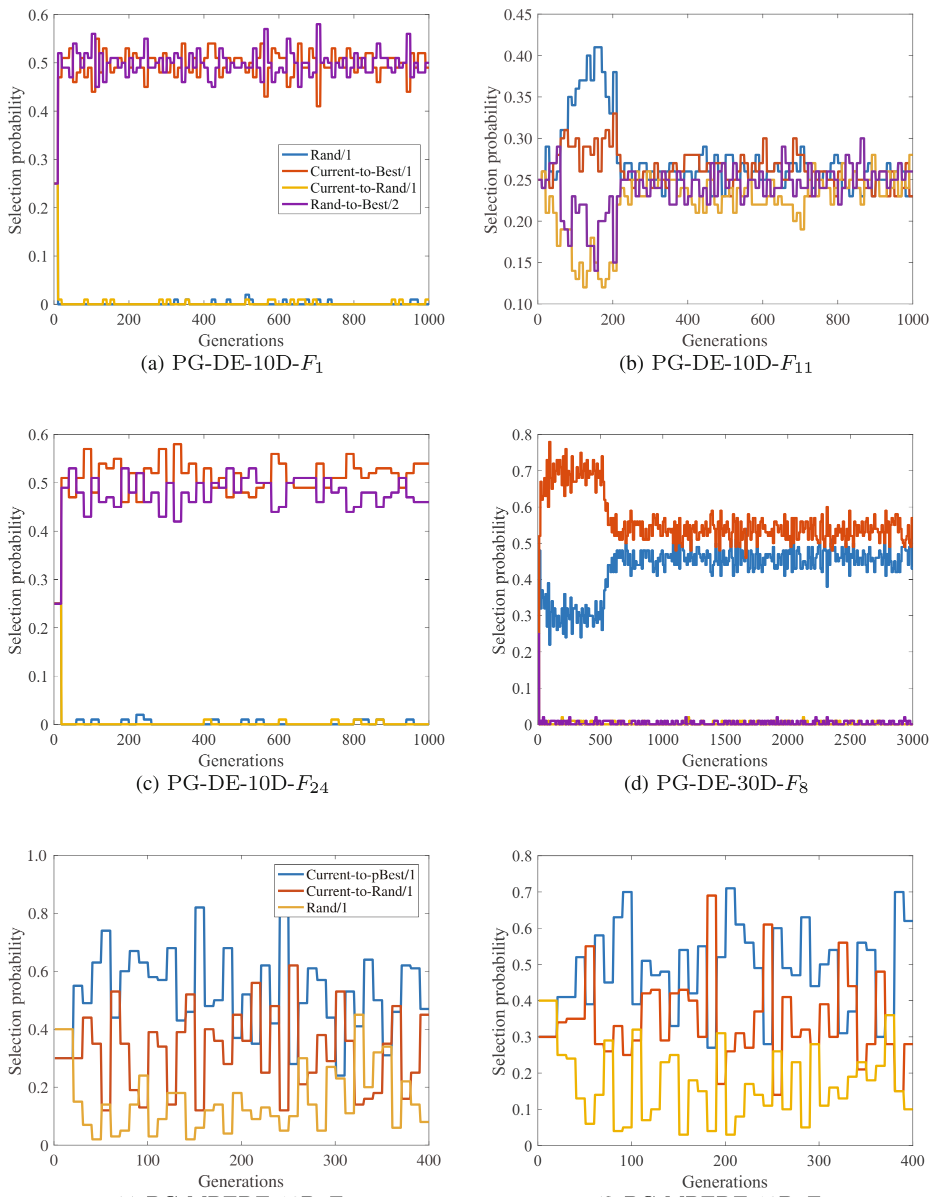
(a) PG-DE (SaDE)-10D-F_8 (b) PG-DE (SaDE)-10D-F_{11}

(c) PG-DE (SaDE)-30D-F_8 (d) PG-DE (SaDE)-30D-F_{15}

(e) PG-MPEDE (MPEDE)-10D-F_5 (f) PG-MPEDE (MPEDE)-10D-F_{11}

(g) PG-MPEDE (MPEDE)-30D-F_{10} (h) PG-MPEDE (MPEDE)-30D-F_{17}

Figure 4 (Color online) The optimization curves obtained by PG-DE (SaDE) (shown in (a)–(d)) and PG-MPEDE (MPEDE) (shown in (e)–(h))



(a) PG-DE-10D-F_1 (b) PG-DE-10D-F_{11}

(c) PG-DE-10D-F_{24} (d) PG-DE-30D-F_8

(e) PG-MPEDE-10D-F_8 (f) PG-MPEDE-10D-F_{11}

Figure 5 (Color online) The operator selection probability of PG-DE and PG-MPEDE

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 62076197) and Key Research and Development Project of Shaanxi Province (Grant No. 2022GXLH-01-15).

References

1 Andrychowicz M, Denil M, Gomez S, et al. Learning to learn by gradient descent by gradient descent. In: Advances in Neural Information Processing Systems, vol. 29. La Jolla: NIPS, 2016, 3981–3989

2 Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940, 2016

3 Chen Y T, Hoffman M W, Colmenarejo S G, et al. Learning to learn without gradient descent by gradient descent. In: International Conference on Machine Learning, vol. 70. San Diego: JMLR, 2017, 748–756

4 Da Silva E K, Barbosa H J C, Lemonge A C C. An adaptive constraint handling technique for differential evolution with dynamic use of variants in engineering optimization. Optim Eng, 2011, 12: 31–54

5 Dai H, Zhang Y, Dilkina B, et al. Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems, vol. 30. La Jolla: NIPS, 2017, 6348–6358

6 Das S, Suganthan P N. Differential evolution: A survey of the state-of-the-art. IEEE Trans Evol Comput, 2011, 15: 4–31

7 Defferrard M, Bresson X, Vandergheynst P. Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems, vol. 29. La Jolla: NIPS, 2016, 3844–3852

8 Eberhart R, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science. Piscataway: IEEE, 1995, 39–43

9 Elsayed S, Sarker R, Coello C C, et al. Adaptation of operators and continuous control parameters in differential evolution for constrained optimization. Soft Comput, 2018, 22: 6595–6616

10 Frazier P I. A tutorial on Bayesian optimization. arXiv:1807.02811, 2018

11 Galletly J. Evolutionary algorithms in theory and practice. Complexity, 1996, 2(8): 26–27

12 Gasse M, Chetelat D, Ferroni N, et al. Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 32. La Jolla: NIPS, 2019, 15580–15592

13 Guo T D, Han C Y, Tang S Q, et al. Solving Combinatorial Problems with Machine Learning Methods. Berlin: Springer, 2019

14 Hansen N, Muller S, Koumoutsakos P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol Comput, 2003, 11: 1–18

15 Hansen N, Ostermeier A. Completely derandomized self adaptation in evolution strategies. Evol Comput, 2001, 9: 159–195

16 Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput, 1997, 9: 1735–1780

17 Huang C W, Yuan B, Li Y X, et al. Automatic parameter tuning using Bayesian optimization method. In: Proceedings of the IEEE Congress on Evolutionary Computation. New York: IEEE, 2019, 2090–2097

18 Hutter F, Hoos H H, Leytonbrown K. Sequential model-based optimization for general algorithm configuration. In: Proceedings of International Conference on Learning and Intelligent Optimization. Lecture Notes in Computer Science. Cham: Springer, 2011, 507–523

19 Kingma D P, Ba J. Adam: A method for stochastic optimization. In: International Conference on Learning Representations. New York: ICLR, 2015, 1–15

20 Li K, Malik J. Learning to optimize. arXiv:1606.01885, 2016

21 Li Z W, Chen Q F, Koltun V. Combinatorial optimization with graph convolutional networks and guided tree search. In: Advances in Neural Information Processing Systems, vol. 31. La Jolla: NIPS, 2018, 537–546

22 Liu X, Sun J Y, Zhang Q F, et al. Learning to learn evolutionary algorithm: A learnable differential evolution. IEEE Trans Emerg Top Comput Intell, 2023, 7: 1605–1620

23 Mallipeddi R, Suganthan P N, Pan Q K, et al. Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput, 2011, 11: 1679–1696

24 Mazyavkina N, Sviridov S, Ivanov S, et al. Reinforcement learning for combinatorial optimization: A survey. Comput Oper Res, 2021, 134: 105400

25 Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv:1312.5602, 2013

26 Price K V. An Introduction to Differential Evolution. New York: McGraw-Hill, 1999

27 Puterman M L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York: John Wiley & Sons, 1994

28 Qin A K, Huang V L, Suganthan P N. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput, 2009, 13: 398–417

29 Rijn S V, Doerr C, Bäck T. Towards an adaptive CMA-ES configurator. In: Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 11101. Cham: Springer, 2018, 54–65

30 Sallam K M, Elsayed S M, Sarker R A, et al. Landscape-based adaptive operator selection mechanism for differential evolution. Inform Sci, 2017, 418: 383–404

31 Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search. Nature, 2016, 529: 484–489

32 Sun J Y, Liu X, Bäck T, et al. Learning adaptive differential evolution algorithm from optimization experiences by policy gradient. IEEE Trans Evol Comput, 2021, 25: 666–680

33 Sun J Y, Zhang Q F, Tsang E. DE/EDA: A new evolutionary algorithm for global optimization. Inform Sci, 2005, 169: 249–262

34 Sutton R, Barto A. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 1998

35 Tanabe R, Fukunaga A. Reviewing and benchmarking parameter control methods in differential evolution. IEEE Trans Syst Man Cybernet, 2020, 50: 1170–1184

36 Tvrdik J. Competitive differential evolution. Mendel, 2006, 5: 7–12

37 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of the 31st International Conference

on Neural Information Processing Systems. Advances in Neural Information Processing Systems, vol. 30. La Jolla: NIPS, 2017, 6000–6010

38 Wang S P, Sun J, Xu Z B. Hyperadam: A learnable task-adaptive Adam for network training. In: The National Conference on Artificial Intelligence. Palo Alto: AAAI, 2019, 5297–5304

39 Wang Y, Cai Z X, Zhang Q F. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans Evol Comput, 2011, 15: 55–66

40 Wu G H, Mallipeddi R, Suganthan P N, et al. Differential evolution with multi-population based ensemble of mutation strategies. Inform Sci, 2016, 329: 329–345

41 Xin L, Song W, Cao Z G, et al. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35. Palo Alto: AAAI, 2021, 12042–12049

42 Zamuda A, Brest J. Population reduction differential evolution with multiple mutation strategies in real world industry challenges. Swarm Evol Comput, 2012, 7269: 154–161

43 Zhan Z, Zhang J, Li Y, et al. Adaptive particle swarm optimization. Syst Man Cybernet, 2009, 39: 1362–1381

44 Zhang G, Shi Y H. Hybrid sampling evolution strategy for solving single objective bound constrained problems. In: Proceedings of the IEEE Congress on Evolutionary Computation. New York: IEEE, 2018, 765–771

45 Zhang H T, Sun J Y, Bäck T, et al. Controlling sequential hybrid evolutionary algorithm by Q-learning. IEEE Comput Intell Mag, 2023, 18: 84–103

46 Zhang H T, Sun J Y, Tan K C, et al. Learning adaptive differential evolution by natural evolution strategies. IEEE Transactions Emerg Top Comput Intell, 2023, 7: 872–886

47 Zhang H T, Sun J Y, Xu Z B. Adaptive structural hyper-parameter configuration by Q-learning. In: Proceedings of the IEEE Congress on Evolutionary Computation. New York: IEEE, 2020, 1–8

48 Zhang H T, Sun J Y, Xu Z B. Learning to mutate for differential evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation. New York: IEEE, 2021, 1–8

49 Zhang H T, Sun J Y, Xu Z B, et al. Learning unified mutation operator for differential evolution by natural evolution strategies. Inform Sci, 2023, 632: 594–616

50 Zhang J Q, Sanderson A C. JADE: Adaptive differential evolution with optimal external archive. IEEE Trans Evol Comput, 2009, 13: 945–458

51 Zhao J J, Wang Z R, Yang F C. Genetic prompt search via exploiting language model probabilities. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. Macao: IJCAI, 2023, 5296–5305

Appendix A SaDE

SaDE [28] is the first of several adaptive DEs, which use online information to guide the selection of mutation operators. The core idea of SaDE is recording the number of successful operators in the previous several generations and is using the information to select an operator for the next generation. We denote by n_k the number of the k-th operator that has been used in previous L generations, and s_k is the number of the k-th operator succeeding in improving the function value. In the t-th generation, SaDE first calculates the success rate for each operator,

$$S_k = \frac{s_k}{n_k} + \varepsilon,$$

and then the probability

$$p_k = \frac{S_k}{\sum_k S_k} \tag{A.1}$$

is calculated for selecting the k-th operator. SaDE is summarized in Algorithm 6. In Line 8, the selection probability p_t^k is calculated by (A.1). In Line 15, the operator is assigned to each individual according to the probability p_t^k. In Line 16, the scaling factor F is generated from a fixed Gaussian distribution. Binomial crossover is used in SaDE. For the crossover rate, in Line 20, CR_{m_k} records the median of memory of CR (i.e., $\mathrm{CRMemory}_k$) for the k-th operator. $\mathrm{CRMemory}_k$ records the CR values, which can succeed in improving the function value in the previous L generations (for the first L generation, CR_{m_k} is initialized as 0.5). In Line 25, the crossover rates are generated from a Gaussian distribution whose mean is CR_{m_k}. Then the offspring $U^t = \{\boldsymbol{u}_i^t\}_{i=1}^N$ is generated by the selected operator and parameters in Line 28. Finally, better individuals can survive.

Algorithm 6 SaDE

```
Require: An objective function $f$, the maximum number of generations $T$, the learning period $L$, the candidate operators
    $\{\mathrm{OP}_k\}_{k=1}^4$ and population size $N$;
Ensure: $X^{\mathrm{T}}, f^*$;
 1: Initialize $X^0 \in \mathbb{R}^{\tilde{d}*N} = \{x_i^0 \in \mathbb{R}^{\tilde{d}}\}_{i=1}^N$ randomly; Set $t \leftarrow 0$;
 2: Initialize $\mathrm{CR}_{m_k}$, $k = 1, \dots, 4$;
 3: $f^* \leftarrow \min\{f(\boldsymbol{x}_i^0) \mid i = 1, \dots, N\}$;
 4: while $t < T$ do
 5:     $s_t^k \leftarrow 0$, $k = 1, \dots, 4$;
 6:     % Calculate probability $p_{t,k}$
 7:     if $t \geqslant L$ then
 8:         Calculate $p_t^k$ by (A.1), $k = 1, \dots, 4$;
 9:     else
10:         Initialize selection probability $p_t^k$ for the $k$-th operator, $k = 1, \dots, 4$;
11:         $n_t^k \leftarrow N \times p_t^k$, $k = 1, \dots, 4$;
12:     end if
13:     % Assign mutation operator and parameter for each individual
14:     for $i = 1 \to N$ do
15:         Assign mutation operator according to $p_t^k$ for $\boldsymbol{x}_i^t$;
16:         $F_i \leftarrow \mathcal{N}(0.5, 0.3)$;
17:     end for
18:     if $t \geqslant L$ then
19:         for $k = 1 \to 4$ do
20:             $\mathrm{CR}_{m_k} \leftarrow \mathrm{median}(\mathrm{CRMemory}_k)$;
21:         end for
22:     end if
23:     for $i = 1 \to N$ do
24:         if $\boldsymbol{x}_i^t$ is assigned with the $k$-th operator then
25:             $\mathrm{CR}_i \leftarrow \mathcal{N}(\mathrm{CR}_{m_k}, 0.1)$;
26:         end if
27:     end for
28:     Generate $U^t$ by selected mutation operator and crossover operator;
29:     for $i = 1 \to N$ do
30:         if $f(\boldsymbol{x}_i^t) < f(\boldsymbol{u}_i^t)$ then
31:             $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{x}_i^t$;
32:         else
33:             $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{u}_i^t$;
34:             if $\boldsymbol{u}_i^t$ is generated by the $k$-th operator then
35:                 $s_t^k \leftarrow s_t^k + 1$;
36:                 Store $\mathrm{CR}_i$ into $\mathrm{CRMemory}_k$;
37:             end if
38:         end if
39:     end for
40:     $t \leftarrow t + 1$;
41:     $f^* \leftarrow \min\{f(\boldsymbol{x}_i^t) \mid i = 1, \dots, N\}$;
42: end while
```

Appendix B MPEDE

MPEDE [40] is one of the state-of-the-art DEs with AOS. In MPEDE, both the AOS and the APC mechanisms are used. The AOS implemented in MPEDE can be described as follows. The whole population is divided into four sub-populations. The first three sub-populations are assigned to three different mutation operators, and the last sub-population is regarded as the "reward sub-population", which will be assigned to the mutation operator that has the best performance in the previous several generations. The performance of each operator is measured as follows. Denote by n_k the number of the k-th operator that has been used in previous L generations and by Δf_k the function improvement of the k-th operator. Then the "reward sub-population" will be assigned to the operator with the maximum value $\Delta f_k/n_k$.

The APC mechanism is similar to that in JADE [50]:

$$\mu_F = (1-c)\cdot\mu_F + c\cdot\mathrm{mean}_L(S_F), \quad \mu_{\mathrm{CR}} = (1-c)\cdot\mu_{\mathrm{CR}} + c\cdot\mathrm{mean}_A(S_{\mathrm{CR}}),$$

$$F \sim \mathcal{C}(\mu_F, 0.1), \quad \mathrm{CR} \sim \mathcal{N}(\mu_{\mathrm{CR}}, 0.1),$$

where S_F and S_{CR} are the vectors recording successful parameters, and mean_L and mean_A are the Lehmer mean operator and the average mean operator, respectively.

MPEDE is summarized in Algorithm 7. In Lines 10 and 11, the parameters are generated for each sub-population and offsprings are generated in Line 14. Binomial crossover is used in MPEDE. From Line 19 to Line 23, the selection process is implemented, in the meantime, the success parameters and the improvement of the function value are updated. AOS mechanism is then implemented in Line 28, and for every L generation, the "best" operator is regarded as the operator with the maximum average function value improvement. In Line 31, the last sub-population is assigned to the "best" operator. Finally, in Lines 32 and 33, μ_F and μ_{CR} are updated.

Algorithm 7 MPEDE

```
Require: An objective function $f$, the maximum number of generations $T$, the learning period $L$, the candidate operators
    $\{\mathrm{OP}_k\}_{k=1}^3$ and population size $N$;
Ensure: $X^{\mathrm{T}}, f^*$;
 1: Initialize $X^0 \in \mathbb{R}^{\tilde{d}*N} = \{x_i^0 \in \mathbb{R}^{\tilde{d}}\}_{i=1}^N$ randomly; Set $t \leftarrow 0$;
 2: $f^* \leftarrow \min\{f(\boldsymbol{x}_i^t) \mid i = 1, \ldots, N\}$;
 3: Initialize $\lambda_k$ and $N_k \leftarrow N \cdot \lambda_k$, $k = 1, \ldots, 4$;
 4: Randomly divide the $X^0$ into $X_k^0$, $k = 1, \ldots, 4$ according to their sizes $N_k$;
 5: Randomly select a sub-population $X_k^0 \leftarrow X_k^0 \cup X_4^0, k = 1, 2, 3$ and $N_k \leftarrow N_k + N_4$;
 6: $\mu_{\mathrm{CR}_k} \leftarrow 0.5$, $\mu_{F_k} \leftarrow 0.5$, $\Delta f_k \leftarrow 0$, $k = 1, \ldots, 3$;
 7: while $t < T$ do
 8:    for $k = 1 \to 3$ do
 9:       for $i = 1 \to N_k$ do
10:          $F_{i,k} \leftarrow \mathcal{C}(\mu_{F_k}, 0.1)$;
11:          $\mathrm{CR}_{i,k} \leftarrow \mathcal{N}(\mu_{\mathrm{CR}_k}, 0.1)$;
12:       end for
13:       $S_{F,k} \leftarrow \emptyset$, $S_{\mathrm{CR},k} \leftarrow \emptyset$;
14:       Obtain $U_k^t$ by Mutation and Crossover on $X_k^t$;
15:    end for
16:    for $k = 1 \to 3$ do
17:       for $i = 1 \to NP_k$ do
18:          if $f(\boldsymbol{x}_{i,k}^t) < f(\boldsymbol{u}_{i,k}^t)$ then
19:             $\boldsymbol{x}_{i,k}^{t+1} \leftarrow \boldsymbol{x}_{i,k}^t$;
20:          else
21:             $\boldsymbol{x}_{i,k}^{t+1} \leftarrow \boldsymbol{u}_{i,k}^t$;
22:             $\Delta f_k \leftarrow \Delta f_k + f(\boldsymbol{x}_{i,k}^t) - f(\boldsymbol{u}_{i,k}^t)$;
23:             Store $\mathrm{CR}_{i,k}$ into $S_{\mathrm{CR},k}$, store $F_{i,k}$ into $S_{F,k}$;
24:          end if
25:       end for
26:    end for
27:    if $\mathrm{mod}(t, L) == 0$ then
28:       $j \leftarrow \mathrm{argmax}_k\{\frac{\Delta f_k}{L \cdot N_k} \mid k = 1, 2, 3\}$;
29:       $\Delta f_k \leftarrow 0$, $k = 1, 2, 3$;
30:    end if
31:    $X_j^t \leftarrow X_j^t \cup X_4^t$ and $N_j \leftarrow N_j + N_4$;
32:    $\mu_{F_k} \leftarrow (1 - c)\mu_{F_k} + c \cdot \mathrm{mean}L(S_{F,k})$, $k = 1, 2, 3$;
33:    $\mu_{\mathrm{CR}_k} \leftarrow (1 - c)\mu_{\mathrm{CR}_k} + c \cdot \mathrm{mean}(S_{\mathrm{CR},k})$, $k = 1, 2, 3$;
34:    $f^* \leftarrow \min\{f(\boldsymbol{x}_i^t) \mid i = 1, \ldots, N\}$;
35:    $t \leftarrow t + 1$;
36: end while
```

Appendix C Proof of Theorem 4.1

Proof of Theorem 4.1. In the proof, we use $\|\cdot\|$ to represent the L_2 norm $\|\cdot\|_2$. We denote by X_i the i-th component of X, and by $\mu(X_i)$ and $\mathrm{var}(X_i)$ the mean and variance of X_i, respectively. It holds that

$$\mathrm{P}(\|X - \mu(X)\| > \delta) = \int_{\|X - \mu(X)\| > \delta} p(X) dX$$

$$
\begin{aligned}
&\leqslant \int_{\|X-\mu(X)\|>\delta} \frac{\|X-\mu(X)\|^2}{\delta^2} p(X)dX \\
&= \int_{\|X-\mu(X)\|>\delta} \frac{\sum_i (X_i-\mu(X_i))^2}{\delta^2} p(X)dX \\
&\leqslant \int_{\mathbb{R}^d} \frac{\sum_i (X_i-\mu(X_i))^2}{\delta^2} p(X)dX \\
&= \sum_i^d \int_{\mathbb{R}^1\times\cdots\times\mathbb{R}^1} \frac{(X_i-\mu(X_i))^2}{\delta^2} p(X_1,\ldots,X_d)dX_1\cdots dX_d \\
&= \frac{\sum_i \operatorname{var}(X_i)}{\delta^2}.
\end{aligned}
$$

Let α_0 denote the $\sum_i \alpha_i$. As known, the variance $\operatorname{var}(X_i)$ of Dirichlet distribution with parameters $M\alpha_1+1,\ldots,M\alpha_d+1$ is $\frac{(M\alpha_i+1)(M\alpha_0-M\alpha_i+d-1)}{(M\alpha_0+d)^2(M\alpha_0+1+d)}$. Thus when $M\to+\infty$, we have $\frac{\sum_i \operatorname{var}(X_i)}{\delta^2}\to 0$. Meanwhile, $\mu(X)$ of Dirichlet distribution with parameters $M\alpha_1,\ldots,M\alpha_d$ is $[\frac{M\alpha_1+1}{M\alpha_0+d},\ldots,\frac{M\alpha_d+1}{M\alpha_0+d}]$. The mode is $m(X)=[\frac{M\alpha_1}{M\alpha_0},\ldots,\frac{n\alpha_d}{M\alpha_0}]=[\frac{\alpha_1}{\alpha_0},\ldots,\frac{\alpha_d}{\alpha_0}]$, i.e., when $M\to+\infty$, $m(X)\to\mu(X)$. We define the space $\mathcal{S}=\{X\|\|X-\mu(X)\|+\|m(X)-\mu(X)\|>\delta\}$. It holds that

$$
\begin{aligned}
&\mathrm{P}(\|X-m(X)\|>\delta) \\
&\quad\leqslant \mathrm{P}(\|X-\mu(X)\|+\|m(X)-\mu(X)\|>\delta) \\
&\quad= \int_{\mathcal{S}} p(X)dX \\
&\quad\leqslant \int_{\mathcal{S}} \frac{\|X-\mu(X)\|^2+\|m(X)-\mu(X)\|^2}{\delta^2} p(X)dX + \int_{\mathcal{S}} \frac{2\|X-\mu(X)\|\cdot\|m(X)-\mu(X)\|}{\delta^2} p(X)dX \\
&\quad\leqslant \int_{\mathbb{R}^d} \frac{\|X-\mu(X)\|^2+\|m(X)-\mu(X)\|^2}{\delta^2} p(X)dX + \int_{\mathbb{R}^d} \frac{2\|X-\mu(X)\|\cdot\|m(X)-\mu(X)\|}{\delta^2} p(X)dX \\
&\quad= \frac{\sum_i \operatorname{var}(X_i)}{\delta^2} + \int_{\mathbb{R}^d} \frac{\|m(X)-\mu(X)\|^2}{\delta^2} p(X)dX + \int_{\mathbb{R}^d} \frac{2\|X-\mu(X)\|\cdot\|m(X)-\mu(X)\|}{\delta^2} p(X)dX.
\end{aligned}
$$

Since when $M\to+\infty$, $m(X)\to\mu(X)$, we have

$$\lim_{M\to+\infty} \mathrm{P}(\|X-m(X)\|>\delta)\leqslant 0.$$

As $\mathrm{P}(\|X-m(X)\|>\delta)$ is a probability, i.e., $\mathrm{P}(\|X-m(X)\|>\delta)\geqslant 0$, then we accomplish the proof. $\square$