Learning adaptive differential evolution algorithm from optimization experiences by policy gradient

Sun, J.; Liu X.; Bäck, T.H.W.; Xu, Z.

Note: To cite this publication please use the final published version (if applicable).

Learning Adaptive Differential Evolution Algorithm From Optimization Experiences by Policy Gradient

Jianyong Sun[ID], *Senior Member, IEEE*, Xin Liu[ID], Thomas Bäck[ID], *Senior Member, IEEE*, and Zongben Xu

Abstract—Differential evolution is one of the most prestigious population-based stochastic optimization algorithm for black-box problems. The performance of a differential evolution algorithm depends highly on its mutation and crossover strategy and associated control parameters. However, the determination process for the most suitable parameter setting is troublesome and time consuming. Adaptive control parameter methods that can adapt to problem landscape and optimization environment are more preferable than fixed parameter settings. This article proposes a novel adaptive parameter control approach based on learning from the optimization experiences over a set of problems. In the approach, the parameter control is modeled as a finite-horizon Markov decision process. A reinforcement learning algorithm, named policy gradient, is applied to learn an agent (i.e., parameter controller) that can provide the control parameters of a proposed differential evolution adaptively during the search procedure. The differential evolution algorithm based on the learned agent is compared against nine well-known evolutionary algorithms on the CEC'13 and CEC'17 test suites. Experimental results show that the proposed algorithm performs competitively against these compared algorithms on the test suites.

Index Terms—Adaptive differential evolution, deep learning, global optimization, policy gradient (PG), reinforcement learning (RL).

I. Introduction

AMONG many evolutionary algorithm (EA) variants, differential evolution (DE) [1], [2] is one of the most prestigious due to its exclusive advantages, such as automatic adaptation, easy implementation, and very few control parameters [3], [4]. The DE variants have been successfully applied to a variety of real-world optimization problems [3], [5] and have been considered very competitive in the evolutionary

computation community according to their performances on various competitions [6]–[9].

However, DE has also some drawbacks, such as stagnation, premature convergence, sensitivity/insensitivity to control parameters, and others [5]. Although various factors, such as dimensionality of the decision space and the characteristics of the optimization problems, can result in these drawbacks, the bad choice of control parameters (namely, the scale factor F, the crossover rate CR, and the population size N) is one of the key problems [10], [11].

It is well acknowledged that control parameters can significantly influence the performance of an EA [10], and this also holds for the control parameters of DE. In the early days of research in DE, the control parameters are usually set by trial and error [12]–[14] according to the *optimization experiences* gained from applying them to a set of test problems. Once the control parameters are set, they are fixed along the search procedure (this parameter determination approach is usually referred to as "parameter tuning"). For example, in [15], F and CR are suggested to be in the range of [0.4, 1] and [0.5, 0.7], respectively, while N is suggested to be $[2-40]n$ where n is the problem dimension [13]. The trial-and-error approach is usually time consuming, not reliable and inefficient [16].

Along with the study of the exploration–exploitation relation in EA, it is found that the optimal control parameter setting for a DE algorithm is problem specific, dependent on the state of the evolutionary search procedure, and on the different requirements of problems [17]. Furthermore, different control parameters impose different influences on the algorithmic performance in terms of effectiveness, efficiency, and robustness [18]. Therefore, it is generally a very difficult task to properly determine the optimal control parameters for a balanced algorithmic performance due to various factors, such as problem characteristics and correlations among them.

Some researchers claim that F and CR both affect the convergence and robustness, but F is more related to convergence [19], CR is more sensitive to the problem characteristics [20], and their optimal values correlate with N [21]. N could cause stagnation if it is too small, and slow convergence if it is too big [17]. Its setting strongly depends on problem characteristics (e.g., separability, multimodality, and others).

It is also observed that the control parameters should be differently set at different generations simply due to changing requirements (i.e., exploration versus exploitation) in different phases of the optimization run. For example, researchers generally believe that F should be set larger at the beginning of the search to encourage exploration and smaller in the end to

ensure convergence [22]. In light of this observation, parameter control methods, i.e., methods for changing the control parameter values dynamically during the run, have become one of the main streams in DE studies [23]. These methods are referred to as "parameter control."

In [24], the parameter control methods are classified as deterministic, adaptive, and self-adaptive, while a hybrid control is included in [23]. In this article, we propose to classify the parameter control methods based on how they are set according to online information collected during the evolutionary search. Our classification differentiates whether parameters are learned during search or not, and is provided next.

1) Parameter Control Without Learning: In this category, online information of any form is not reflected in the control of the parameters. Rather, a simple rule is applied deterministically or probabilistically along the search process.

The simple rule is constructed usually in three ways. First, no information is used at all. For example, in [25] and [26], F is sampled uniformly at random from a prefixed range at each generation for each individual. In [27], a combination of F and CR is randomly picked from three predefined pairs of F and CR at each generation for each individual.

Second, the simple rule is time varying depending on the generation. For example, in [25], F decreases linearly, while in [28], F and CR are determined based on a sinusoidal function of the generation index. In [6], the population size N is linearly decreased with respect to the number of fitness evaluations used.

Third, information collected in the *current generation*, such as the range of the fitness values, the diversity of the population, the distribution of individuals, and the rank of individuals, is used to specify the simple rule. For example, in [29], the minimum and maximum fitness values at current population are used to determine the value of F at each generation. In [30], F and CR are sampled based on the diversity of the objective values of the current population, while the average of the current population in the objective space is used in [31]. The individual's rank is used to determine F and CR for each individual in [32]. It is used to compute the mean values of the normal distributions associated with F and CR in [33] from which F and CR are sampled for each individual.

2) Parameter Control With Learning From the Search Process: In this category, collectable information during the search process is processed for updating the control parameters. The information used in these methods is mainly the successful trials obtained by using previous F and CR values.

There are mainly three ways. First, a trial F and CR is decided by an ϵ-greedy strategy, as initially developed in [34]. That is, if a uniformly sampled value is less than a hyperparameter ϵ, a random number in [0, 1] is uniformly sampled as the trial F (resp. CR); otherwise, previous F (resp. CR) is used. If the trial by using the F and CR is successful, the sampled F and CR will be passed to the next generation; otherwise, the previous F and CR will be kept. The sampled F value is taken as a perturbation in [35]. The hyperparameter ϵ is adaptively determined either by the current population diversity [30] or by fitness values [31].

Second, successful control parameters are stored in memory (or pool) during the optimization process which are then used to create the next parameters [36], [37]. For example, the median (or mean) value of the memory values is used as the mean of the control parameter distribution for sampling new control parameters [36]. The distribution is assumed to be normal [36] or Cauchy [38]. To make the sampling adapt to the search state, hyperparameters are proposed in the distribution [14], [38] while the hyperparameters are updated at each generation according to the success of previous distributions.

Third, some authors proposed to update the mean of the control parameter's distribution through a convex linear combination between the arithmetic [37] or Lehmer [6], [39] mean of the stored pool of successful and the current control parameters. In [40], an ensemble of two sinusoidal waves is added to adapt F based on successful performance of previous generations. In [41], a semiparameter adaptation approach based on randomization and adaptation is developed to effectively adapt F values. In [42], the control parameters are also modified based on the ratio of the current number of function evaluations to the maximum allowed number of function evaluations. A grouping strategy with an adaptation scheme is proposed in [43] and [44] to tackle the improper updating method of CR. In [45] and [46], a memory update mechanism is further developed, while new control parameters are assumed to follow normal and Cauchy distribution, respectively. In [47], the control parameters are updated according to the formulas of particle swarm optimization, in which the control parameters are considered as particles and evolved along the evolution process.

In this article, we propose a novel approach to adaptively control F and CR. In our approach, the control parameters at each generation are the output of a nonlinear function which is modeled by a deep neural network (DNN). The DNN works as the parameter controller. The parameters of the network are learned from the experiences of optimizing a set of training functions by a proposed DE. The learning is based on the formalization of the evolutionary procedure as a finite-horizon Markov decision process (MDP). One of the reinforcement learning (RL) algorithms, policy gradient (PG) [48], is applied to optimize for the optimal parameters of the DNN.

This method can be considered as an automatic alternative to the trial-and-error approach in the early days of DE study. Note that the trial-and-error approach can only provide possible values or ranges of the control parameters. For a new test problem, these values need to be further adjusted which could result in spending a large amount of computational resources. Our approach does not need such extra adjustment for a new test problem. It can provide control parameter settings not only adaptively to the search procedure but also to test problems.

In the remainder of this article, we first introduce deep learning and RL which are the preliminaries for our approach in Section II. Section III presents the proposed learning method. Experimental results are presented in Sections IV and V including the comparison with several well-known DEs and a state-of-the-art EA. The related work is presented in Section VI. Section VII concludes this article.

II. Parameterized Knowledge Representation by Deep and Reinforcement Learning

A. Deep Learning

Deep learning is a class of machine learning algorithms for learning data representation [49]. It consists of multiple layers of nonlinear processing units for extraction of meaningful features. The deep learning architecture can be seen as a parameterized model for knowledge representation and a tool for knowledge extraction. It can also be seen as an efficient expert system. That is, given the current state of a system, through the DNN, a proper decision can be made if the deep network is optimally trained. The DNN has a high order of modeling freedom due to its large number of parameters, which make it able to make accurate predictions. Deep learning has had remarkable success in recent years in image processing, natural language processing and other application domains [49].

B. Reinforcement Learning

RL deals with the situation that an agent interacts with its surrounding environment. The aim of learning is to find an optimal policy for the agent that can respond to the environment for a maximized cumulative reward. RL can be modeled as a 4-tuple (s, a, r, and p_a) MDP, where s (resp. a, r, and p_a) represents the state (resp. action, reward, and transition probability).

Formally, at each time step t, a state s_t and reward r_t are associated to it. A policy π is a conditional probability distribution, i.e., $\pi = p(A_t|S_t; \theta)$ with parameter θ, where A_t (resp. S_t) represents the random variable for action (resp. state). Given the current state s_t, the agent takes an action a_t by sampling from π. Given this action, the environment responds with a new state s_{t+1} and a reward r_{t+1}. The expectation of total rewards $U(\theta) = \mathbb{E}_{\tau \sim q(\tau)}(\sum_{t=0}^{T} \gamma_t r_t)$, where γ_t denotes the time-step-dependent weighting factor, is to be maximized for an optimal policy π^* where the expectation is taken over trajectory $\tau = \{s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T, \ldots\}$ with the joint probability distribution $q(\tau)$.

To train an RL agent for the optimal parameter, various RL algorithms, such as temporal difference, Q-learning, SARSA, PG, and others have been widely used for different scenarios (e.g., discrete or continuous action and state space) [48]. If the policy is represented by a DNN, it leads to the so-called deep RL, which has gained incredible success on playing games, such as AlphaGo [50].

III. Adaptive Parameter Control via Policy Gradient

In this section, we show how to learn to control the adaptive settings of a typical DE. Before presenting the algorithm, the notations used in this article are listed in Table I.

A. Typical DE

In the proposed DE, the current-to-$p$best/1 mutation operator and the binomial crossover are employed. In the current-to-$p$best/1 mutation operator [37], at the t-th generation, for

TABLE I
Notations

N	the population size
n	dimension of the decision variable
f	the objective (fitness) function
$\mathbf{x}_i^t \in \mathbb{R}^n$	the i-th individual at the t-th generation
$\mathcal{P}^t \in \mathbb{R}^{N \times n}$	the t-th population
$\mathcal{F}^t \in \mathbb{R}^N$	the fitness values of the t-th population
$\mathcal{V}^t \in \mathbb{R}^{N \times n}$	the mutated population at the t-th generation
$\widetilde{\mathcal{P}}^t \in \mathbb{R}^{N \times n}$	the trial solutions obtained at the t-th generation
$\widetilde{\mathcal{F}}^t \in \mathbb{R}^N$	the fitness values of the trial solutions
$\mathcal{U}_t$	the statistics at the t-th generation
$\mathcal{H}_t$	the memory at the t-th generation
$\mathbf{M}$	the mutation operator
$\mathbf{CR}$	the crossover operator
$\mathbf{S}$	the selection operator

each individual $\mathbf{x}_i^t$, a mutated individual $\mathbf{v}_i^t$ is generated in the following manner:

$$\mathbf{v}_i^t = \mathbf{x}_i^t + F_i^t \cdot \left(\mathbf{x}_{\text{pbest}}^t - \mathbf{x}_i^t\right) + F_i^t \cdot \left(\mathbf{x}_{r_1}^t - \mathbf{x}_{r_2}^t\right) \quad (1)$$

where $\mathbf{x}_{\text{pbest}}^t$ is an individual randomly selected from the best $N \times p$ ($p \in (0, 1]$) individuals at generation t. The indices r_1 and r_2 are randomly selected from $[1, N]$ such that they differ from each other and i.

The binomial crossover operator works on the target individual $\mathbf{x}_i^t = (x_{i,1}^t, \ldots, x_{i,n}^t)$ and the corresponding mutated individual $\mathbf{v}_i^t = (v_{i,1}^t, \ldots, v_{i,n}^t)$ to obtain a trial individual $\widetilde{\mathbf{x}}_i^t = (\widetilde{x}_{i,1}^t, \ldots, \widetilde{x}_{i,n}^t)$ element by element as follows:

$$\widetilde{x}_{i,j}^t = \begin{cases} v_{i,j}^t, & \text{if } \text{rand}[0, 1] \leq \text{CR}_i^t \text{ or } j = j_{\text{rand}} \\ x_{i,j}^t, & \text{otherwise} \end{cases} \quad (2)$$

where $\text{rand}[0, 1]$ denotes a uniformly sampled number from $[0, 1]$ and j_{rand} is an integer randomly chosen from $[1, n]$.

Given the trial individuals, the next generation is selected individual by individual. For each individual, if $f(\widetilde{\mathbf{x}}_i^t) \leq f(\mathbf{x}_i^t)$, then $\mathbf{x}_i^{t+1} = \widetilde{\mathbf{x}}_i^t$; otherwise, $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t$.

It is clear that the control parameters of the proposed DE include N and $\{F_i^t, \text{CR}_i^t, 1 \leq i \leq N\}$ at each generation. In the following, we will show how $\{F_i^t, \text{CR}_i^t, 1 \leq i \leq N\}$ can be learned from optimization experiences.

B. Embed Recurrent Neural Network Within the Typical DE

The evolution procedure of the proposed DE can be formalized as follows. At generation t, a mutation population $\mathcal{V}^t = \{\mathbf{v}_1^t, \ldots, \mathbf{v}_N^t\}$ is first generated by applying the mutation operator (denoted as $\mathbf{M}$) on the current population $\mathcal{P}^t$; a trial population $\widetilde{\mathcal{P}}^t = \{\widetilde{\mathbf{x}}_1^t, \ldots, \widetilde{\mathbf{x}}_N^t\}$ is further obtained by applying the binomial crossover (denoted as $\mathbf{CR}$) operator. The new population is then formed by applying the selection (denoted as $\mathbf{S}$) operation. In the sequel, we denote $\Theta_F^t = \{F_i^t, 1 \leq i \leq N\}$ and $\Theta_{\text{CR}}^t = \{\text{CR}_i^t, 1 \leq i \leq N\}$.

Formally, the evolution procedure can be written as follows:

$$\begin{aligned} \mathcal{V}^t &= \mathbf{M}\left(\mathcal{P}^t; \Theta_F^t\right) \\ \widetilde{\mathcal{P}}^t &= \mathbf{CR}\left(\mathcal{V}^t, \mathcal{P}^t; \Theta_{\text{CR}}^t\right) \\ \widetilde{\mathcal{F}}^t &= f\left(\widetilde{\mathcal{P}}^t\right) \\ \mathcal{P}^{t+1}, \mathcal{F}^{t+1} &= \mathbf{S}\left(\mathcal{F}^t, \widetilde{\mathcal{F}}^t, \mathcal{P}^t, \widetilde{\mathcal{P}}^t\right). \end{aligned} \quad (3)$$
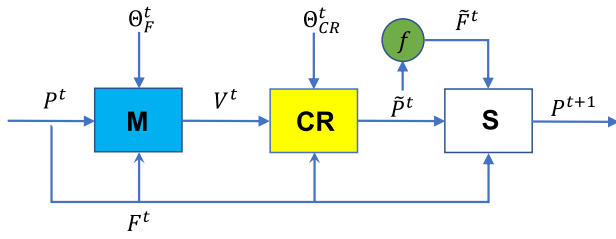
Fig. 1. Flowchart of a typical DE at the tth generation. In the figure, the mutation operator (resp. crossover and selection operator) is denoted as **M** (resp. **CR** and **S**).
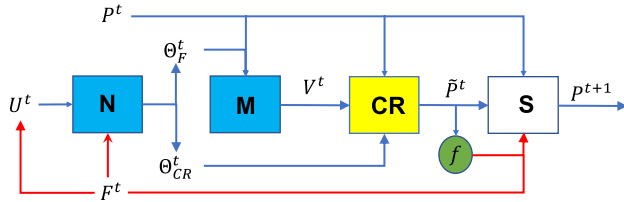

Fig. 2. Proposed DE with embedded neural network (denoted as **N**) at the tth generation.

Due to the stochastic nature of the mutation and crossover operators, the evolution procedure can be considered as a stochastic time series. The creation of solutions at generation t depends on information collected from previous generations from 1 to $t-1$. Fig. 1 shows the flowchart of the procedure at the tth generation.

As discussed in the introduction, recent studies focused on controlling F and CR by learning from online information. Various kinds of information are derived and used to update the control parameters for the current population. Generally speaking, the updated control parameters can be considered as output of a nonlinear function with the collected information as input. Since an artificial neural network (ANN) is a universal function approximator [51], this motivates us to take an ANN to approximate the nonlinear function.

The neural network can be embedded in the evolution procedure as a parameter controller. Fig. 2 shows the flowchart of the proposed DE with embedded neural network at the tth generation. As illustrated in the figure, the neural network (**N** in the figure) outputs Θ_F^t and Θ_{CR}^t. Θ_F^t is used as the input to the mutation operator, while Θ_{CR}^t is the input to the crossover operator. The mutated population (i.e., $\mathcal{V}^t$) and trial population (i.e., $\tilde{\mathcal{P}}^t$) are then generated, respectively. $\mathcal{V}^t$ and $\mathcal{P}^t$ are the input to **CR**; and $\tilde{\mathcal{P}}^t$ and $\mathcal{P}^t$ are the input to **S** for selecting the new population $\mathcal{P}^{t+1}$.

Existing research only utilizes the information collected from the current and/or previous generations for the parameter control. However, all the information until the current generation should have certain influences, although with different importances. The closer to the current generation, the more influential.

To accommodate the time-dependence feature, we take the neural network to be a long short-term memory (LSTM) [52]. LSTM is a kind of recurrent neural network. As its name suggests, LSTM is capable of capturing long-term dependencies among input signals. There are a variety of LSTM variants.
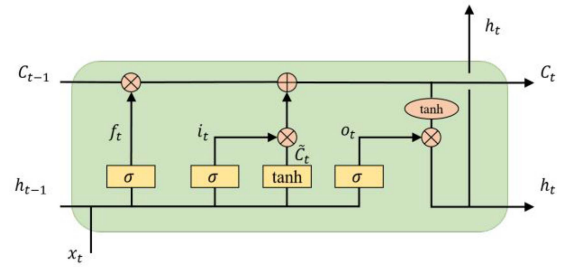
Fig. 3. Flowchart of the LSTM.

Its simplest form is formulated as follows:

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \\
C_t &= f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t \otimes \tanh(C_t)
\end{aligned}
$$

where x_t is the input at step t, $[h_{t-1}, x_t]$ means the catenation of h_{t-1} and x_t, $\otimes$ means the Hadamard product, and σ and tanh are the sigmoid activation function and tanh function, respectively

$$
\sigma(z) = \frac{1}{1 + e^{-z}}; \quad \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.
$$

The parameters of the LSTM include W_f, W_i, W_c, and W_o which are matrices and b_f, b_i, b_c, and b_o which are biases. Fig. 3 shows the flowchart of the LSTM.

Omitting the intermediate variables, the LSTM can be formally written as

$$
C_t, h_t = \text{LSTM}(x_t, h_{t-1}, C_{t-1}; \mathbf{W}) \tag{4}
$$

where $\mathbf{W} = [W_f, W_i, W_c, W_o, b_f, b_i, b_c, b_o]$ denotes its parameter.

In our context, we consider the input to the LSTM as the catenation of $\mathcal{F}^t$ and $\mathcal{U}^t$ which is some statistics derived from $\mathcal{F}^t$, and denote $\mathcal{A}^t = [\mathcal{F}^t, \mathcal{U}^t]$.[1] In addition, we use $\mathcal{H}^t$ and $\mathcal{C}^t$ to represent the short- and long-term memory. Formally, the parameter controller can be written as follows:

$$
\begin{aligned}
\mathcal{C}^t, \mathcal{H}^t &= \text{LSTM}\left(\mathcal{A}^t, \mathcal{H}^{t-1}, \mathcal{C}^{t-1}; \mathbf{W}_L\right) \\
\Theta_F^t &= \text{FullConnect}\left(\mathcal{H}^t; \mathbf{W}_F, b_F\right) \\
\Theta_{CR}^t &= \text{FullConnect}\left(\mathcal{H}^t; \mathbf{W}_C, b_C\right)
\end{aligned} \tag{5}
$$

where $\mathbf{W}_L$ is the parameter of LSTM, and FullConnect$(\cdot; \mathbf{W}, b)$ represents a fully connected neural network with weight matrix $\mathbf{W}$ and bias b. Here

$$
\begin{aligned}
\Theta_F^t &= \sigma\left(\mathcal{H}^{t\top} \mathbf{W}_F + b_F\right) \\
\Theta_{CR}^t &= \sigma\left(\mathcal{H}^{t\top} \mathbf{W}_C + b_C\right).
\end{aligned} \tag{6}
$$

[1]Here, $[\mathcal{F}^t, \mathcal{U}^t]$ also means to catenate the two vectors $\mathcal{F}^t$ and $\mathcal{U}^t$ into one single vector.

In the sequel, we denote $\Theta^t = [\Theta_F^t, \Theta_{CR}^t]$ and use the following concise formula to represent (5):

$$\Theta^t, \mathcal{C}^t, \mathcal{H}^t = \text{LSTM}\left(\mathcal{A}^t, \mathcal{H}^{t-1}, \mathcal{C}^{t-1}; \mathbf{W}\right) \qquad (7)$$

where $\mathbf{W} = [\mathbf{W}_L, \mathbf{W}_F, b_F, \mathbf{W}_C, b_C]$.

C. Model the Evolution Search Procedure as MDP

To learn the parameters of the LSTM, i.e., the agent or the controller, embedded in the DE, we first model the evolution procedure of the proposed DE as an MDP with the following definitions of environment, state, action, policy, reward, and transition.

1) Environment: For parameter control, an optimal controller is expected to be learned from optimization experiences obtained when optimizing a set of optimization problems. Therefore, the environment consists of a set of optimization problems (called training functions). They are used to evaluate the performance of the controller when learning. Note that these training functions should have some common characteristics for which can be learned for a good parameter controller.

2) State (S_t): We take the fitness $\mathcal{F}^t$, the statistics $\mathcal{U}^t$, and the memories $\mathcal{H}^t$ as the state S_t.

Particularly, $\mathcal{U}^t$ includes the histogram[2] of the normalized $\mathcal{F}^t$ (denoted as $\mathbf{h}_t$) and the moving average of the histogram vectors over the past g generations (denoted as $\bar{\mathbf{h}}_t$). Formally

$$\begin{aligned} \bar{f}_i &= \frac{f_i - \min\{\mathcal{F}^t\}}{\max\{\mathcal{F}^t\} - \min\{\mathcal{F}^t\}} \\ \mathbf{h}_t &= \text{histogram}\left(\{\bar{f}_i\}, b\right) \\ \bar{\mathbf{h}}_t &= \frac{1}{g}\sum_{i=t-g}^{t-1} \mathbf{h}_i \end{aligned} \qquad (8)$$

where b is the number of bins. The lower (resp. upper) range of the bins is defined as $\min\{\mathcal{F}_t\}$ (resp. $\max\{\mathcal{F}_t\}$). That is, to derive $\mathcal{U}^t$, the fitness values of the current population are first normalized. Its histogram is then computed and taken as the input to LSTM. This is to represent the information of the current population. Furthermore, the statistics represented by $\bar{\mathbf{h}}_t$ is computed as the information from previous search history.

It should be noted that the statistics $\mathcal{U}^t$ is computed at each generation with respect to (w.r.t.) the current population, not to each individual.

3) Action (A_t) and Policy (π): In the MDP, given state S_t, the agent can choose (sample) an action from policy π defined as a probability distribution $p(A_t|S_t; \theta)$, where θ represents the parameters of the policy. Here, we define A_t as the control parameters, i.e., $A_t = \{F_i^t, CR_i^t, 1 \leq i \leq N\} \in \mathbb{R}^{2N}$.

Since the control parameters take continuous values, we assume the policy is normal. That is

$$\begin{aligned} \pi(A_t|S_t) &= \mathcal{N}\left(A_t|\text{LSTM}(S_t), \sigma^2\right) \\ &= \frac{1}{(2\pi\sigma^2)^N} \exp\left\{-\frac{1}{2\sigma^2}(A_t - \text{LSTM}(S_t; \mathbf{W}))^2\right\}. \end{aligned} \qquad (9)$$

It is seen that the policy is uniquely determined by the LSTM parameter $\mathbf{W}$.

4) Reward (r_{t+1}): The environment responds with a reward r_{t+1} after the action. In our case, the reward r_{t+1} is defined as the relative improvement of the best fitness

$$r_{t+1} = \frac{\max\{\mathcal{F}^{t+1}\} - \max\{\mathcal{F}^t\}}{\max\{\mathcal{F}^t\}} \qquad (10)$$

where $\max\{\mathcal{F}^t\}$ denotes the best fitness obtained at generation t. That is, after determining the control parameters, the mutation, crossover, and selection operations are performed to obtain the next generation. The relative improvement is considered as the outcome of the application of the sampled control parameters.

A higher reward (improvement) indicates that the determined control parameters have a more positive impact on the search for global optimum.

5) Transition: The transition is also a probability $p(S_{t+1}|A_t = a_t, S_t = s_t)$. In our case, the probability distribution is not available. It will be seen in the following that the transition distribution does not affect the learning.

D. Learn the Control Parameter by Policy Gradient

A variant of the RL algorithm, PG, is able to deal with the scenario when the transition probability is not known. PG works by updating the policy parameters via stochastic gradient ascent on the expectation of the reward

$$\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta U(\theta_t) \qquad (11)$$

where α_t denotes the learning rate and $\nabla_\theta U(\theta)$ is the gradient of the cumulative reward $U(\theta)$.

An evolutionary search procedure with a finite number of generations (denoted as T) can be considered as a finite-horizon MDP. For such an MDP, given previously defined state and action, a trajectory τ is $\{S_0, A_0, r_1, \ldots, S_{T-1}, A_{T-1}, r_T\}$. The joint probability of the trajectory can be written as

$$q(\tau; \theta) = p(S_0) \prod_{t=0}^{T-1} \pi(A_t|S_t; \theta) p(S_{t+1}|A_t, S_t). \qquad (12)$$

Furthermore, $\nabla_\theta U(\theta)$ can be derived as follows:

$$\begin{aligned} \nabla_\theta U(\theta) &= \sum_\tau r(\tau)\nabla_\theta q(\tau; \theta) = \sum_\tau r(\tau)\frac{\nabla_\theta q(\tau; \theta)}{q(\tau; \theta)} q(\tau; \theta) \\ &= \sum_\tau r(\tau) q(\tau; \theta) \nabla_\theta\left[\log q(\tau; \theta)\right] \\ &= \sum_\tau r(\tau) q(\tau; \theta)\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(A_t|S_t; \theta)\right] \end{aligned} \qquad (13)$$

where $r(\tau)$ is the cumulative reward of the trajectory τ. The expectation of (13) can be calculated by sampling L

---

[2]A histogram is constructed by dividing the entire range of values into a series of intervals (i.e., bins), and count how many values fall into each bin.

Algorithm 1 Learning to Control the Parameters of the DE

Require: the LSTM parameter $\mathbf{W}$, the number of epochs Q, the number of training functions M, the population size N, the number of trajectories L, the trajectory length T and the learning rate α

1: Initialize $\mathbf{W}$ uniformly at random;
2: **for** *epoch* = $1 \rightarrow Q$ **do**
3:     ÂăInitialize $\mathcal{P}^0 = [\mathbf{x}_1^0, \cdots, \mathbf{x}_N^0]$ uniformly at random;
4:     **for** $k = 1 \rightarrow M$ **do**
5:         Set $\mathcal{P}_k^0 = \mathcal{P}^0$;
6:         Evaluate $\mathcal{F}_k^0 = \{f_k(\mathbf{x}_i^0), 1 \leq i \leq N\}$;
7:         ▷ *trajectory sampling;*
8:         **for** $l = 1 \rightarrow L$ **do**
9:             Set $t \leftarrow 0$, $\mathcal{H}_k^0 = \mathbf{0}$ and $\mathcal{C}_k^0 = \mathbf{0}$;
10:            **repeat**
11:                Compute $\mathcal{U}_k^t = [\mathbf{h}_t, \bar{\mathbf{h}}_t]$ by Eq. 8;
12:                Set $\mathcal{A}_k^t = [\mathcal{U}_k^t, \mathcal{F}_k^t]$;
13:                Apply LSTM: $\Theta_k^t, \mathcal{C}_k^{t+1}, \mathcal{H}_k^{t+1} \leftarrow \text{LSTM}(\mathcal{A}_k^t, \mathcal{H}_k^t, \mathcal{C}_k^t; \mathbf{W})$;
14:                Create the trial population: $\widetilde{\mathcal{P}}_k^t = [\widetilde{\mathbf{x}}_1^t, \cdots, \widetilde{\mathbf{x}}_N^t] \leftarrow \mathbf{CR} \circ \mathbf{M}(\mathcal{P}_k^t; \Theta_k^t)$;
15:                Evaluate the trial population: $\widetilde{\mathcal{F}}_k^t \leftarrow \{f(\widetilde{\mathbf{x}}_i^t), 1 \leq i \leq N\}$;
16:                Form the new population: $\mathcal{P}_k^{t+1}, \mathcal{F}_k^{t+1} \leftarrow \mathbf{S}(\mathcal{F}_k^t, \widetilde{\mathcal{F}}_k^t, \mathcal{P}_k^t, \widetilde{\mathcal{P}}_k^t)$;
17:                Calculate r_k^{t+1} using Eq. (10); and set $t \leftarrow t + 1$;
18:            **until** $t \geq T$
19:         **end for**
20:     **end for**
21:     ▷ *policy parameter updating;*
22:     Update $\mathbf{W}$ using Eq. (14) and Eq. (11);
23: **end for**
24: **return** $\mathbf{W}$;

---

trajectories $\tau^1, \ldots, \tau^L$

$$\nabla U_\theta(\theta) \approx \frac{1}{L} \sum_{i=1}^{L} r(\tau^i) \sum_{t=0}^{T-1} \nabla_\theta \ln \pi \left(A_t^{(i)} = a_t^{(i)} | S_t^{(i)} = s_t^{(i)}; \theta \right) \quad (14)$$

where $a_t^{(i)}$ (resp. $s_t^{(i)}$) denotes action (resp. state) value at time t in the ith trajectory.

A detailed description on how to update $\mathbf{W}$ is given in Algorithm 1. In the algorithm, to obtain the optimal $\mathbf{W}$, the optimization experience of a set of M training functions is used. For each function, first a set of L trajectories is sampled by applying the proposed DE (lines 8–19) for T generations. The control parameters of the proposed DE are obtained by forward computation of the LSTM given the present $\mathbf{W}$ at each generation. With the sampled trajectories, the reward at each generation for each optimization function is computed, and used for updating $\mathbf{W}$ (line 22).

Note that in the learning, a set of optimization functions are used. For each optimization function, the proposed DE is applied for T generations to sample the trajectories. Each trajectory can be considered as an optimization experience for a particular training function. For each function, there are L trajectories sampled. M functions can provide $L \times M$ optimization experiences. Learning from these experiences could thus be able to lead to a good parameter controller.

E. Embed the Learned Controller Within the DE

After training, it is assumed that we have secured the required knowledge for generating control parameters through

the learning from optimization experiences. Given a new test problem, the proposed DE with the learned parameter controller can be applied directly. The detailed algorithm, named as the learned DE (LDE), is summarized in Algorithm 2.

In Algorithm 2, the evolution procedure is the same as a typical DE except that the control parameters are the samples of the output of the controller (lines 8–10). One of the inputs of the LSTM, the statistics $\mathcal{U}^t$ is computed at each generation (line 6), and the hidden information $\mathcal{H}^t$ and $\mathcal{C}^t$ are initialized (line 3) and maintained during the evolution.

Note that in the LDE, the controller contains knowledge learned from optimization experiences which can be considered as extraneous/offline information, while the use of $\mathcal{U}^t$, $\mathcal{H}^t$ and $\mathcal{C}^t$ represent the information learned during the search procedure which is intraneous/online information. The time complexity for one generation of LDE is $O(H^2 + N \cdot H + N \cdot n)$, where H denotes the number of neurons used in the hidden layer.

IV. EXPERIMENTAL STUDY

In this section, we first present the implementation details of both Algorithms 1 and 2. The training details and the comparison results against some known DEs and a state-of-the-art EA are presented afterward.

Some or all functions in CEC'13 [53][3] are used as the training functions. In the comparison study, functions that have

---

[3]https://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2013/CEC2013.htm

---

Algorithm 2 LDE

Require: the trained agent with parameter $\mathbf{W}$
1: Initialize population $\mathcal{P}^0$ uniformly at random;
2: Evaluate $\mathcal{F}^0 = f(\mathcal{P}^0)$;
3: Set $g \leftarrow 0$, $\mathcal{H}^g = 0$ and $\mathcal{C}^g = 0$;
4: **while** the termination criteria have not been met **do**
5:     Compute $\mathcal{U}^g = [\mathbf{h}_g, \bar{\mathbf{h}}_g]$ by Eq. 8;
6:     Set $\mathcal{A}^g = [\mathcal{U}^g, \mathcal{F}^g]$;
7:     $g \leftarrow g + 1$;
8:     $\Theta^{g-1}, \mathcal{C}^g, \mathcal{H}^g \leftarrow \text{LSTM}(\mathcal{A}^{g-1}, \mathcal{H}^{g-1}, \mathcal{C}^{g-1}; \mathbf{W})$;
9:     $\widetilde{\Theta}^{g-1} \sim \mathcal{N}(\Theta^{g-1}, \sigma^2)$;
10:    $\mathcal{P}^g, \mathcal{F}^g \leftarrow \text{DE}(\mathcal{P}^{g-1}, \mathcal{F}^{g-1}; \widetilde{\Theta}^{g-1})$;
11: **end while**
12: **return** $\mathbf{x}^* = \arg\max f(\mathcal{P}^g)$

---

not been used for training from CEC'13 or CEC'17 [54][4] are tested. The CEC'13 test suite consists of five unimodal functions f_1–f_5, 15 basic multimodal functions f_6–f_{20}, and eight composition functions f_{21}–f_{28}. The CEC'17 test suite includes two unimodal functions F_1 and F_2, seven simple multimodal functions F_3–F_9, ten hybrid functions F_{10}–F_{19}, and ten more complex composition functions F_{20}–F_{29}.

When training, the following settings were used, including the number of epochs $Q = 150$, the population size $N = 50$ when $n = 10$ and $N = 100$ when $n = 30$ for the mutation strategy, the number of bins $b = 5$, the number of previous generations $g = 5$, the trajectory length $T = 50$, the number of trajectories $L = 20$, and the learning rate $\alpha = 0.005$.

For the experimental comparison, the same criteria as explained in [53] and [54] are used. Each algorithm is executed 51 runs for each function. The algorithm terminates if the maximum number of objective function evaluations (MAXNFE) exceeds $n \times 10^4$ or the difference between the function values of the found best solution and the optimal solution (also called the function error value) is smaller than 10^{-8}.

The compared algorithms include the following.

DE [12]: The original DE algorithm with DE/rand/1/bin mutation and binomial crossover.

JADE [37]: The classical adaptive DE method in which the DE/current-to-*p*best/1 mutation strategy was first proposed. JADE has two versions. One is with an external archive. The external archive is to aid the generation of offspring when mutation. The other is without the archive. In JADE, each F (resp. CR) is generated for each individual by sampling from a Cauchy (resp. normal) distribution and the location parameter of the distribution is updated by the Lehmer (resp. arithmetic) mean of the successful F's (resp. CR's).

jSO [55][5]: Which ranked second and the best DE-based algorithm in the CEC'17 competition. As an elaborate variation of JADE, two scale factors are associated with the mutation strategy. They are different from each other and limited within various bounds along the evolution.

CoBiDE [56][6]: In which the F (resp. CR) values are generated from a bimodal distribution consisting of two Cauchy distributions. Trial vectors are formed in the Eigen coordinate system built by the eigenvectors of the covariance matrix of the top individuals.

cDE [14][7]: In which both F and CR are selected from a predefined pool. The selection probability is proportional to the corresponding number of the successful individuals obtained from previous generations.

CoBiDE-PCM [24][8] and cDE-PCM [24][9]: That were proposed in [24] for studying the effect of parameter control management. These two algorithms are largely consistent with CoBiDE and cDE, but are equipped with different mutation and crossover operators. They are ranked first or second on the BBOB benchmark in [24].

HSES [57][10]: The winner of the bound constraint competition of CEC'18.[11] HSES is a three-phase algorithm. A modified univariate sampling is employed in the first stage for good initial points, while CMA-ES [58] is used in the second stage, followed by another univariate sampling, for local refinement.

The parameters and hyperparameters of these methods are kept the same as the settings in the original references in our experiments. Table II shows the detailed variation operators used and the parameter (hyperparameter) settings for the compared algorithms.

It should be noted that JADE, cDE, and CoBiDE are not tested on the CEC'13 or CEC'17 test suites in the original references, but on 20 basic functions, six basic functions, and CEC'05 [59], respectively. The parameters of these algorithms are tuned manually by grid search based on the mean fitness values found over a number of independent runs for each function. It is expected that the parameter tuning procedures of these algorithms are time consuming and computationally intensive. However, we should be frank that there is a possibility that these algorithms' performances could be improved if their parameters are tuned on the CEC'13 or CEC'17 test suites.

A. Comparison Results on the CEC'13 Test Suite

In this experiment, we use the first 20 functions f_1–f_{20} in CEC'13 as the training functions. The remaining eight functions f_{21}–f_{28} are used for comparison. As in [54], the function error value is used as the metric, and recorded for each run. The mean and standard deviation of the error values obtained for each function over 51 runs are used for comparison. In the following, the experimental results are summarized in tables, in which the means, standard deviations, and the Wilcoxon rank-sum hypothesis test results are included. The best (minimum) mean values are typeset in bold.

---

[6]The code is available at https://sites.google.com/view/pcmde/.

[7]The code is available in the extended CD version of the original article at https://www1.osu.cz/~tvrdik/wp-content/uploads/men05_CD.pdf.

[8]The code is available at https://sites.google.com/view/pcmde/.

[9]The code is available at https://sites.google.com/view/pcmde/.

[10]The code is available at https://github.com/P-N-Suganthan/CEC2018.

[11]The test functions in CEC'18 are the same as those in CEC'17 except the function F_2 in CEC'17 is removed.

---

[4]https://github.com/P-N-Suganthan/CEC2017-BoundContrained

[5]The code is available at https://github.com/P-N-Suganthan/CEC2017.

TABLE II
DETAILED REPRODUCTION OPERATORS USED AND PARAMETER AND HYPERPARAMETER SETTINGS OF THE COMPARED ALGORITHMS

Alg.	Operations		Control Parameters	Hyper-parameters
	Mutation	Crossover		
DE	DE/rand/1	bin	$N = 5n, F = 0.5, CR = 0.8$	NA
JADE	DE/current-to-pbest/1	bin	$N = 30, 100$ when $n = 10, 30$	$N_{\text{archive}} = N, \mu_F = 0.5$ $\mu_{CR} = 0.5, c = 0.1, p = 0.05$
jSO	DE/current-to-pbest-w/1	bin	$N_{\text{init}} = 25 \log(n) \sqrt{n}, N_{\text{min}} = 4$	$N_{\text{archive}} = 20, M_F = 0.5, M_{CR} = 0.8$ $p_{\max} = 0.25, p_{\min} = 0.125, H = 5$
CoBiDE	DE/rand/1	bin, covariance matrix learning	$N = 60$	$pb = 0.4, ps = 0.5$
CoBiDE-PCM	DE/best/1	bin	$N = 5 \times n$	NA
cDE	DE/rand/1, DE/best/2	bin	$N = \max(20, 2n), F = \{0.5, 0.8, 1\}$ $CR = \{0, 0.5, 1\}$	$n_0 = 2, \delta = 1/45$
cDE-PCM	DE/rand/1 ($n = 10$) DE/best/1 ($n = 30$)	bin bin	$N = 5n, F = \{0.5, 0.8, 1\}$ $CR = \{0, 0.5, 1\}$	$n_0 = 2, \delta = 1/45$
HSES	modified univariate sampling, CMA-ES		$M = 200$	$N_1 = 100, N_2 = 160, cc = 0.96$ $I = 20, \lambda = 3 \ln n + 80$

TABLE III
MEANS AND STANDARD DEVIATIONS OF FUNCTION ERROR VALUES FOR COMPARISON OF LDE ON THE CEC'13 BENCHMARK
SUITE FOR $n = 10$ AT GENERATION $T = 50$

	LDE		DE			JADE w/o archive			JADE with archive			jSO		
	Mean	Std. Dev.	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
f_{21}	5.39E+02	3.73E+01	4.07E+02	6.79E+00	+	**3.99E+02**	**1.37E+01**	+	4.01E+02	9.99E-01	+	4.01E+02	1.50E-01	+
f_{22}	1.71E+03	1.83E+02	2.00E+03	2.30E+02	−	**1.55E+03**	**2.11E+02**	+	1.56E+03	2.29E+02	+	1.78E+03	1.85E+02	≈
f_{23}	2.25E+03	2.23E+02	2.16E+03	1.90E+02	+	2.26E+03	2.03E+02	≈	2.29E+03	2.16E+02	≈	**1.94E+03**	**2.03E+02**	+
f_{24}	2.28E+02	5.48E+00	2.23E+02	6.16E+00	+	2.23E+02	3.46E+00	+	2.24E+02	3.52E+00	+	**2.21E+02**	**4.01E+00**	+
f_{25}	2.30E+02	2.35E+00	2.22E+02	3.14E+00	+	2.22E+02	3.72E+00	+	2.22E+02	5.97E+00	+	**2.19E+02**	**2.49E+00**	+
f_{26}	1.98E+02	1.14E+01	1.68E+02	1.81E+01	+	1.80E+02	2.88E+01	+	1.89E+02	3.99E+01	+	**1.62E+02**	**1.74E+01**	+
f_{27}	6.66E+02	3.08E+01	5.25E+02	4.71E+01	+	4.89E+02	7.35E+01	+	5.32E+02	6.64E+01	+	**4.72E+02**	**2.51E+01**	+
f_{28}	9.31E+02	1.05E+02	5.24E+02	4.73E+01	+	4.16E+02	8.88E+01	+	4.43E+02	9.77E+01	+	**3.92E+02**	**1.61E+01**	+
$+/ \approx /-$			7/0/1			7/1/0			7/1/0			7/1/0		

TABLE IV
MEANS AND STANDARD DEVIATIONS OF FUNCTION ERROR VALUES FOR COMPARISON OF LDE ON THE CEC'13 BENCHMARK
SUITE FOR $n = 10$ WHEN MAXNFE HAS BEEN REACHED

	LDE		DE			JADE w/o archive			JADE with archive			jSO		
	Mean	Std. Dev.	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
f_{21}	**2.35E+02**	**9.04E+01**	3.75E+02	7.10E+01	−	3.96E+02	2.78E+01	−	4.00E+02	5.68E-14	−	4.00E+02	5.68E-14	−
f_{22}	1.73E+01	2.60E+01	4.83E+02	2.82E+02	−	1.87E+01	3.20E+01	≈	2.68E+01	4.00E+01	≈	**6.80E+00**	**1.24E+01**	+
f_{23}	7.66E+02	2.62E+02	1.21E+03	1.51E+02	−	5.56E+02	2.07E+02	+	5.88E+02	2.29E+02	+	**2.13E+02**	**1.15E+02**	+
f_{24}	**1.87E+02**	**3.80E+01**	1.97E+02	1.96E+01	−	2.06E+02	5.80E+00	≈	2.06E+02	5.39E+00	≈	1.99E+02	1.03E+01	−
f_{25}	2.01E+02	7.87E+00	2.00E+02	1.41E+00	+	2.04E+02	5.14E+00	≈	2.05E+02	1.08E+01	−	**2.00E+02**	**7.73E-05**	+
f_{26}	1.16E+02	1.80E+01	1.34E+02	2.92E+01	−	1.46E+02	4.71E+01	≈	1.57E+02	6.06E+01	≈	**1.05E+02**	**8.38E+00**	+
f_{27}	3.31E+02	4.85E+01	**3.00E+02**	**7.96E-15**	+	3.25E+02	6.34E+01	+	3.63E+02	9.75E+01	−	3.00E+02	4.92E-05	+
f_{28}	**2.25E+02**	**9.67E+01**	2.88E+02	4.71E+01	−	3.04E+02	5.05E+01	≈	3.00E+02	3.43E+01	≈	3.00E+02	0.00E+00	−
$+/ \approx /-$			2/0/6			2/4/2			1/4/3			5/0/3		

The Wilcoxon rank-sum hypothesis test is performed to test the significant differences between LDE and the compared algorithms. The test results are shown by using symbols $+$, $-$, and $\approx$ in the tables. The symbol $+$ (resp. $-$ and $\approx$) indicates that LDE performs significantly worse than (resp. better than and similar to) the compared algorithms at a significance level of 0.05. The results are summarized in the "WR" column of the tables.

Tables IV and V summarize the experimental results when terminating at MAXNFE, while Table III shows the results when the algorithms terminate at the maximum number of generations $T = 50$ which is the number of generations used for training the agent.

From Table III, we can see that LDE does not perform satisfactorily. It is worse than the compared algorithms when the algorithm terminates at generation $T = 50$. However, it performs better as the process of evolution continues up to

MAXNFE. Note that in the training, the optimization procedure does not terminate at MAXNFE. The poor performance of LDE implies that the agent trained in $T = 50$ generations is not good enough. However, the experimental results show that the knowledge learned in $T = 50$ generations can be beneficial for the evolutionary search in further generations.

Tables IV and V show that when MAXNFE has been reached, LDE exhibits superior overall performance as compared with DE/rand/1/bin and JADE with archive and shows similar performance as compared with JADE without archive. However, LDE is outperformed by jSO in five and six out of eight 10-D and 30-D test functions, respectively.

Specifically, on the eight 10-D complex composition functions, LDE performs better than DE on six functions $f_{21}, f_{22}, f_{23}, f_{24}, f_{26}$, and f_{28} and surpasses JADE with archive on three functions f_{21}, f_{25}, and f_{27} and that without archive on two functions f_{21} and f_{22}. However, LDE performs statistically

TABLE V

MEANS AND STANDARD DEVIATIONS OF FUNCTION ERROR VALUES FOR COMPARISON OF LDE ON THE CEC'13 BENCHMARK
SUITE FOR $n = 30$ WHEN MAXNFE HAS BEEN REACHED

	LDE		DE			JADE w/o archive			JADE with archive			jSO		
	Mean	Std. Dev.	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
f_{21}	3.34E+02	9.21E+01	**2.70E+02**	**5.88E+01**	+	2.91E+02	6.54E+01	+	2.93E+02	6.88E+01	+	3.12E+02	8.28E+01	≈
f_{22}	3.04E+02	8.11E+01	6.17E+03	3.21E+02	−	9.40E+01	3.07E+01	+	**8.58E+01**	**3.55E+01**	+	1.22E+02	4.88E+00	+
f_{23}	4.28E+03	9.20E+02	7.18E+03	2.25E+02	−	3.37E+02	3.70E+02	+	3.46E+03	2.57E+02	+	**2.51E+03**	**3.14E+02**	+
f_{24}	2.07E+02	3.75E+00	2.04E+02	6.92E-01	+	2.06E+02	4.97E+00	≈	2.11E+02	1.03E+01	≈	**2.00E+02**	**1.06E-01**	+
f_{25}	2.46E+02	3.09E+01	2.53E+02	1.10E+01	≈	2.75E+02	1.08E+01	−	2.75E+02	1.15E+01	−	**2.35E+02**	**1.56E+01**	+
f_{26}	2.00E+02	2.77E-03	2.02E+02	5.29E-01	−	2.18E+02	4.55E+01	−	2.08E+02	3.06E+01	−	**2.00E+02**	**2.16E-10**	+
f_{27}	3.69E+02	3.38E+01	3.86E+02	2.38E+01	−	6.23E+02	2.07E+02	−	7.11E+02	2.13E+02	−	**3.04E+02**	**3.11E+00**	+
f_{28}	**3.00E+02**	**0.00E+00**	3.00E+02	1.18E-06	−	**3.00E+02**	**0.00E+00**	≈	**3.00E+02**	**0.00E+00**	≈	**3.00E+02**	**0.00E+00**	≈
$+/≈/−$			2/1/5			3/2/3			3/2/3			6/2/0		

TABLE VI

AVERAGE RANKS OF THE COMPARED ALGORITHMS ACCORDING TO
THEIR APS VALUES ON THE LAST EIGHT FUNCTIONS IN THE CEC'13
TEST SUITE

Alg.	jSO	LDE	JADE w/o archive	JADE with archive	DE
$n = 10$	**0.625**	1.250	1.375	1.875	1.625
$n = 30$	**0.250**	1.625	1.625	1.875	2.250
Avg.	**0.4375**	1.437	1.500	1.875	1.937

better than jSO on only three functions f_{21}, f_{24}, and f_{28} out of the eight test functions. For 30-D test problems, LDE yields better performance than DE on five functions $f_{22}, f_{23}, f_{26}, f_{27}$, and f_{28}. LDE shows the same advantage over JADE with and without archive on f_{25}, f_{26}, and f_{27} on the CEC'13 benchmarks with 30-D. Again, LDE performs worse than jSO on six test functions f_{22}–f_{27} with 30-D.

To see the overall performances, we rank the compared algorithms by using the average performance score (APS) [60]. The APS is defined based on the error values obtained by the compared algorithms for the test functions. Suppose there are m algorithms $A_1, \ldots, A_m$ to compare on a set of M functions. For each $i, j \in [1, m]$, if A_j performs better than A_i on the kth function $F_k, k \in [1, M]$ with statistical significance (i.e., $p < 0.05$), then set $\delta_{ij} = 1$; otherwise, $\delta_{ij} = 0$. The performance score of A_i on F_k is computed as follows:

$$P_k(A_i) = \sum_{j \in [1,n] \setminus \{i\}} \delta_{ij}. \tag{15}$$

The AP value of A_i is the average of the performance score values of A_i over the test functions. A smaller APS value indicates a better performance. Table VI summarizes the ranks of the compared algorithms in terms of their APS values. It can be seen that jSO is superior to LDE. LDE ranks the second, which is better than the other algorithms. This shows that the proposed method is quite promising.

Fig. 4 shows the evolution of the control parameters during the optimization obtained by the learned controller and jSO when optimizing f_{21}. In the figure, we show the mean values of F and CR at each generation by clustering $\mathcal{F}^t$ into three groups. The upper plot shows the mean F values, while the lower plot shows the mean CR values associated with the individuals in the groups. From the upper plot of Fig. 4(a) for the learned controller, it is seen that high-quality individuals generally have a smaller F value than the low-quality individuals, while the middle-quality individuals have higher CR values.
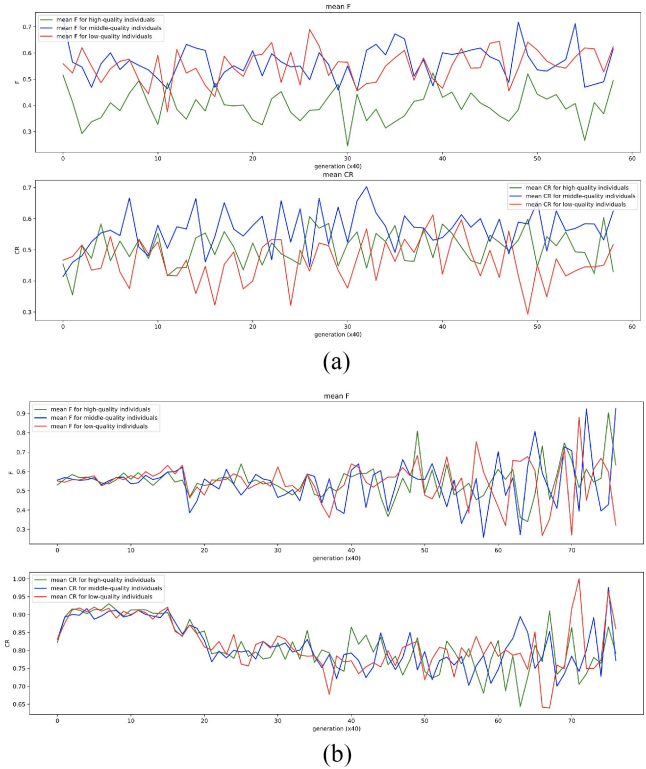
Fig. 4. Evolution of F and CR along optimization for f_{21} obtained by the learned controller. The upper (resp. lower) plot shows the F (resp. CR) values. The population's fitness is grouped into three clusters at each generation. The associated F and CR values are averaged. (a) LDE. (b) jSO.

From Fig. 4(b) for jSO, it is seen that along evolution, the F and CR values become scattered.

The better performance of LDE when running more generations indicates that the learned controller is promising for adaptive parameter control.

B. Comparison Results on the CEC'17 Test Suite

In this section, all 28 test problems in the CEC'13 test suite are used as the training functions. LDE is then compared with the other algorithms on the 29 functions of the CEC'17 test suite. Tables VII and VIII summarize the means and standard deviations of the function error values obtained by all the compared methods over 51 times on the CEC'17 test suite for 10-D and 30-D, respectively.

TABLE VII
MEANS AND STANDARD DEVIATIONS OF THE ERROR VALUES OBTAINED BY LDE AND THE COMPARED ALGORITHMS ON THE CEC'17 BENCHMARK SUITE FOR $n = 10$ WHEN MAXNFE HAS BEEN REACHED OR FUNCTION ERROR IS LESS THAN 10^{-8}

	LDE		DE			JADE w/o archive			JADE with archive			jSO		
	Mean	Std. Dev.	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
F_1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_2	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_3	**0.00E+00**	**0.00E+00**	1.37E+00	5.06E-01	−	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_4	4.53E+00	1.91E+00	1.68E+01	5.92E+00	−	3.66E+00	1.16E+00	≈	3.68E+00	1.19E+00	≈	2.07E+00	8.08E-01	+
F_5	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_6	1.49E+01	2.13E+00	3.07E+01	5.01E+00	−	1.34E+01	1.38E+00	+	1.38E+01	1.82E+00	+	1.20E+01	5.65E-01	+
F_7	4.58E+00	1.98E+00	1.73E+01	6.21E+00	−	3.85E+00	1.00E+00	≈	3.79E+00	1.41E+00	≈	2.04E+00	7.06E-01	+
F_8	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	≈	1.76E-03	1.24E-02	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_9	1.87E+02	1.31E+02	7.55E+02	2.30E+02	−	1.26E+02	1.06E+02	+	1.21E+02	9.55E+01	+	**4.86E+01**	**6.38E+01**	+
F_{10}	9.75E-02	3.55E-01	3.23E-01	4.69E-01	−	1.94E+00	9.13E-01	−	1.78E+00	9.85E-01	−	**0.00E+00**	**0.00E+00**	+
F_{11}	2.20E+01	4.59E+01	9.88E+00	3.20E+01	≈	3.18E+02	1.74E+02	−	3.66E+02	2.10E+02	−	2.72E+01	1.66E+01	≈
F_{12}	1.95E+00	2.21E+00	3.15E+00	2.35E+00	−	5.39E+00	3.57E+00	−	4.94E+00	3.10E+00	−	2.05E+00	2.31E+00	≈
F_{13}	3.12E-01	5.39E-01	3.71E-01	6.22E-01	≈	2.03E+00	5.46E+00	−	2.76E+00	6.51E+00	−	1.56E-01	3.62E-01	≈
F_{14}	3.30E-02	8.62E-02	1.24E-01	1.88E-01	≈	2.91E-01	1.58E-01	−	2.85E-01	2.34E-01	−	3.14E-01	2.08E-01	−
F_{15}	3.14E-01	2.36E-01	4.86E-01	2.86E-01	−	7.71E+00	2.79E+01	−	2.97E+00	1.65E+01	−	6.46E-01	1.98E-01	−
F_{16}	3.68E-01	4.08E-01	3.53E-01	3.04E-01	≈	5.00E-01	2.84E+00	−	5.25E-01	2.77E+00	−	5.93E-01	4.22E-01	−
F_{17}	1.14E-01	9.09E-02	1.14E-01	2.07E-01	≈	5.88E+00	8.87E+00	−	7.85E+00	9.63E+00	−	2.43E-01	2.09E-01	−
F_{18}	1.01E-02	1.17E-02	**4.97E-03**	**9.34E-03**	+	3.49E-02	2.07E-01	≈	1.44E-01	4.26E-01	≈	3.35E-03	6.92E-03	≈
F_{19}	1.22E-02	6.06E-02	2.40E-01	2.49E-01	−	4.28E-02	1.24E-01	≈	4.90E-01	2.76E+00	−	3.68E-01	1.88E-01	−
F_{20}	**1.25E+02**	**4.55E+01**	1.70E+02	5.90E+01	−	1.79E+02	4.58E+01	−	1.90E+02	3.80E+01	−	1.42E+02	5.05E+01	≈
F_{21}	9.06E+01	2.88E+01	1.00E+02	2.27E-01	−	9.62E+01	1.94E+01	−	1.00E+02	1.57E-01	−	9.88E+01	8.54E+00	≈
F_{22}	3.06E+02	3.04E+00	3.04E+02	2.78E+00	+	3.06E+02	1.98E+00	≈	3.05E+02	1.69E+00	≈	**3.01E+02**	**1.24E+00**	+
F_{23}	**2.42E+02**	**1.16E+02**	3.06E+02	7.83E+01	−	3.19E+02	5.47E+01	−	3.11E+02	6.96E+01	−	2.70E+02	1.00E+02	−
F_{24}	4.07E+02	1.80E+01	4.17E+02	2.30E+01	−	4.28E+02	2.18E+01	−	4.22E+02	2.71E+01	−	4.08E+02	1.87E+01	−
F_{25}	**3.00E+02**	**0.00E+00**	**3.00E+02**	**0.00E+00**	≈	3.00E+02	2.55E+01	≈	3.38E+02	1.76E+02	≈	**3.00E+02**	**0.00E+00**	≈
F_{26}	3.90E+02	1.38E+00	3.92E+02	2.61E+00	−	3.92E+02	2.86E+00	−	3.91E+02	6.93E+00	≈	**3.89E+02**	**3.56E-01**	+
F_{27}	3.00E+02	5.78E+01	3.51E+02	1.09E+02	≈	4.19E+02	1.40E+02	−	4.74E+02	1.49E+02	−	**3.00E+02**	**0.00E+00**	≈
F_{28}	2.33E+02	4.39E+00	2.36E+02	5.75E+00	−	2.56E+02	2.06E+01	−	2.48E+02	9.30E+00	−	2.37E+02	2.93E+00	−
F_{29}	3.97E+02	8.13E+00	4.54E+02	8.53E+01	−	1.45E+05	3.11E+05	−	1.58E+05	3.34E+05	−	**3.93E+02**	**2.87E+00**	+
$+/\approx/-$			2/11/16			2/12/15			2/11/16			7/15/7		

	CoBiDE			CoBiDE-PCM			cDE			cDE-PCM			HSES		
	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
F_1	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_2	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_3	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_4	3.57E+00	1.52E+00	+	1.14E+01	5.01E+00	−	4.63E+00	1.83E+00	≈	3.47E+00	1.10E+00	+	**1.01E+00**	**8.92E-01**	+
F_5	**0.00E+00**	**0.00E+00**	≈	2.68E-06	5.33E-06	−	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_6	1.35E+01	1.99E+00	+	1.94E+01	5.70E+00	−	1.55E+01	2.71E+00	≈	1.45E+01	1.48E+00	≈	**1.14E+01**	**7.20E-01**	+
F_7	3.77E+00	1.70E+00	+	1.29E+01	5.72E+00	−	5.74E+00	2.09E+00	−	3.62E+00	1.62E+00	+	**5.46E-01**	**7.71E-01**	+
F_8	**0.00E+00**	**0.00E+00**	≈	7.65E-02	1.69E-01	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_9	9.18E+01	9.47E+01	+	4.01E+02	2.37E+02	−	1.88E+02	1.09E+02	≈	1.80E+02	7.37E+01	≈	1.04E+02	1.56E+02	+
F_{10}	9.75E-02	2.96E-01	≈	5.18E+00	5.39E+00	−	5.55E-01	9.06E-01	−	**0.00E+00**	**0.00E+00**	≈	7.80E-02	2.67E-01	≈
F_{11}	**1.67E-01**	**1.30E-01**	≈	3.62E+02	2.01E+02	−	1.39E+02	1.59E+02	−	5.83E+01	2.31E+01	≈	1.09E+01	3.10E+01	≈
F_{12}	**1.07E+00**	**1.82E+00**	+	8.38E+00	5.47E+00	−	5.05E+00	2.95E+00	−	1.63E+00	2.13E+00	≈	3.44E+00	2.48E+00	−
F_{13}	**0.00E+00**	**0.00E+00**	+	9.58E+00	1.02E+01	−	6.49E-01	9.64E-01	−	1.95E-02	1.38E-01	+	6.92E+00	3.24E+01	−
F_{14}	**9.31E-03**	**3.49E-02**	+	2.57E+00	2.69E+00	−	3.09E-01	4.80E-01	−	6.38E-03	9.38E-03	≈	5.59E-01	7.50E-01	−
F_{15}	**2.25E-01**	**1.51E-01**	≈	3.65E+01	5.42E+01	−	2.96E+00	1.67E+01	≈	2.99E-01	1.29E-01	≈	3.07E+00	1.64E+01	−
F_{16}	1.57E+00	6.12E-01	−	1.32E+01	2.39E+01	−	1.33E+00	8.04E+00	≈	**8.27E-02**	**1.04E-01**	+	1.66E+01	1.09E+01	−
F_{17}	**6.38E-03**	**2.75E-02**	+	2.24E+01	2.45E+01	−	1.14E+00	3.89E+00	−	7.34E-02	1.45E-01	−	5.24E-01	4.11E-01	−
F_{18}	8.99E-03	1.33E-02	≈	1.12E+00	9.42E-01	−	4.20E-03	8.00E-03	≈	1.59E-02	1.24E-02	−	8.22E-01	1.75E+00	−
F_{19}	**0.00E+00**	**0.00E+00**	≈	6.25E+00	2.35E+01	−	9.91E-02	1.96E-01	≈	**0.00E+00**	**0.00E+00**	≈	1.24E+01	1.06E+01	−
F_{20}	1.38E+02	5.10E+01	≈	1.91E+02	4.52E+01	−	1.79E+02	4.86E+01	−	1.44E+02	5.22E+01	≈	1.91E+02	3.04E+01	−
F_{21}	**7.86E+01**	**4.12E+01**	≈	9.70E+01	2.02E+01	−	9.28E+01	2.63E+01	−	9.22E+01	2.69E+01	≈	1.00E+02	0.00E+00	≈
F_{22}	3.05E+02	1.57E+00	≈	3.12E+02	4.90E+00	−	3.07E+02	2.54E+00	−	3.05E+02	2.27E+00	≈	3.01E+02	1.67E+00	+
F_{23}	2.74E+02	1.02E+02	≈	3.25E+02	6.59E+01	−	3.26E+02	4.58E+01	−	2.79E+02	9.96E+01	≈	3.28E+02	7.42E-01	≈
F_{24}	4.00E+02	9.06E+00	≈	4.20E+02	2.31E+01	−	4.17E+02	2.32E+01	−	**4.01E+02**	**1.07E+01**	+	4.46E+02	1.02E+00	−
F_{25}	**3.00E+02**	**0.00E+00**	≈	3.30E+02	1.36E+02	≈	3.19E+02	1.29E+02	≈	**3.00E+02**	**0.00E+00**	≈	**3.00E+02**	**0.00E+00**	≈
F_{26}	3.89E+02	8.94E-01	+	3.98E+02	1.68E+01	−	3.90E+02	1.99E+00	≈	3.89E+02	9.90E-01	+	3.97E+02	1.81E+00	−
F_{27}	3.06E+02	3.93E+01	≈	4.32E+02	1.37E+02	−	4.04E+02	1.38E+02	−	3.11E+02	5.51E+01	≈	5.99E+02	2.55E+01	−
F_{28}	**2.30E+02**	**2.75E+00**	+	2.53E+02	2.37E+01	−	2.36E+02	6.37E+00	−	2.32E+02	4.27E+00	≈	2.64E+02	1.09E+01	−
F_{29}	3.96E+02	3.47E+00	+	2.50E+05	4.16E+05	−	8.06E+04	2.43E+05	−	4.04E+02	1.80E+01	−	4.17E+02	2.49E+01	−
	12/16/1			0/5/24			0/13/16			6/21/2			5/10/14		

For 10-D test functions, LDE exhibits superiority over HSES and most of the conventional and classical DE-based algorithms, except for CoBiDE and cDE-PCM. It performs similar to jSO.

Particularly, LDE performs better than the classical DE and JADE with archive on 16 functions, CoBiDE-PCM on 24 functions, cDE on 16 functions, and HSES on 14 functions. LDE performs worse than CoBiDE on 12 functions, jSO on

TABLE VIII
MEANS AND STANDARD DEVIATIONS OF THE ERROR VALUES OBTAINED BY LDE AND THE COMPARED ALGORITHMS ON THE CEC'17 BENCHMARK SUITE FOR $n = 30$ WHEN MAXNFE HAS BEEN REACHED OR FUNCTION ERROR IS LESS THAN 10^{-8}

	LDE		DE			JADE w/o archive			JADE with archive			jSO		
	Mean	Std. Dev.	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
F_1	**0.00E+00**	**0.00E+00**	9.21E+00	3.42E+00	−	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_2	3.25E-05	6.70E-05	4.20E+04	6.20E+03	−	6.02E+03	1.34E+04	−	6.96E+03	1.51E+04	−	**0.00E+00**	**0.00E+00**	+
F_3	4.41E+01	2.65E+01	6.11E+01	5.07E+00	−	2.98E+01	2.97E+01	+	4.26E+01	2.70E+01	+	5.86E+01	0.00E+00	≈
F_4	3.71E+01	7.35E+00	1.83E+02	7.53E+00	−	2.71E+01	4.34E+00	+	2.67E+01	4.26E+00	+	9.51E+01	1.93E+00	+
F_5	1.66E-04	6.07E-05	8.86E-04	1.98E-04	−	**0.00E+00**	**0.00E+00**	+	**0.00E+00**	**0.00E+00**	+	8.24E-09	3.29E-08	+
F_6	6.66E+01	8.80E+00	2.26E+02	8.94E+00	−	5.50E+01	3.79E+00	+	5.37E+01	3.60E+00	+	**3.89E+01**	**1.72E+00**	+
F_7	3.86E+01	9.40E+00	1.86E+02	9.06E+00	−	2.64E+01	4.17E+00	+	2.50E+01	4.41E+00	+	9.44E+00	1.81E+00	+
F_8	**0.00E+00**	**0.00E+00**	2.35E-09	7.56E-09	≈	5.27E-03	2.11E-02	≈	3.37E-02	1.08E-01	≈	**0.00E+00**	**0.00E+00**	≈
F_9	1.80E+03	3.63E+02	6.81E+03	2.76E+02	−	1.89E+03	1.93E+02	≈	1.87E+03	2.55E+02	≈	1.58E+03	2.25E+02	+
F_{10}	1.20E+01	4.33E+00	7.57E+01	1.86E+01	−	3.10E+01	2.67E+01	−	3.12E+01	2.50E+01	−	**3.63E+00**	**2.12E+00**	+
F_{11}	2.13E+03	2.05E+03	9.85E+05	3.93E+05	−	2.92E+03	2.95E+03	−	1.21E+03	4.25E+02	−	1.34E+02	1.06E+02	+
F_{12}	2.28E+01	8.46E+00	1.93E+02	2.16E+01	−	4.27E+01	2.96E+01	−	1.52E+03	6.29E+03	−	**1.24E+01**	**8.22E+00**	+
F_{13}	2.64E+01	9.72E+00	9.53E+01	7.13E+00	−	2.32E+03	1.03E+04	−	5.78E+03	9.17E+03	−	2.29E+01	1.55E+00	≈
F_{14}	7.45E+00	2.40E+00	6.22E+01	5.56E+00	−	3.99E+02	2.64E+03	−	1.28E+03	4.06E+03	−	**2.29E+00**	**1.24E+00**	+
F_{15}	4.18E+02	1.72E+02	1.13E+03	1.35E+02	−	4.07E+02	1.44E+02	≈	4.12E+02	1.44E+02	≈	**1.21E+02**	**1.04E+02**	+
F_{16}	4.10E+01	1.94E+01	3.44E+02	5.56E+01	−	6.82E+01	1.41E+01	−	7.18E+01	2.80E+01	−	**3.46E+01**	**5.89E+00**	≈
F_{17}	2.31E+01	4.07E+00	1.49E+02	2.27E+01	−	1.51E+04	4.34E+04	−	1.16E+04	3.16E+04	−	2.12E+01	5.39E-01	≈
F_{18}	5.99E+00	9.65E-01	2.38E+01	2.57E+00	−	1.15E+01	5.32E+00	−	1.70E+03	4.61E+03	−	**3.39E+00**	**6.35E-01**	+
F_{19}	5.62E+01	5.50E+01	1.59E+02	8.72E+01	−	1.07E+02	5.37E+01	−	1.11E+02	5.41E+01	−	3.17E+01	6.69E+00	≈
F_{20}	2.39E+02	8.39E+00	3.75E+02	1.01E+01	−	2.28E+02	4.31E+00	+	2.26E+02	4.55E+00	+	**2.09E+02**	**2.09E+00**	+
F_{21}	**1.00E+02**	**0.00E+00**	1.00E+02	1.56E-07	−	**1.00E+02**	**0.00E+00**	≈	1.00E+02	2.33E+00	≈	**1.00E+02**	**0.00E+00**	≈
F_{22}	3.85E+02	1.00E+01	5.26E+02	1.04E+01	−	3.72E+02	5.38E+00	+	3.73E+02	5.77E+00	+	**3.48E+02**	**3.57E+00**	+
F_{23}	4.65E+02	1.19E+01	5.95E+02	7.42E+00	−	4.39E+02	4.61E+00	+	4.42E+02	5.67E+00	+	4.24E+02	2.02E+00	+
F_{24}	3.87E+02	6.48E+01	3.87E+02	6.63E-02	−	3.87E+02	2.12E-01	−	3.87E+02	1.52E-01	−	3.87E+02	1.42E-02	−
F_{25}	1.25E+03	4.05E+02	2.72E+03	1.02E+02	−	1.17E+03	1.38E+02	+	1.19E+03	6.15E+01	+	8.95E+02	2.91E+01	+
F_{26}	4.97E+02	8.45E+00	**4.84E+02**	**1.28E+01**	+	5.03E+02	7.62E+00	−	5.04E+02	6.44E+00	−	4.92E+02	8.09E+00	+
F_{27}	3.06E+02	2.52E+01	3.30E+02	3.87E+01	−	3.37E+02	5.38E+01	−	3.40E+02	5.74E+01	≈	**3.00E+02**	**0.00E+00**	+
F_{28}	4.37E+02	3.42E+01	9.58E+02	9.09E+01	−	4.80E+02	3.21E+01	−	4.77E+02	2.73E+01	−	4.37E+02	1.76E+01	≈
F_{29}	1.99E+03	4.46E+01	8.32E+03	1.55E+03	−	2.18E+03	1.59E+02	−	2.13E+03	1.62E+02	−	**1.96E+03**	**1.10E+01**	+
$+/\approx/-$			1/1/27			9/5/15			9/7/13			20/8/1		

	CoBiDE			CoBiDE-PCM			cDE			cDE-PCM			HSES		
	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR	Mean	Std. Dev.	WR
F_1	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈	**0.00E+00**	**0.00E+00**	≈
F_2	**0.00E+00**	**0.00E+00**	+	**0.00E+00**	**0.00E+00**	+	**0.00E+00**	**0.00E+00**	+	**0.00E+00**	**0.00E+00**	+	**0.00E+00**	**0.00E+00**	+
F_3	4.20E+01	2.79E+01	≈	2.83E+01	2.94E+01	+	5.69E+01	1.08E+01	≈	3.23E+01	2.98E+01	+	**4.31E+00**	**8.62E+00**	+
F_4	3.99E+01	9.66E+00	≈	7.80E+01	2.12E+01	−	5.07E+01	6.12E+00	−	4.86E+01	8.45E+00	−	**8.76E+00**	**2.65E+00**	+
F_5	3.49E-08	4.73E-08	+	2.57E-02	6.51E-02	−	**0.00E+00**	**0.00E+00**	+	3.73E-02	1.60E-01	≈	**0.00E+00**	**0.00E+00**	+
F_6	7.12E+01	1.00E+01	−	1.15E+02	2.44E+01	−	9.21E+01	6.39E+00	−	8.74E+01	9.67E+00	−	4.07E+01	3.81E+00	+
F_7	3.92E+01	1.07E+01	≈	7.27E+01	2.03E+01	−	5.82E+01	9.09E+00	−	4.99E+01	7.49E+00	−	**7.88E+00**	**2.87E+00**	+
F_8	0.00E+00	0.00E+00	≈	3.10E+01	4.44E+01	−	7.53E-01	1.43E+00	−	3.04E+00	2.23E+00	−	**0.00E+00**	**0.00E+00**	≈
F_9	1.82E+03	4.52E+02	≈	2.81E+03	5.74E+02	−	2.33E+03	2.84E+02	−	2.72E+03	2.97E+02	−	**9.90E+02**	**3.86E+02**	+
F_{10}	1.62E+01	9.72E+00	−	1.23E+02	6.39E+01	−	2.02E+01	1.19E+01	−	1.74E+02	6.24E+01	−	1.37E+01	2.14E+01	−
F_{11}	2.83E+03	5.16E+03	−	1.18E+04	1.08E+04	−	2.02E+04	1.43E+04	−	1.11E+04	1.11E+04	−	**4.42E+01**	**1.00E+02**	+
F_{12}	2.51E+01	8.58E+00	≈	1.12E+02	1.30E+02	−	5.27E+01	2.28E+01	−	2.28E+03	8.37E+03	−	2.90E+01	1.33E+01	≈
F_{13}	**1.09E+01**	**4.66E+00**	+	1.82E+02	7.05E+01	−	3.41E+01	9.14E+00	−	2.21E+02	6.77E+01	−	1.43E+01	1.08E+01	+
F_{14}	6.85E+00	2.77E+00	≈	1.68E+02	1.19E+02	−	1.86E+01	4.77E+00	−	3.50E+02	1.50E+02	−	5.59E+00	3.71E+00	+
F_{15}	3.78E+02	1.46E+02	≈	7.32E+02	2.57E+02	−	4.88E+02	1.35E+02	−	4.56E+02	2.13E+02	≈	2.33E+02	2.04E+02	+
F_{16}	4.38E+01	3.27E+01	≈	2.50E+02	1.54E+02	−	9.68E+01	2.88E+01	−	1.61E+02	1.24E+02	−	5.93E+01	1.06E+02	−
F_{17}	**1.86E+01**	**8.57E+00**	+	9.05E+01	6.82E+01	−	3.16E+01	5.41E+00	−	8.72E+02	1.29E+03	−	2.07E+01	5.78E+00	+
F_{18}	4.78E+00	1.43E+00	+	1.14E+02	6.85E+01	−	1.54E+01	2.29E+00	−	2.21E+02	8.50E+01	−	3.88E+00	1.59E+00	+
F_{19}	**4.31E+01**	**5.74E+01**	+	2.37E+02	1.27E+02	−	1.00E+02	5.43E+01	−	1.11E+02	8.17E+01	−	1.57E+02	5.04E+01	−
F_{20}	2.43E+02	9.18E+00	−	2.75E+02	1.86E+01	−	2.58E+02	7.12E+00	−	2.50E+02	1.07E+01	−	2.09E+02	3.93E+00	+
F_{21}	**1.00E+02**	**0.00E+00**	≈	5.74E+02	1.12E+03	−	2.08E+02	5.32E+02	≈	1.01E+02	1.54E+00	≈	**1.00E+02**	**0.00E+00**	≈
F_{22}	3.88E+02	8.88E+00	−	4.25E+02	2.63E+01	−	3.98E+02	6.50E+00	−	3.99E+02	1.49E+01	−	3.52E+02	8.46E+00	+
F_{23}	4.64E+02	1.17E+01	≈	4.97E+02	2.11E+01	−	4.80E+02	8.36E+00	−	4.81E+02	2.54E+01	−	**4.19E+02**	**5.49E+00**	+
F_{24}	3.87E+02	4.68E-01	−	3.88E+02	2.57E+00	−	3.87E+02	4.70E-01	−	3.91E+02	1.00E+01	−	**3.87E+02**	**2.67E-02**	+
F_{25}	1.37E+03	2.88E+02	−	1.94E+03	3.74E+02	−	1.55E+03	2.01E+02	−	1.48E+03	4.62E+02	−	**8.93E+02**	**1.44E+02**	+
F_{26}	4.97E+02	1.03E+01	≈	5.25E+02	1.39E+01	−	4.96E+02	1.23E+01	≈	5.43E+02	2.70E+01	−	5.17E+02	8.85E+00	−
F_{27}	3.28E+02	4.72E+01	−	3.57E+02	6.20E+01	≈	3.24E+02	4.59E+01	−	3.60E+02	6.40E+01	≈	3.24E+02	4.38E+01	−
F_{28}	**4.30E+02**	**4.64E+01**	+	6.66E+02	1.36E+02	−	5.20E+02	4.85E+01	−	5.81E+02	1.44E+02	−	4.65E+02	6.57E+01	≈
F_{29}	2.06E+03	8.33E+01	−	2.22E+03	2.39E+02	−	2.13E+03	1.44E+02	−	2.38E+03	2.38E+02	−	2.05E+03	3.35E+01	−
$+/\approx/-$	7/15/7			2/2/25			2/4/23			2/5/22			17/6/6		

seven functions, and cDE-PCM on six functions. LDE performs similar to CoBiDE and jSO on 16 and 15 functions, respectively.

For 30-D test problems, it is seen that jSO and HSES perform better than LDE on most of the test functions. However, LDE performs better than the rest of the algorithms in general.

TABLE IX
AVERAGE RANKING OF THE COMPARED ALGORITHMS ACCORDING TO THEIR APS VALUES ON THE CEC'17 BENCHMARK FUNCTIONS

Alg.		jSO	CoBiDE	LDE	HSES	JADE w/o archive	JADE with archive	cDE-PCM	cDE	DE	CoBiDE-PCM
	$n = 10$	1.4482	**0.7586**	1.2414	4.0345	3.5172	3.5172	1.0690	3.7241	3.5517	6.3793
Rank	$n = 30$	**0.5517**	2.1379	2.3793	1.0345	3.1034	3.1379	5.8276	4.4138	7.1379	6.6207
	Avg.	**1.0**	1.4483	1.8103	2.5345	3.3103	3.3276	3.4483	4.0690	5.3448	6.5

TABLE X
RESULTS OF LAST EIGHT PROBLEMS OF CEC'13 WITH DIFFERENT CELL SIZES FOR $n = 10$ AT TERMINATION

cell size	500		1000		1500		2000		2500		3000	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
f_{21}	4.00E+02	0.00E+00	3.94E+02	4.09E+01	3.90E+02	4.96E+01	**2.35E+02**	**9.04E+01**	3.69E+02	7.28E+01	3.96E+02	2.78E+01
f_{22}	3.15E+02	1.06E+02	5.45E+01	2.32E+01	2.95E+01	2.39E+01	1.73E+01	2.60E+01	**1.26E+01**	**1.52E+01**	2.02E+01	3.10E+01
f_{23}	1.13E+03	1.81E+02	1.08E+03	1.43E+02	**6.04E+02**	**2.02E+02**	7.66E+02	2.62E+02	6.52E+02	2.27E+02	6.28E+02	1.99E+02
f_{24}	**1.52E+02**	**3.28E+01**	1.72E+02	3.80E+01	1.74E+02	3.79E+01	1.87E+02	3.80E+01	1.70E+02	4.29E+01	1.88E+02	3.31E+01
f_{25}	1.99E+02	5.84E+00	1.94E+02	1.76E+01	1.96E+02	1.65E+01	2.01E+02	7.87E+00	**1.92E+02**	**2.60E+01**	1.93E+02	2.46E+01
f_{26}	1.17E+02	1.23E+01	1.14E+02	1.26E+01	1.07E+02	1.33E+01	1.16E+02	1.80E+01	1.10E+02	1.34E+01	**1.07E+02**	**6.15E+00**
f_{27}	3.08E+02	2.22E+01	3.01E+02	3.48E+00	**3.00E+02**	**3.33E-02**	3.31E+02	4.85E+01	3.10E+02	2.97E+01	3.06E+02	2.35E+01
f_{28}	2.97E+02	1.91E+01	2.97E+02	2.09E+01	2.96E+02	2.77E+01	**2.25E+02**	**9.67E+01**	2.65E+02	7.62E+01	2.76E+02	6.44E+01

TABLE XI
RESULTS OF LAST EIGHT PROBLEMS OF CEC'13 WITH DIFFERENT POPULATION SIZES AND CELL SIZES FOR $n = 30$ AT TERMINATION

N	100		100		100		150		200	
cell size	2000		3000		3500		3000		1500	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
f_{21}	3.36E+02	9.03E+01	3.34E+02	9.21E+01	**3.08E+02**	**6.18E+01**	3.42E+02	8.43E+01	3.21E+02	5.01E+01
f_{22}	**1.33E+02**	**1.29E+01**	3.04E+02	8.11E+01	3.69E+03	2.74E+02	5.04E+02	1.23E+02	3.13E+03	1.62E+02
f_{23}	**3.43E+03**	**4.80E+02**	4.28E+03	9.20E+02	7.11E+03	3.02E+02	4.14E+03	7.97E+02	6.88E+03	3.56E+02
f_{24}	2.09E+02	4.68E+00	2.07E+02	3.75E+00	2.07E+02	3.71E+00	2.04E+02	2.46E+00	**2.02E+02**	**2.01E+00**
f_{25}	2.58E+02	1.32E+01	**2.46E+02**	**3.09E+01**	2.50E+02	2.86E+01	2.73E+02	2.37E+01	2.73E+02	3.26E+01
f_{26}	2.00E+02	3.60E-03	2.00E+02	2.77E-03	2.00E+02	1.30E-02	**2.00E+02**	**1.98E-03**	2.00E+02	1.21E-02
f_{27}	4.51E+02	9.55E+01	3.69E+02	3.38E+01	3.49E+02	2.65E+01	5.79E+02	1.25E+02	**3.31E+02**	**2.29E+01**
f_{28}	3.00E+02	0.00E+00	3.00E+02	0.00E+00	**2.99E+02**	**4.33E+00**	3.00E+02	0.00E+00	3.00E+02	0.00E+00

Particularly, LDE performs better than classical DE and the other adaptive DEs on more functions than that it performs worse than these algorithms. The performance of LDE is similar to CoBiDE in the sense that the numbers of functions that LDE outperforms CoBiDE and CoBiDE outperforms LDE are the same.

The ranking result of all algorithms is shown in Table IX. It is seen that LDE ranks the third on the CEC'17 benchmark suite. Note that first the agent is learned from CEC'13. Its performance on CEC'17 implies that indeed some useful knowledge which is helpful for parameter control is effectively learned. Second, once the controller has been learned, it is applied to solve new test functions without requiring any tuning of the algorithmic parameters. This can greatly reduce possibly large amount of computational efforts.

V. SENSITIVITY ANALYSIS

One of the main parameters that greatly influence the performance of LDE is the number of neurons (i.e., the sizes of $\mathcal{H}^t$ and $\mathcal{C}^t$) used in the hidden layers. A higher number can increase the representation ability of the LSTM but may cause over-fitting. Here, we investigate the effect of different neuron sizes to the performance of LDE on the last eight functions f_{21}–f_{28} of CEC'13 for 10-D and 30-D.

Six agents with different number of neurons are learned on 10-D functions. A set of neuron sizes, from 500 to 3000 with an interval of 500, is studied when the population size is fixed as 50. The obtained results are summarized in Table X.

From Table X, we see that: 1) the performance of LDE differs w.r.t. the size of neurons; 2) for different functions, the best result is obtained by taking different neuron size; and 3) a higher number of neurons does not always mean better performance.

Generally speaking, the population size ought to be increased for problems with larger dimensions. To see the effect of population size, we carry out experiments to learn five controllers that are with different population and neuron sizes on the same CEC'13 training functions (i.e., f_1–f_{20} with 30-D). The performance of the learned controller is again tested on f_{21}–f_{28} with 30-D.

Table XI lists the comparison results of the five designed controllers. From the table, it is observed that the neuron size takes the same effects as those in 10-D case. Furthermore, it can be seen that the population and neuron size together have a very complex effect on the performance of the learned controller.

VI. RELATED WORK

In this article, RL is used as the main technique to learn on how to adaptively control the algorithmic parameters from optimization experiences. To the best of our knowledge, there is no related work on controlling the DE parameters by learning from optimization experiences. However, we found some works on controlling the parameters of genetic algorithms (GAs). These works relate to our approach but with significant differences.

In [61], four control parameters (including crossover rate, mutation rate, tournament proportion, and population size) of a GA are dynamically regulated with the help of the RL. The learning algorithm is a mix of Q-learning and SARSA which involves maintaining a discrete table of state–action pairs. In [61], information along the GA's search procedure is extracted as the state. Two RL algorithms switch in a predefined frequency to find a new action (i.e., control parameter value). The work shows that the RL-enhanced GA outperforms a steady-state GA in terms of fitness and success rate.

In [62], the Q-learning algorithm is applied to choose a suitable reproduction operator which can generate a promising individual in a short time. The authors propose a new reward function incorporating GA's multipoint search feature and the time complexity of recombination operators. Furthermore, the action-value function is updated after generating all individuals. Similarly, in [63], the Q-learning is also used to adaptively select reproduction operators. But the chosen operator is applied to the whole population. This method is shown empirically that it tends to avoid obstructive operators and thus solve the problems more efficiently than random selection.

In [64], a universal controller by using RL is found to be able to adapt to any existing EA and to adjust any parameter involved. In their method, a set of observables (considered as states) is fed to a binary decision tree consisting of only one root node for representing a universal state. SARSA [48] is carried out to update the state–action value. It is shown that the RL-enhanced controller exhibits superiority over two benchmark controllers on most common complex problems.

Here, we would like to point out the significant differences between the proposed approach and the aforementioned RL-based approaches. First, the RL methods, such as Q-learning or SARSA used in existing approaches are developed for MDP with the discrete state and action. Second, existing parameter controllers are not learnt from optimization experiences, but are updated based on the online information obtained from the search procedure during a single run for a single test problem. The main idea behind the existing study is the same as in the DE parameter control methods reviewed in the introduction. They all try to use information obtained online to update the control parameters. Third, different from RL which aims to learn an agent with a converged optimal policy, the policy derived from the state–action pairs in existing study is not necessarily convergent or even stable. However, the proposed approach in this article can learn from the extraneous information for a stable policy.

The only work that applies an idea similar to our approach is the DE-DDQN [65], in which a set of mutation operators is adaptively selected based on the learning from optimization experiences over a set of training functions. In DE-DDQN, double deep Q learning is applied for the selection. Various features are defined as states and taken as input to the DNN at each generation.

VII. Conclusion

This article proposed a new adaptive parameter controller by learning from the optimization experiences of a set of training functions. The adaptive parameter control problem was modeled as an MDP. A recurrent neural network, called LSTM, was employed as the parameter controller. The RL algorithm, PG, was used to learn the parameters of the LSTM. The learned controller was embedded within a DE for new test problem optimization. In the experiments, functions in the CEC'13 test suite were used in training. After training, the trained agent was studied on the CEC'13 and CEC'17 test suites in comparison with some well-known DE algorithms and a state-of-the-art EA. The experimental results showed that the LDE was very competitive to the compared algorithms which indicated the effectiveness of the proposed controller.

From our experimental study, we find that training the parameter controller for 30-D problems is rather difficult in terms of the computational resources. Particularly, the CPU/GPU time used in the training process is considerable. Furthermore, as the number of dimension increases, it is expected that there will be an increasing need for training time and powerful computing devices. It is also hard to choose the training functions to make the training stable. Moreover, there has no theoretical foundation or practical principles on deciding the cell size in the employed neural network. Another disadvantage of the learned algorithm is that its time complexity is greater than the compared algorithms.

Note that the training and test functions share similar features since they are all constructed by using the same basic functions. As a result, its performance over unrelated functions is not predictable, may be limited on totally different set of functions, such as real-world problems. A possible way to improve the applicability of LDE maybe is to use new learning techniques or incorporate existing DE techniques in the LDE.

In the future, we plan to improve the performance of the LDE in a number of ways, such as using different statistics $\mathcal{U}^t$, adopting different neural networks, considering different output of the neural network, and others. Furthermore, we intend to apply the LDE on some real-world optimization and engineering problems. We also intend to study on the use of RL for adaptive mutation/crossover strategy, on the learning for hyperparameters of state-of-the-art EAs, and on the learning for metaheuristics for combinatorial optimization problems.

References

[1] R. Storn and K. Price, *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces*, Int. Sci. Comput. Inst., Berkley, CA, USA, 1995.

[2] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[3] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution—An updated survey," *Swarm Evol. Comput.*, vol. 27, pp. 1–30, Apr. 2016.

[4] F. Neri and V. Tirronen, "Recent advances in differential evolution: A survey and experimental analysis," *Artif. Intell. Rev.*, vol. 33, nos. 1–2, pp. 61–106, 2010.

[5] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.

[6] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 1658–1665.

[7] J. Brest, M. S. Maučec, and B. Bošković, "The 100-digit challenge: Algorithm jDE100," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Wellington, New Zealand, 2019, pp. 19–26.

[8] U. Škvorc, T. Eftimov, and P. Korošec, "CEC real-parameter optimization competitions: Progress from 2013 to 2018," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Wellington, New Zealand, 2019, pp. 3126–3133.

[9] K. Price, N. Awad, M. Ali, and P. Suganthan, "The 2019 100-digit challenge on real-parameter, single objective optimization: Analysis of results," Dept. Computer Science, University of Freiburg, Freiburg, Germany, School of Computer Information Systems, Jordan University of Science and Technology, Jordan, and School of EEE, Nanyang Technol. Univ., Singapore, Rep. 2019, 2019. [Online]. Available: https://www.ntu.edu.sg/home

[10] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 124–141, Jul. 1999.

[11] G. Karafotias, M. Hoogendoorn, and A. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Trans. Evol. Comput.*, vol. 19, no. 2, pp. 167–187, Apr. 2015.

[12] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Adv. Intell. Syst. Fuzzy Syst. Evol. Comput.*, vol. 10, no. 10, pp. 293–298, 2002.

[13] J. Rönkkönen, S. Kukkonen, and K. V. Price, "Real-parameter optimization with differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, vol. 1, Edinburgh, U.K., 2005, pp. 506–513.

[14] J. Tvrdík, "Competitive differential evolution," in *Proc. 12th Int. Conf. Soft Comput. (MENDEL)*, Brno, Czech Republic, 2006, pp. 7–12.

[15] P. Kaelo and M. M. Ali, "Differential evolution algorithms using hybrid mutation," *Comput. Optim. Appl.*, vol. 37, no. 2, pp. 231–246, 2007.

[16] J. Tvrdík, "Adaptation in differential evolution: A numerical comparison," *Appl. Soft Comput.*, vol. 9, no. 3, pp. 1149–1155, 2009.

[17] V. Feoktistov, *Differential Evolution: In Search of Solutions*. New York, NY, USA: Springer, 2006.

[18] J. Brest, *Constrained Real-Parameter Optimization With ε-Self-Adaptive Differential Evolution*. Heidelberg, Germany: Springer, 2009, pp. 73–93.

[19] K. Price, *Eliminating Drift Bias From the Differential Evolution Algorithm*. Heidelberg, Germany: Springer, 2008, pp. 33–88.

[20] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, vol. 2. Edinburgh, U.K., 2005, pp. 1785–1791.

[21] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Process. Lett.*, vol. 17, no. 1, pp. 93–105, 2003.

[22] G.-Y. Li and M.-G. Liu, "The summary of differential evolution algorithm and its improvements," in *Proc. 3rd Int. Conf. Adv. Comput. Theory Eng. (ICACTE')*, vol. 3. Chengdu, China, 2010, pp. V3153–V3156.

[23] E.-N. Dragoi and V. Dafinescu, "Parameter control and hybridization techniques in differential evolution: A survey," *Artif. Intell. Rev.*, vol. 45, no. 4, pp. 447–470, 2016.

[24] R. Tanabe and A. Fukunaga, "Reviewing and benchmarking parameter control methods in differential evolution," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1170–1184, Mar. 2020.

[25] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *Proc. 7th Annu. Genet. Evol. Comput. Conf. (GECCO)*, Washington, DC, USA, 2005, pp. 991–998.

[26] D. Zou, J. Wu, L. Gao, and S. Li, "A modified differential evolution algorithm for unconstrained optimization problems," *Neurocomputing*, vol. 120, pp. 469–481, Nov. 2013.

[27] Y. Wang, Z. Cai, and Q. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 55–66, Feb. 2011.

[28] A. Draa, S. Bouzoubia, and I. Boukhalfa, "A sinusoidal differential evolution algorithm for numerical optimisation," *Appl. Soft Comput.*, vol. 27, pp. 99–126, Feb. 2015.

[29] M. M. Ali and A. A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Comput. Oper. Res.*, vol. 31, no. 10, pp. 1703–1725, 2004.

[30] V. Tirronen and F. Neri, *Differential Evolution With Fitness Diversity Self-Adaptation*. Heidelberg, Germany: Springer, 2009, pp. 199–234.

[31] L. Jia and C. Zhang, "An improved self-adaptive control parameter of differential evolution for global optimization," *Int. J. Digit. Content Technol. Appl.*, vol. 6, no. 8, pp. 343–350, 2012.

[32] T. Takahama and S. Sakai, "Efficient constrained optimization by the ε constrained rank-based differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Brisbane, QLD, Australia, 2012, pp. 1–8.

[33] L. Tang, Y. Dong, and J. Liu, "Differential evolution with an individual-dependent mechanism," *IEEE Trans. Evol. Comput.*, vol. 19, no. 4, pp. 560–574, Aug. 2015.

[34] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, Dec. 2006.

[35] F. Lezama, J. Soares, R. Faia, and Z. Vale, "Hybrid-adaptive differential evolution with decay function (HyDE-DF) applied to the 100-digit challenge competition on single objective numerical optimization," in *Proc. Genet. Evolut. Comput. Conf. Companion (GECCO)*, Prague, Czech Republic, 2019, pp. 7–8.

[36] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[37] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

[38] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Hong Kong, Jun. 2008, pp. 1110–1116.

[39] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Cancun, Mexico, 2013, pp. 71–78.

[40] N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Donostia-San Sebastian, Spain, 2017, pp. 372–379.

[41] A. W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi, "LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Donostia-San Sebastian, Spain, 2017, pp. 145–152.

[42] A. Zamuda, "Function evaluations upto 1e+12 and large population sizes assessed in distance-based success history differential evolution for 100-digit challenge and numerical optimization scenarios (DISHchain 1e+12): A competition entry for '100-digit challenge, and four other numerical optimization competitions' at the genetic and evolutionary computation conference (GECCO) 2019," in *Proc. Genet. Evolut. Comput. Conf. Companion (GECCO)*, Prague, Czech Republic, 2019, pp. 11–12.

[43] Z. Meng, J.-S. Pan, and K.-K. Tseng, "PaDE: An enhanced differential evolution algorithm with novel control parameter adaptation schemes for numerical optimization," *Knowl. Based Syst.*, vol. 168, pp. 80–99, Mar. 2019.

[44] Z. Meng, J.-S. Pan, and L. Kong, "Parameters with adaptive learning mechanism (PALM) for the enhancement of differential evolution," *Knowl. Based Syst.*, vol. 141, pp. 92–112, Feb. 2018.

[45] J. Brest, M. S. Maučec, and B. Bošković, "iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC')*, Vancouver, BC, Canada, 2016, pp. 1188–1195.

[46] Z. Zhao, J. Yang, Z. Hu, and H. Che, "A differential evolution algorithm with self-adaptive strategy and control parameters based on symmetric Latin hypercube design for unconstrained optimization problems," *Eur. J. Oper. Res.*, vol. 250, no. 1, pp. 30–45, 2016.

[47] Z.-H. Zhan and J. Zhang, "Self-adaptive differential evolution based on PSO learning strategy," in *Proc. Annu. Genet. Evol. Comput. Conf. (GECCO')*, Portland, OR, USA, 2010, pp. 39–46.

[48] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[50] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[51] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signal Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[52] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[53] J. Liang, B. Qu, P. N. Suganthan, and A. G. Hernández-Díaz, "Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization," Dept. Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China and Nanyang Technol. Univ., Singapore, Rep. 201212, 2013.

[54] N. H. Awad, M. Z. Ali, P. N. Suganthan, J. J. Liang, and B. Y. Qu, "Problem definitions and evaluation criteria for the CEC2017 special
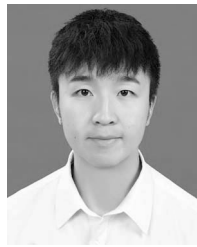
session and competition on single objective real-parameter numerical optimization," Nanyang Technological University, Singapore and Jordan University of Science and Technology, Jordan and Zhengzhou University, Zhengzhou China, Rep., 2016.

[55] J. Brest, M. S. Maučec, and B. Bošković, "Single objective real-parameter optimization: Algorithm jSO," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Donostia-San Sebastian, Spain, 2017, pp. 1311–1318.

[56] Y. Wang, H.-X. Li, T. Huang, and L. Li, "Differential evolution based on covariance matrix learning and bimodal distribution parameter setting," *Appl. Soft Comput.*, vol. 18, pp. 232–247, May 2014.

[57] G. Zhang and Y. Shi, "Hybrid sampling evolution strategy for solving single objective bound constrained problems," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Rio de Janeiro, Brazil, 2018, pp. 1–7.

[58] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003.

[59] P. N. Suganthan *et al.*, "Problem definitions and evaluation criteria for the CEC2005 special session on real-parameter optimization," Dept. Kanpur Genet. Algorithms Lab., IIT, Kanpur, India, KanGAL Rep. 2005005, 2005.

[60] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based many-objective optimization," *Evol. Comput.*, vol. 19, no. 1, pp. 45–76, Mar. 2011.

[61] A. E. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut, "Reinforcement learning for online control of evolutionary algorithms," in *Proc. Int. Workshop Eng. Self-Org. Appl. (ESOA)*, Hakodate, Japan, 2006, pp. 151–160.

[62] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta, "A method to control parameters of evolutionary algorithms by using reinforcement learning," in *Proc. 6th Int. Conf. Signal Image Technol. Internet Based Syst. (SITIS)*, Kuala Lumpur, Malaysia, 2010, pp. 74–79.

[63] A. Buzdalova, V. Kononov, and M. Buzdalov, "Selecting evolutionary operators using reinforcement learning: Initial explorations," in *Proc. Companion Publ. Genet. Evol. Comput. Conf. (GECCO Comp)*, Vancouver, BC, Canada, 2014, pp. 1033–1036.

[64] G. Karafotias, A. E. Eiben, and M. Hoogendoorn, "Generic parameter control with reinforcement learning," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Vancouver, BC, Canada, 2014, pp. 1319–1326.

[65] M. Sharma, A. Komninos, M. López-Ibáñez, and D. Kazakov, "Deep reinforcement learning based parameter control in differential evolution," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Prague, Czech Republic, 2019, pp. 709–717.

Xin Liu received the B.Sc. degree in electronics and information engineering and the M.Eng. degree in electronics and communication engineering from Northwestern Polytechnical University, Xi'an, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an.

His research interests include evolutionary algorithms and reinforcement learning.

Thomas Bäck (Senior Member, IEEE) received the Ph.D. degree in computer science from University Dortmund, Dortmund, Germany, in 1994, under the supervision of H.-P. Schwefel.

He is a Full Professor of Computer Science with the Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands, where he has been the Head of the Natural Computing Group since 2002. He gained experience in solving real-life problems in optimization and data mining with global enterprises, such as BMW, Beiersdorf, Daimler, Ford, and Honda. He has authored over 200 publications on natural computing, as well as two books on evolutionary algorithms, *Evolutionary Algorithms in Theory and Practice* (1996) and *Contemporary Evolution Strategies* (2013). He is a Co-Editor of *Handbook of Evolutionary Computation*, and most recently, *Handbook of Natural Computing*.

Dr. Bäck received the Best Dissertation Award from the German Society of Computer Science (Gesellschaft für Informatik, GI) in 1995. He is an editorial board member and an associate editor of a number of journals on evolutionary and natural computation and a Fellow of the International Society for Genetic and Evolutionary Computation, for his contributions to the field.

Jianyong Sun (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in mathematics from Xi'an Jiaotong University, Xi'an, China, in 1997 and 1999, respectively, and the Ph.D. degree in computer science from the University of Essex, Colchester, U.K., in 2006.

He is a Full Professor with the School of Mathematics and Statistics, Xi'an Jiaotong University. His research interests include evolutionary computation and optimization, statistical machine learning, and big data and their applications.

Zongben Xu received the Ph.D. degree in mathematics from Xi'an Jiaotong University, Xi'an, China, in 1987.

He served as the Chief Scientist with the National Basic Research Program of China (973 Project), and the Director of the Institute for Information and System Sciences, Xi'an Jiaotong University. His current research interests include intelligent information processing and applied mathematics.

Dr. Xu is a recipient of the Tan Kan Kee Science Award in Science Technology in 2018, the National Natural Science Award of China in 2007, the National Award on Scientific and Technological Advances of China in 2011, the CSIAM Su Buchin Applied Mathematics Prize in 2008, and the ITIQAM Richard Price Award. He delivered a 45-min talk on the International Congress of Mathematicians 2010. He was elected as a member of the Chinese Academy of Science in 2011.