A Gradient-based Method for Differential Evolution Parameter Control by Smoothing

Haotian Zhang
Xi'an Jiaotong University
Xi'an, ShaanXi, China
ht.zhang@xjtu.edu.cn

Jialong Shi
Xi'an Jiaotong University
Xi'an, ShaanXi, China
jialong.shi@xjtu.edu.cn

Jianyong Sun
Xi'an Jiaotong University
Xi'an, ShaanXi, China
jy.sun@xjtu.edu.cn

Ali W. Mohamed
Cairo University
Cairo, Egypt
aliwagdy@gmail.com

Zongben Xu
Xi'an Jiaotong University
Xi'an, ShaanXi, China
zbxu@mail.xjtu.edu.cn

ABSTRACT

Differential evolution (DE) is one of the most studied algorithms in evolutionary computation. However, the parameters in DE need to be tuned carefully, which costs much computational resources. The reason is that the basic paradigm of DE (mutation, crossover, bound constraint and selection) contains non-differentiable operators. In this paper, we propose a DE paradigm called "smoDE" for the first time by smoothing the crossover operator and the bound constraint operator to make them differentiable with respect to the parameters. The experiments show that we can tune the parameters of smoDE by gradient descent with much fewer computational resources than commonly used tools such as the Bayesian optimization algorithm (BOA). Then we analyze the population diversity of smoDE theoretically and prove that smoDE can converge faster than DE. A simple experiment also validates that. We further propose the "ada-smoDE" by embedding a neural network in smoDE to output parameters of smoDE adaptively and test ada-smoDE on the CEC 2018 test suite. The results show that ada-smoDE can perform competitively on the whole test suite and significantly better than DE on some problems.

CCS CONCEPTS

• **Computing methodologies** → **Continuous space search**; **Neural networks**.

KEYWORDS

Differential evolution, Parameter control, Learning to optimize.

ACM Reference Format:
Haotian Zhang, Jialong Shi, Jianyong Sun, Ali W. Mohamed, and Zongben Xu. 2024. A Gradient-based Method for Differential Evolution Parameter Control by Smoothing. In *GECCO '24:The Genetic and Evolutionary Computation Conference 2024, July 14–18, 2024, Melbourne, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

GECCO '24, July 14–18, 2024, Melbourne, Australia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Evolutionary computation is a fruitful research region which has been applied to several regions, including Community Detection [24] and Combinatorial Optimization [34]. There exist many kinds of evolutionary algorithms (EAs) [9] such as differential evolution (DE) [17, 20, 22], evolutionary strategies (ES) [12, 13], particle swarm optimization (PSO) [6]. Among these EAs, DE attracts more attention than other EAs as its good performance [5] in CEC competitions.

Parameter tuning and *parameter control* are two important frameworks to find proper parameters for EAs. In parameter tuning, the parameters are always set by prior knowledge before implementing the algorithm and the parameters are fixed during the optimization process. While in parameter control, the parameters are adjusted during the search process based on some collected information.

There exist some methods to tune the parameters such as Bayesian optimization algorithm (BOA) [7] and sequential model-based optimization for algorithm configuration (SMAC) [15]. In these methods, the best function value f_λ^* found by a specific algorithm $\mathcal{A}(\cdot)$ with given parameters λ is regarded as the function of the parameters, namely $f_\lambda^* = \langle \mathcal{A}(\lambda, \cdot), f(x) \rangle$, where f is the objective function and $\cdot$ in $\mathcal{A}$ denotes other inputs needed by the algorithm (e.g. the initial population, the calculation budget). For convenience we denote $\mathcal{A}(\lambda, \cdot)$ as $\mathcal{A}(\lambda)$. If the primal optimization problem is $min_x f(x)$, then finding the best parameters for an algorithm can be converted to an optimization problem :

$$\lambda^* = argmin_\lambda \langle \mathcal{A}(\lambda), f(x) \rangle \tag{1}$$

We denote $F(\lambda) = f_\lambda^* = \langle \mathcal{A}(\lambda), f(x) \rangle$ as "response function". The response function $F(\lambda)$ is always derivative-free since there are many non-differentiable operators in an evolutionary algorithm $\mathcal{A}$ (e.g. the crossover operator). Therefore, finding the optimal λ^* for EAs is a black-box optimization problem. BOA and SMAC are good tools for black-box optimization problems hence they can handle parameter tuning problems well. Since DE has fewer parameters (scaling factor F, crossover rate CR, population size NP) than many other evolutionary algorithms, thus it is easier to conduct parameter tuning on DE than on some other EAs.

Compared with tuning, parameter control framework attracts more efforts and many studies are proposed. In the parameter control framework, since the parameters are different in each generation, these EAs are called adaptive EAs and the mechanism for

controlling the parameters is called the adaptive parameter control mechanism (APC). For instance, adaptive PSO [32], adaptive ES [29] [33] and many adaptive DEs such as JADE [39], SHADE [26], LSHADE [27], iLSHADE [3], jSO [4] and many variants [11, 16, 18, 19, 21] are proposed recently. Interested readers can refer to the reviews of APC [1, 28]. However, these methods were designed without considering the optimization experiences of similar optimization problems.

Recently, some studies resort to using "meta learning/learning to optimize" to learn a good adaptive mechanism for parameter control. "Learning to optimize" was proposed in [2], which used a deep neural network to output the descent direction of a gradient descent method. For EAs, the concept of "learning to optimize" is borrowed to control parameters. For instance, QHSES and DQHSES [35, 37] used an intelligent agent to control a parameter for adaptively allocating the computational resources of two main components in HSES [33] (namely controlling the ratio of two components). The agent is trained by Q-learning [25] (a reinforcement learning method). In [36, 38], Zhang et al. proposed to use the deep neural networks to control parameters in DE. DE-DDQN [23] used the deep Q network to select operators for DE.

The studies based on "learning to optimize" for EA always use an agent or a deep neural network $\psi(\cdot, w)$ to output the parameters λ of the algorithm $\mathcal{A}(\lambda)$. Namely $\lambda = \psi(s, w)$ where s is the information collected during the search process of $\mathcal{A}(\lambda)$. Therefore, finding the optimal adaptive parameter control mechanism can be converted to finding the optimal parameters w^* for the neural network:

$$w^* = argmin_w \langle \mathcal{A}(\psi(\cdot, w)), f(x) \rangle \tag{2}$$

However, like parameter tuning, since there are non-differentiable operators in the evolutionary algorithm $\mathcal{A}$, we cannot use gradient-based methods to find w^*. To handle Eq. (2), the optimization methods in existing "learning to optimize" studies for EA can be divided into two categories: reinforcement learning and "black-box" optimization methods. For instance, the Deep Q-learning in [23] and the Q-learning in [37] do not need derivative information. "Black-box" optimization methods such as natural evolution strategies (NES) [30] is used in [38]. However, neither "black-box" optimization methods nor reinforcement learning are as efficient as gradient-based methods. Thus, for both parameter tuning and parameter control, it is a challenge to revise the paradigm of EAs such that gradient-based methods can be used to optimize Eq. (1) and Eq. (2).

In this paper, we propose a new paradigm for DE by smoothing the crossover and bound constraint operators. We obtain a new DE paradigm named "smoDE". The smoDE has the same parameters as DE (F, CR, NP). To verify the paradigm, we first show that the gradient descent method can be used to estimate the gradient of F, CR for smoDE so that we can find the optimal values of F, CR by a gradient descent method with few computational resources. We analyze the population diversity of smoDE, which shows that smoDE can converge faster than classic DE. Second, we embed a deep neural network into smoDE to output F, CR in each generation, thus we obtain adaptive smoDE, namely ada-smoDE. Similarly, to train the neural network, we use the gradient descent method instead of reinforcement learning or NES. In other words, we propose a gradient-based method for tuning/controlling parameters for DE. Third, we test ada-smoDE on the CEC 2018 test suite, which shows that ada-smoDE can perform competitively with the classic DE.

The paper is organized as follows: We briefly introduce DE and "learning to optimize" technology in Section 2. In Section 3 we introduce the core idea of smoDE and ada-smoDE. In Section 4 we analyze the population diversity of smoDE. In Section 5 we verify the effectiveness and efficiency of smoDE and ada-smoDE through several experiments. Finally, in Section 6, we conclude the paper.

2 PRELIMINARIES

2.1 DE

Differential evolution (DE) [5] is a kind of EA based on the population $X \in \mathbb{R}^{NP \times D}$ (NP is the population size and D is the dimension of the problem). Namely, X can be represented as $\{x_i\}_{i=1}^{NP}, x_i \in \mathbb{R}^D$. DE mainly contains four parts: mutation, crossover, bound constraint and selection. Mutation always uses several mutation operators to generate offspring. For the t-th generation, the mutation is taken as follows:

$$v_i^t = x_{r_1^i}^t + F \cdot (x_{r_2^i}^t - x_{r_3^i}^t), \quad i = 1, \cdots, NP \tag{3}$$

where r_1^i, r_2^i, r_3^i are three different random integers from $[1, NP]$ and F is a parameter called scaling factor. This operator is called $rand/1$ which is one of the most used operators. Crossover is taken as follows:

$$u_{i,j}^t = \begin{cases} v_{i,j}^t & \text{if } rand_j(0,1) \le CR \text{ or } j = j_{rand} \\ x_{i,j}^t & \text{else} \end{cases}$$

where $v_{i,j}^t$ is the j-th component of v_i^t, $rand_j(0,1)$ is a random variable follows uniform distribution and j_{rand} is a random integer from $[1, D]$. The control parameter CR is called the crossover rate. The crossover operator is a non-differentiable operator. As DE always handles problems with box boundary (B_{low}, B_{up}), the bound constraint is another component in DE. In this paper, we consider the strategy: If the solution is outside of the boundary, the function value is set as the boundary. That is:

$$Cu_{i,j}^t = \begin{cases} B_{up} & \text{if } u_{i,j}^t > B_{up} \\ u_{i,j}^t & \text{if } u_{i,j}^t \in [B_{low}, B_{up}] \\ B_{low} & \text{if } u_{i,j}^t < B_{low} \end{cases}$$

This bound constraint operator is also a non-differentiable operator. The selection is implemented as:

$$x_i^{t+1} = \begin{cases} Cu_i^t & \text{if } f(Cu_i^t) < f(x_i^t) \\ x_i^t & \text{else} \end{cases}$$

Namely, the individual with better function value will survive.

2.2 Learning to optimize and Bi-level optimization

"Learning to optimize" is firstly proposed in [2]. The core idea of "learning to optimize" is modeling the parameter tuning/controlling problem into a bi-level optimization problem, which means it has two optimization loops (inner loop and outer loop). First of all, we assume the primal optimization problem can be written as $min_x f(x)$ and the optimization method is $\mathcal{A}(\lambda)$ where λ is the parameter. For parameter tuning, our aim is to find the optimal parameter

λ^* (Eq. (1)). Here Eq. (1) equals to handle a bi-level optimization problem:

$$\begin{aligned} \text{Inner Loop:} \quad & f_\lambda^* = \langle \mathcal{A}(\lambda), f(x) \rangle \\ \text{Outer Loop:} \quad & \lambda^* = argmin_\lambda f_\lambda^* \end{aligned}$$

The inner loop is the primal optimization problem (minimize $f(x)$ by given parameter λ in algorithm $\mathcal{A}$), the outer loop is finding the optimal parameter.

For parameter control, Eq. (2) also equals to:

$$\begin{aligned} \text{Inner Loop:} \quad & f_w^* = \langle \mathcal{A}(\psi(\cdot, w)), f(x) \rangle \\ \text{Outer Loop:} \quad & w^* = argmin_w f_w^* \end{aligned}$$

We can see that, the difficulty of the outer loop is that $f_w^*(resp.f_\lambda^*)$ is non-differentiable w.r.t. $w(resp.\lambda)$ since there exist many complex operators in $\mathcal{A}$ for EAs.

3 METHOD

We propose smoDE in this section. As mentioned above, the crossover operator and bound constraint operator are two non-differentiable operators, thus the core idea of smoDE is to smooth the two operators. Note that we do not modify the selection operator in DE since it does not influence the gradient flow (which will be discussed in subsection 3.3). Then we embed a neural network to adaptively output the parameters of smoDE so that we can obtain ada-smoDE.

3.1 Smooth Crossover

The crossover in DE can be written as follows:

$$u_{i,j}^t = v_{i,j}^t + (x_{i,j}^t - v_{i,j}^t) \cdot g(\epsilon_{i,j} - CR) \cdot (1 - \delta(\eta_i - j))$$

where $\epsilon_{i,j} \sim \mathcal{U}(0,1)$ ($\mathcal{U}$ represents the uniform distribution), η_i is a random value from $1, 2, \cdots, D$, $g(\cdot)$ and $\delta(\cdot)$ are defined as follows:

$$g(x) = \begin{cases} 1, & x > 0 \\ 0, & x \le 0 \end{cases}, \delta(x) = \begin{cases} 1, & x = 0 \\ 0, & x \ne 0 \end{cases}$$

It is the functions $g(\cdot), \delta(\cdot)$ that make the operator non-differentiable w.r.t. CR. The core idea of smoothing the crossover operator is to calculate the expectation of $u_{i,j}^t$. We have:

$$\begin{aligned} \mathbb{E}_{\epsilon_{i,j}}[g(\epsilon_{i,j} - CR)] &= \int_{\epsilon_{i,j} > CR} d\epsilon_{i,j} = 1 - CR \\ \mathbb{E}_{\eta_i}[\delta(\eta_i - j)] &= p(\eta_i = j) \cdot 1 = \frac{1}{D} \end{aligned}$$

Thus we have:

$$\mathbb{E}_{\epsilon_{i,j}, \eta_i}[u_{i,j}^t] = v_{i,j}^t + (x_{i,j}^t - v_{i,j}^t) \cdot (1 - CR) \cdot \frac{D-1}{D} \tag{4}$$

3.2 Smooth bound constraint

We can rewrite the bound constraint operator as:

$$Cu_{i,j}^t = u_{i,j}^t + (B_{up} - u_{i,j}^t) g(u_{i,j}^t - B_{up}) \tag{5}$$

$$Cu_{i,j}^t = u_{i,j}^t + (B_{low} - u_{i,j}^t) g(B_{low} - u_{i,j}^t) \tag{6}$$

The problem here is that g is a non-smooth function, thus we borrow the idea of stochastic resonance [8, 10]. We add a normal gaussian noise $\varepsilon_{i,j}$ on $u_{i,j}^t$ and take the expectation:

$$\begin{aligned} \mathbb{E}_{\varepsilon_{i,j}}[g(u_{i,j}^t - B_{up} + \varepsilon_{i,j})] &= \int_{u_{i,j}^t - B_{up} + \varepsilon_{i,j} > 0} \phi(\varepsilon_{i,j}) d\varepsilon_{i,j} \\ &= \int_{\varepsilon_{i,j} > B_{up} - u_{i,j}^t} \phi(\varepsilon_{i,j}) d\varepsilon_{i,j} = 1 - \Phi(B_{up} - u_{i,j}^t) \end{aligned}$$

where $\phi(\cdot)$ is the probability density function (pdf) of the normal Gaussian distribution and $\Phi(\cdot)$ is the cumulative distribution function (cdf) of the normal Gaussian distribution. Similarly, we have:

$$\mathbb{E}_{\varepsilon_{i,j}}[g(B_{low} - u_{i,j}^t + \varepsilon_{i,j})] = 1 - \Phi(u_{i,j}^t - B_{low})$$

Thus Eq. (5) and Eq. (6) are smoothed as:

$$\tilde{Cu}_{i,j}^t = u_{i,j}^t + (B_{up} - u_{i,j}^t)(1 - \Phi(B_{up} - u_{i,j}^t)) \tag{7}$$

$$Cu_{i,j}^t = \tilde{Cu}_{i,j}^t + (B_{low} - \tilde{Cu}_{i,j}^t)(1 - \Phi(\tilde{Cu}_{i,j}^t - B_{low})) \tag{8}$$

3.3 smoDE

By combining the components above, we can summarize the flowchart of smoDE:

- Mutation: $v_{i,j}^t = x_{r_1^i,j}^t + F(x_{r_2^i,j}^t - x_{r_3^i,j}^t)$
- Crossover: $u_{i,j}^t = v_{i,j}^t + (x_{i,j}^t - v_{i,j}^t) \cdot (1 - CR) \cdot \frac{D-1}{D}$
- Constrain: Generate $Cu_{i,j}^t$ by Eq. (7) and Eq. (8)
- Selection:

$$x_i^{t+1} = \begin{cases} Cu_i^t & \text{if } f(Cu_i^t) < f(x_i^t) \\ x_i^t & \text{else} \end{cases}$$

Now we consider how to obtain the optimal parameters of smoDE. First of all, we need to define the loss function:

$$\begin{aligned} min_{F,CR} L(F, CR) &= min_{F,CR}[f(x_{F,CR}^*)] \\ &= min_{F,CR}[min_i f(x_i^G)] \end{aligned} \tag{9}$$

where G is the maximum number of generations. The loss function is defined as Eq. (1), namely given parameters F, CR, we aim to minimize the best function value found by smoDE.

Here we consider using gradient descent to optimize F and CR. First is the selection operator. We assume the best individual is the i-th individual in the G-th generation (namely x_i^G). x_i^G is obtained by two possible ways by selection: $x_i^G = x_i^{G-1}$ or $x_i^G = Cu_i^{G-1}$. For the former case, the F, CR do not contribute to x_i^G, thus the gradient is zero. For the latter case, Cu_i^{G-1} is generated by F, CR, thus

$$\nabla_F L(F, CR) = \nabla f(x_i^G) \cdot \frac{\partial x_i^G}{\partial Cu_i^{G-1}} \cdot \frac{\partial Cu_i^{G-1}}{\partial F} = \nabla f(x_i^G) \cdot \frac{\partial Cu_i^{G-1}}{\partial F}$$

$$\nabla_{CR} L(F, CR) = \nabla f(x_i^G) \cdot \frac{\partial x_i^G}{\partial Cu_i^{G-1}} \cdot \frac{\partial Cu_i^{G-1}}{\partial CR} = \nabla f(x_i^G) \cdot \frac{\partial Cu_i^{G-1}}{\partial CR}$$

Actually for x_i^G, we can always find a $Cu_i^{G-\alpha_i}, \alpha_i \le G$ so that $x_i^G = Cu_i^{G-\alpha_i}$. If not, it implies that the mutation and crossover do not generate any better solutions from the initial solutions. Then the gradient calculation process is as follows:

$$\nabla_{CR} L(F, CR) = \nabla f(x_i^G) \cdot \frac{\partial x_i^G}{\partial Cu_i^{G-\alpha_i}} \cdot \frac{\partial Cu_i^{G-\alpha_i}}{\partial u_i^{G-\alpha_i}} \cdot \frac{\partial u_i^{G-\alpha_i}}{\partial CR}$$

$$\nabla_F L(F, CR) = \nabla f(x_i^G) \cdot \frac{\partial x_i^G}{\partial Cu_i^{G-\alpha_i}} \cdot \frac{\partial Cu_i^{G-\alpha_i}}{\partial u_i^{G-\alpha_i}} \cdot \frac{\partial u_i^{G-\alpha_i}}{\partial v_i^{G-\alpha_i}} \cdot \frac{\partial v_i^{G-\alpha_i}}{\partial F}$$

where $\frac{\partial Cu_i^{G-\alpha_i}}{\partial u_i^{G-\alpha_i}}$ can be calculated based on Eq. (7) and Eq. (8), $\frac{\partial u_i^{G-\alpha_i}}{\partial CR}$, $\frac{\partial u_i^{G-\alpha_i}}{\partial v_i^{G-\alpha_i}}$ can be calculated based on Eq. (4) and $\frac{\partial v_i^{G-\alpha_i}}{\partial F}$ can be calculated based on Eq. (3). The detailed calculation of the gradient can be found in the appendix.

Finally, we talk about the calculation of $\frac{\partial u_i^{G-\alpha_i}}{\partial CR}$ and $\frac{\partial v_i^{G-\alpha_i}}{\partial F}$. When we calculate $\frac{\partial u_i^{G-\alpha_i}}{\partial CR}$, based on Eq. (4), we ignore that $x_i^{G-\alpha_i}$ is also generated by CR, thus the result is $\frac{\partial u_i^{G-\alpha_i}}{\partial CR} = -(x_i^{G-\alpha_i} - v_i^{G-\alpha_i}) \cdot \frac{D-1}{D}$. The technique is called "truncated gradient" which has been used in LSTM [14]. The "truncated gradient" can make gradient methods more efficient. For instance, in our problem, if we do not ignore $x_i^{G-\alpha_i}$ is also generated by CR, the computational resources will increase exponentially. Similarly, $\frac{\partial v_i^{G-\alpha_i}}{\partial F} = x_{r_2^i}^{G-\alpha_i} - x_{r_3^i}^{G-\alpha_i}$.

3.4 Learning period

We can see that our loss function Eq. (9) aims to minimize the best function value obtained by smoDE. In [2], different from Eq. (9), the loss function is the sum of function value in the optimization trajectory. We can also borrow the idea from it. In [2], $\mathcal{A}$ is the gradient method ($x_{k+1} = x_k - h_k \nabla f(x_k)$, h_k is the step size), the optimization trajectory is clear (i.e. $\{x_k\}_{k=0}^T$). In smoDE, as mentioned above, the best solution x_i^G is generated from $Cu_i^{G-\alpha_i}$, $Cu_i^{G-\alpha_i}$ is generated from $u_i^{G-\alpha_i}$, $u_i^{G-\alpha_i}$ is generated from $v_i^{G-\alpha_i}$, $v_i^{G-\alpha_i}$ is generated from $x_{r_1^i}^{G-\alpha_i}$. Thus we call $x_{r_1^i}^{G-\alpha_i}$ as the "parent" of x_i^G. Similarly, we can also find "parent" of $x_{r_1^i}^{G-\alpha_i}$. These individuals construct a trajectory. The "learning period" (LP) represents the number of individuals in the trajectory. For instance, if $LP = 3$, the loss function is $L = f(x_i^G) + f(x_j^{G_j}) + f(x_k^{G_k})$ where $j = r_1^i, G_j = G - \alpha_i, k = r_1^j, G_k = G_j - \alpha_j$, namely $x_j^{G_j}$ is the parent of x_i^G and $x_k^{G_k}$ is the parent of $x_j^{G_j}$. We call this gradient-based tuning method **GBT**.

3.5 ada-smoDE

As mentioned above, the gradient method can tune the parameters of smoDE efficiently. However, adaptively generating parameters is the trend of DEs in recent years [28]. In this subsection, we embed a neural network into smoDE to adaptively output F, CR. The network is a basic multi-layer perceptron (MLP) that has one hidden layer and the activation function is sigmoid on both the hidden layer and the output layer. The input of the network is defined as follows:

$$Input = [s_1, s_2], \text{where } s_1 = mean(std(X)), s_2 = mean(f(X))$$

And the output is $[F, CR]$. For the training of the network, we can resort to the normal back propagation (BP) algorithm. The resultant algorithm is named adaptive smooth DE (ada-smoDE). The algorithm for ada-smoDE and the training algorithm are summarized in the appendix.

4 THEORETICAL ANALYSIS

In this section, we analyze the population diversity of smoDE. Population diversity is first analyzed by Zaharie in [31]. It measures the difference between the variance of parent population X and the variance of offspring population U. In [38], Zhang et al. analyzed the hybrid mutation operator (Eq. (10)):

$$v_i^t = \left[\omega x_i^t + (1-\omega) x_{\text{best}}^t\right] \gamma + (1-\gamma) x_{r_1^i}^t + W(x_{r_2^i}^t - x_{r_3^i}^t) \quad (10)$$

where W, ω, γ are parameters of the operator. The theoretical result shows:

$$\mathbb{E}[\mathbb{D}(V)] = \left[2W^2 + \frac{NP-1}{NP}(1-\gamma)^2 + \omega^2\gamma^2\right] \mathbb{D}(X)$$

where $\mathbb{D}$ represents variance. Actually, we can borrow the theoretical results in [38]. We denote $p_m = 1-(1-CR)\frac{D-1}{D} = CR \cdot \frac{D-1}{D} + \frac{1}{D}$. The generation of u_i^t in smoDE can be re-written as:

$$u_i^t = (1-p_m)x_i^t + p_m v_i^t = (1-p_m)x_i^t + p_m x_{r_1^i}^t + p_m \cdot F(x_{r_2^i}^t - x_{r_3^i}^t)$$

Our model is the special case in [38] (Eq. (10)) by setting $\omega = 1, \gamma = (1-p_m), W = p_m \cdot F$ and we can obtain the population diversity of smoDE:

$$\begin{aligned} \mathbb{E}[\mathbb{D}(U)] &= \left[2p_m^2F^2 + \frac{NP-1}{NP}p_m^2 + (1-p_m)^2\right]\mathbb{D}(X) \\ &= K_{smoDE}\mathbb{D}(X) \end{aligned}$$

In [31], Zaharie analyzed classic DE (rand/1/bin):

$$\mathbb{E}[\mathbb{D}(U)] = \left[1 + 2p_mF^2 - \frac{p_m(2-p_m)}{NP}\right]\mathbb{D}(X) = K_{DE}\mathbb{D}(X)$$

We compare K_{smoDE} and K_{DE}:

$$\begin{aligned} K_{smoDE} \quad -K_{DE} &= 2p_m^2F^2 + \frac{NP-1}{NP}p_m^2 + (1-p_m)^2 \\ - \quad & (1 + 2p_mF^2 - \frac{p_m(2-p_m)}{NP}) \\ = \quad & 2F^2(p_m^2 - p_m) + (1-p_m)^2 \\ + \quad & \frac{(NP-2)p_m^2 + 2p_m - NP}{NP} \\ = \quad & (2F^2 + 2 - \frac{2}{NP})(p_m^2 - p_m) \le 0 \end{aligned}$$

The last inequality holds since $NP \ge 1$ and $0 < p_m \le 1$. The result shows that $K_{smoDE} \le K_{DE}$, which means smoDE can converge faster to a local minimum than DE since a smaller variance of population implies faster convergence.

5 EXPERIMENTS

In this section, we first show the performance of ada-smoDE by testing on the CEC 2018 test suite. Second, we visualize the gradient vector field of the loss function (Eq. (9)) on specific functions and validate its correctness. Third, we show that tuning parameters of smoDE by GBT can be more efficient than the Bayesian optimization algorithm (BOA). Forth, we validate the correctness of the analysis of population diversity. Finally, we compare the convergence rate of smoDE and DE.

5.1 Test on the CEC 2018

The CEC 2018 test suite contains 29 test functions for 10D, 30D, 50D and 100D. Here we test DE and smoDE on 10D and 30D test functions with $10000 \cdot D$ maximum fitness evaluations. For training smoDE, we choose a 10D quadratic function $f_1(x) = \sum_{i=1}^{D}(x_i - 1)^2$ as training function. The maximum training epoch is 50 and the learning rate is 0.01. Note that the smoDE can converge faster than classic DE, we can regard it as a local search method. Thus we do not use it separately, we use it to enhance the classic DE instead. Specifically, we first apply the classic DE to 90% of the computational budget and apply ada-smoDE to the remaining budget sequentially. The parameters for DE are as follows: $NP = 50, F = 0.5, CR = 0.9$. The parameters for ada-smoDE are as follows: $NP = 50$, F, CR are generated adaptively by the network trained by f_1.

Table 1 shows the results for 10D and 30D over 51 runs. Meanwhile, the Wilcoxon rank sum hypothesis test between classic DE and ada-smoDE at a significance level of 5% is also summarized in the table, where †, § means classic DE performs better/worse than ada-smoDE respectively, ≈ means the two algorithms have similar performance. From the table, we see that for 10D problems, ada-smoDE performs better than DE on four functions and worse on zero functions. For 30D problems, ada-smoDE performs better than DE on six functions and worse on four functions. *BM* summarizes the number of best mean achieved for one method. We can also see that ada-smoDE has larger *BM* values than DE for 10D and 30D test functions. Thus we may conclude that ada-smoDE can perform competitively or even a little bit better than classic DE.

5.2 Visualization of Gradient field

In this subsection, we show the visualization of the gradient field of parameters in smoDE. We divide [0,1] into small grids, where the grid spacing interval is 0.05 and each grid point represents an (F, CR) pair, namely $\{F_i, CR_j\}_{i,j=1}^{21}$. First, we estimate the contour of function value w.r.t F, CR. We calculate the best function value for the objective function $f_1(x) = \sum_{i=1}^{D}(x_i - 1)^2$ found by smoDE with each (F_i, CR_j) pair and maximum generation $T = 200$, population size $NP = 50$. We show the logarithm value of $f_1(x)$ in Figure 1 where these values construct an approximate contour of the logarithm function value w.r.t. F, CR. Second we calculate the **negative gradient vector** of loss function Eq. (9) at each (F_i, CR_j) pair with different learning period (LP) and show them in Figure 3(a,b). Meanwhile, we also calculate the **negative gradient vector** at each (F_i, CR_j) pair for $f_2 = \sum_{i=1}^{d-1}(100({x_i}^2 - x_{i+1})^2 + (x_i - 1)^2)$ which is shown in Figure 3(c,d).

From the figure, we see that even $LP = 1$, the gradient vector can approximately estimate the trend of the function value. However, when $LP = 1$, since we use "truncated gradient" technique so the gradient vector cannot be correct at every (F, CR) point. When $LP = 5$, the estimation is more correct since larger LP can provide more information.

5.3 Efficiency for tuning parameters

First, we use GBT for tuning parameters in smoDE on $f_1(x)$. Figure 4 shows the tuning process of F, CR on f_1. From the figure we see that, by using gradient information, we can find the optimal parameter fast.

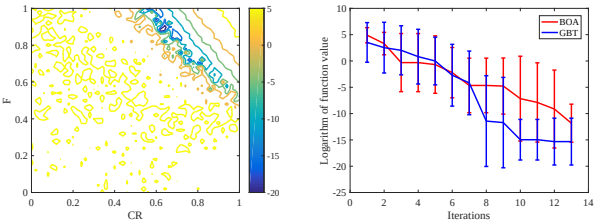
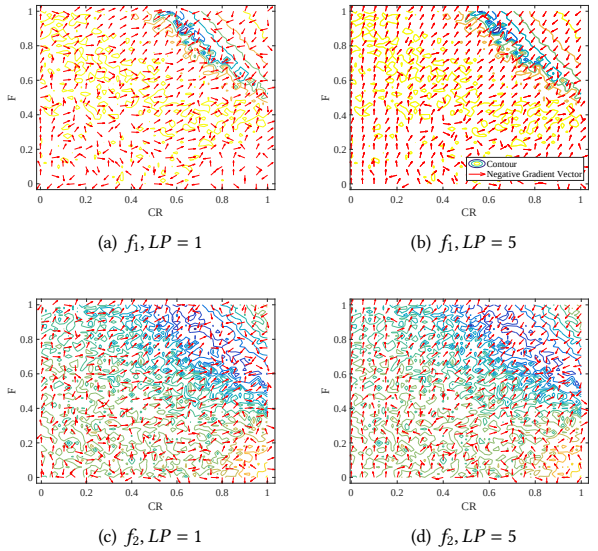


Figure 1: The contour of logarithm function value w.r.t. F, CR.

Figure 2: The tuning process on f_1 by BOA and gradient-based tuning (GBT).



(a) $f_1, LP = 1$ (b) $f_1, LP = 5$

(c) $f_2, LP = 1$ (d) $f_2, LP = 5$

Figure 3: Validation of Gradient Field.

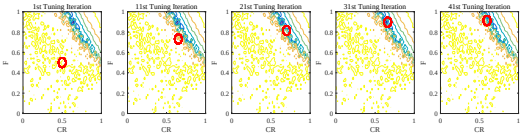
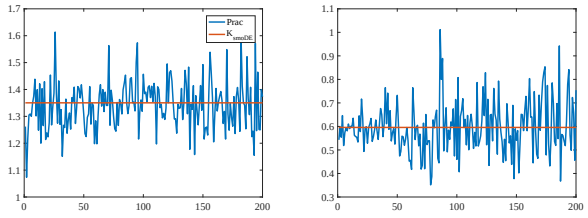


Figure 4: The tuning process on f_1, the center of red circle is (F, CR) pair.

Second, we compare GBT with a commonly used tuning tool BOA. For classic DE, grid search, random search and BOA, are mainly used as tuning methods, which need evaluation on objective function $f(x)$. However, grid search and random search need many evaluations on objective functions, which are brute-force methods and always cost much. Meanwhile, BOA needs to calculate the inverse matrix thus its time complexity is $O(n^3)$ where n is the sampling number. Here we compare the performance of BOA and GBT on tuning parameters for smoDE. The initial sampling points for BOA are randomly set, the kernel function is Gaussian kernel, the maximum sampling number is set as 13. Similarly, for GBT, the

Table 1: Experimental Results obtained by DE and Ada-smoDE on 10D and 30D test functions over 51 runs.

	10D				30D			
	DE		Ada-smoDE		DE		Ada-smoDE	
	mean	std	mean	std	mean	std	mean	std
F_1	**0.00e+00**$^{\approx}$	0.00e+00	**0.00e+00**	0.00e+00	**1.71e-04**†	1.91e-04	5.94e-04	3.60e-04
F_3	**0.00e+00**$^{\approx}$	0.00e+00	**0.00e+00**	0.00e+00	5.58e+03§	2.38e+03	**1.70e+03**	1.01e+03
F_4	2.05e+00§	9.77e-01	**1.33e+00**	1.21e+00	**5.86e+01**†	7.77e-01	5.86e+01	7.77e-01
F_5	6.87e+00$^{\approx}$	4.59e+00	**6.54e+00**	3.79e+00	1.85e+02$^{\approx}$	1.17e+01	**1.84e+02**	1.04e+01
F_6	**0.00e+00**$^{\approx}$	0.00e+00	**0.00e+00**	0.00e+00	**3.11e-04**†	1.11e-04	6.65e-04	2.43e-04
F_7	**2.02e+01**$^{\approx}$	6.50e+00	2.17e+01	7.29e+00	2.20e+02$^{\approx}$	1.06e+01	**2.18e+02**	1.20e+01
F_8	**6.16e+00**$^{\approx}$	3.35e+00	6.54e+00	4.07e+00	**1.88e+02**$^{\approx}$	9.50e+00	1.89e+02	8.07e+00
F_9	**0.00e+00**$^{\approx}$	0.00e+00	**0.00e+00**	0.00e+00	**0.00e+00**$^{\approx}$	0.00e+00	**0.00e+00**	0.00e+00
F_{10}	**3.09e+02**$^{\approx}$	1.81e+02	3.69e+02	2.37e+02	**6.69e+03**$^{\approx}$	3.29e+02	**6.69e+03**	3.62e+02
F_{11}	7.83e-01$^{\approx}$	9.35e-01	**7.34e-01**	9.42e-01	7.28e+01$^{\approx}$	2.11e+01	**7.25e+01**	2.28e+01
F_{12}	5.18e+01$^{\approx}$	7.47e+01	**3.32e+01**	5.58e+01	3.58e+03§	9.12e+02	**2.84e+03**	3.41e+02
F_{13}	**3.95e+00**$^{\approx}$	2.38e+00	4.29e+00	2.15e+00	8.86e+01$^{\approx}$	5.96e+00	**8.74e+01**	7.29e+00
F_{14}	**4.29e-01**$^{\approx}$	6.05e-01	4.68e-01	6.08e-01	6.60e+01$^{\approx}$	4.98e+00	**6.50e+01**	5.66e+00
F_{15}	**2.17e-01**$^{\approx}$	3.57e-01	3.01e-01	3.81e-01	4.65e+01§	4.94e+00	**4.47e+01**	5.09e+00
F_{16}	**4.79e+00**$^{\approx}$	6.46e+00	7.47e+00	1.45e+01	**1.13e+03**$^{\approx}$	1.94e+02	1.16e+03	1.78e+02
F_{17}	4.65e+00$^{\approx}$	8.27e+00	**3.39e+00**	7.28e+00	**2.11e+02**$^{\approx}$	1.34e+02	2.51e+02	1.48e+02
F_{18}	2.57e+00§	6.53e+00	**6.57e-02**	2.94e-01	4.30e+01$^{\approx}$	3.43e+00	**4.25e+01**	3.20e+00
F_{19}	1.01e-01$^{\approx}$	3.56e-01	**8.67e-02**	3.22e-01	**2.93e+01**$^{\approx}$	2.26e+00	2.97e+01	2.43e+00
F_{20}	**2.73e-01**$^{\approx}$	3.24e-01	2.92e-01	2.75e-01	**7.11e+01**†	4.14e+01	8.63e+01	5.52e+01
F_{21}	**1.75e+02**$^{\approx}$	5.17e+01	1.79e+02	4.94e+01	**3.74e+02**$^{\approx}$	1.10e+01	3.75e+02	1.01e+01
F_{22}	9.31e+01$^{\approx}$	2.49e+01	**8.71e+01**	3.36e+01	1.45e+03§	2.77e+03	**9.17e+02**	2.26e+03
F_{23}	3.07e+02§	3.01e+00	**3.06e+02**	3.65e+00	5.33e+02$^{\approx}$	7.59e+00	**5.32e+02**	9.57e+00
F_{24}	3.27e+02$^{\approx}$	4.67e+01	**3.26e+02**	4.62e+01	**5.99e+02**$^{\approx}$	1.00e+01	6.00e+02	1.04e+01
F_{25}	4.21e+02$^{\approx}$	2.35e+01	**4.18e+02**	2.32e+01	**3.86e+02**$^{\approx}$	1.81e-02	**3.86e+02**	1.91e-02
F_{26}	**3.01e+02**$^{\approx}$	9.24e+00	**3.01e+02**	9.24e+00	**2.77e+03**$^{\approx}$	1.19e+02	**2.77e+03**	1.15e+02
F_{27}	**3.89e+02**$^{\approx}$	1.17e+00	**3.89e+02**	1.26e+00	**4.90e+02**$^{\approx}$	8.96e+00	4.93e+02	6.54e+00
F_{28}	**3.75e+02**$^{\approx}$	1.14e+02	3.95e+02	1.26e+02	3.13e+02§	3.59e+01	**3.08e+02**	3.02e+01
F_{29}	**2.31e+02**$^{\approx}$	4.43e+00	**2.31e+02**	2.92e+00	**7.80e+02**$^{\approx}$	2.06e+02	7.94e+02	1.96e+02
F_{30}	1.12e+05§	2.83e+05	**1.65e+04**	1.14e+05	2.00e+03§	2.81e+01	**1.97e+03**	2.74e+01
BM	17		**19**		16		**17**	
†/≈ /§	0/26/4				4/20/6			



(a) $F = 0.7, CR = 0.8$ (b) $F = 0.4, CR = 0.5$

Figure 5: Validation of population diversity with different parameters.

initial F, CR are set randomly, learning rate for GBT is 0.05, $LP = 5$. To reduce the impact of randomness, we run GBT and BOA 5 times, and we show the mean and std. of function values of f_1 tuned by BOA and gradient-based tuning (GBT) in Figure 2. From the figure, we see that GBT can obtain better performance. Last but not least, the time cost for BOA is 45s, while the time for GBT is only 2.3s. It implies that GBT is an efficient tuning method.

5.4 Validation of Theoretical Analysis on Population Diversity

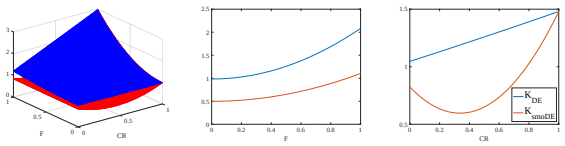
In this subsection, we validate the correctness of the theoretical analysis on population diversity. We optimize f_1 by using smoDE with different parameters and record X, U in each generation, $NP = 50$. We then calculate $prac^t = mean[\mathbb{D}(U^t) \oslash \mathbb{D}(X^t)]$ as the practical value, where $\oslash$ means element division. Figure 5 shows the $prac$ value and theorem value K_{smoDE}. From the figure, we see that the $prac$ is around K_{smoDE}. Meanwhile, we define $K_{prac} = \frac{1}{T}\sum_{t=1}^{T} prac^t$ where T is the maximum generation. Table 2 shows K_{prac} and K_{smoDE}. From the table, we see that for different parameters, K_{prac} is close to K_{smoDE}, which implies the correctness of the theoretical analysis on population diversity.

Table 2: The practical value and theoretical value for different parameters.

	$F = 0.7, CR = 0.8$	$F = 0.4, CR = 0.5$
K_{prac}	1.3515	0.5981
K_{smoDE}	1.3414	0.5958

5.5 Compare with DE

We prove that $K_{DE} \geq K_{smoDE}$. Now we visualize the K_{DE} and K_{smoDE} to show the correctness of our proof. Figure 6 shows the value of K_{DE} and K_{smoDE}. Figure 6(a) shows the 3-D landscape



(a) $NP = 50$ (b) $NP = 50, CR = 0.5$ (c) $NP = 50, F = 0.5$

Figure 6: The visualization of K_{DE} and K_{smoDE}.

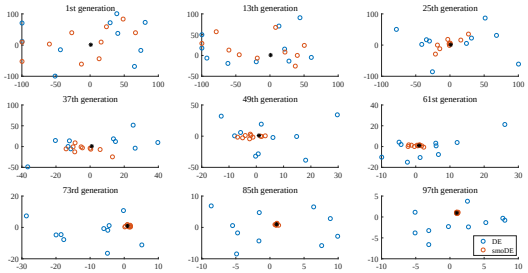


Figure 7: The convergence rate of DE and smoDE.

of K_{DE} and K_{smoDE}. To show clearly, we fix $CR = 0.5$ and vary the value of F. Figure 6(b) shows the curves of K_{DE} and K_{smoDE} with fixed $CR = 0.5$. Similarly, figure 6(c) shows the curves of K_{DE} and K_{smoDE} with fixed $F = 0.5$. We can see that for all F, CR, $K_{DE} \geq K_{smoDE}$ is held.

Then we study the convergence rate of DE and smoDE. We optimize 10D $f_1(x)$ with $NP = 50$, maximum generation $T = 200$. $F = 0.7, CR = 0.8$ are set for smoDE and DE. To show the convergence rate, we record the first 10 individuals in the population of each generation and plot the first two dimensions of these 10 individuals in Figure 7. In the figures, black $*$ is the global minimum. We can see that in the first 50 generations, DE and smoDE cannot find the global optimum and after 50 generations, smoDE converges to the global minimum faster than DE.

6 CONCLUSION

In this paper, we proposed smoDE by smoothing crossover and bound constraint. smoDE allows using the gradient method to tune the parameters. Then we embedded a neural network into smoDE (namely ada-smoDE) to generate parameters adaptively. We also analyzed the population diversity of smoDE and found that smoDE can converge faster than DE. In the experiment part, we tested ada-smoDE on CEC 2018 test suite. The results showed that ada-smoDE has a competitive performance compared with DE. Additionally, we showed the correctness of gradient calculation and the gradient-based tuning (GBT) is more efficient than the commonly used tool BOA. Finally, we validated the correctness of the theorem and we compared the convergence rate of smoDE and DE.

APPENDIX

The appendix contains two parts: The first part shows detailed information about the calculation of the gradient of the loss function (Eq. (9)). The second part summarizes the algorithm ada-smoDE and the training function for ada-smoDE.

Calculation of gradient

In this subsection, we will show the detailed calculation of the gradient. Here we consider the element-wise gradient: $Cu_{i,j}^{G-\alpha_i}$, $u_{i,j}^{G-\alpha_i}$, $v_{i,j}^{G-\alpha_i}$ represent the j-th component of $Cu_i^{G-\alpha_i}$, $u_i^{G-\alpha_i}$, $v_i^{G-\alpha_i}$ respectively.

For bound constraint Eq. (7),

$$\frac{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}}{\partial u_{i,j}^{G-\alpha_i}} = 1 + (B_{up} - u_{i,j}^{G-\alpha_i})\phi(B_{up} - u_{i,j}^{G-\alpha_i}) - (1 - \Phi(B_{up} - u_{i,j}^{G-\alpha_i}))$$

Similarly, for bound constraint Eq. (8) the gradient flow is :

$$\frac{\partial Cu_{i,j}^{G-\alpha_i}}{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}} = 1 - (B_{low} - \tilde{Cu}_{i,j}^{G-\alpha_i})\phi(\tilde{Cu}_{i,j}^{G-\alpha_i} - B_{low}) - (1 - \Phi(\tilde{Cu}_{i,j}^{G-\alpha_i} - B_{low}))$$

From Eq. (4),

$$\frac{\partial u_{i,j}^{G-\alpha_i}}{\partial CR} = -(x_{i,j}^{G-\alpha_i} - v_{i,j}^{G-\alpha_i}) \cdot \frac{D-1}{D}$$

$$\frac{\partial u_{i,j}^{G-\alpha_i}}{\partial v_{i,j}^{G-\alpha_i}} = 1 - (1 - CR) \cdot \frac{D-1}{D}$$

From Eq. (3),

$$\frac{\partial v_{i,j}^{G-\alpha_i}}{\partial F} = x_{r_2^i}^{G-\alpha_i} - x_{r_3^i}^{G-\alpha_i}$$

Since each component of x_i^G is generated by F, CR, thus the gradient should be the summary of the gradient of each component:

$$\nabla_{CR} L(F, CR) = \sum_{j=1}^{D} [\nabla f(x_i^G)_j \cdot \frac{\partial x_{i,j}^G}{\partial Cu_{i,j}^{G-\alpha_i}} \cdot \frac{\partial Cu_{i,j}^{G-\alpha_i}}{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}} \cdot \frac{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}}{\partial u_{i,j}^{G-\alpha_i}} \cdot \frac{\partial u_{i,j}^{G-\alpha_i}}{\partial CR}] \tag{11}$$

$$\nabla_F L(F, CR) = \sum_{j=1}^{D} [\nabla f(x_i^G)_j \cdot \frac{\partial x_{i,j}^G}{\partial Cu_{i,j}^{G-\alpha_i}} \cdot \frac{\partial Cu_{i,j}^{G-\alpha_i}}{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}} \cdot \frac{\partial \tilde{Cu}_{i,j}^{G-\alpha_i}}{\partial u_{i,j}^{G-\alpha_i}} \cdot \frac{\partial u_{i,j}^{G-\alpha_i}}{\partial v_{i,j}^{G-\alpha_i}} \cdot \frac{\partial v_{i,j}^{G-\alpha_i}}{\partial F}] \tag{12}$$

Ada-smoDE Algorithm

Alg. 1 summarizes ada-smoDE. The algorithm takes an objective function, maximum generation and parameter for the network as input. In line 3, the information $[mean(std(X)), mean(f(X))]$ are collected. The information is sent to network $\psi(\cdot, \Theta)$ and network outputs scaling factor F_t, crossover rate CR_t. Then mutation is implemented by using Eq. (3), crossover is implemented by using Eq. (4), bound constraint is implemented by using Eq. (7) and Eq. (8).

Alg. 2 shows the training algorithm for ada-smoDE. The algorithm takes an objective function, maximum generation, learning rate, training Epoch and initial network parameter as input. In

Algorithm 1: Adaptive Smooth Differential Evolution (ada-smoDE)

Input: An objective function f, the maximum number of generation T, and network parameter Θ

Initialize X^0 randomly; $t \leftarrow 0$;
while $t < T$ **do**
  $s_t \leftarrow [mean(std(X^t)), mean(f(X^t))]$;
  $[F_t, CR_t] \leftarrow \psi(s_t, \Theta)$;
  $V^t \leftarrow$**Mutation**(X^t, F_t);
  $U^t \leftarrow$**Crossover**(CR_t, V^t, X^t);
  $CU^t \leftarrow$**Bound**(U_t, B_{up}, B_{low});
  $X^{t+1} \leftarrow$**Selection**(CU^t, X^t, f);
  $t \leftarrow t + 1$;
end
return $X^T, minf(X^T)$

Algorithm 2: Training algorithm for ada-smoDE

Input: An objective function f, the maximum number of generation T, learning rate α, training Epoch $Epoch$, and initial network parameter Θ_0

$t \leftarrow 0$;
while $t < Epoch$ **do**
  $[X^*, f^*] \leftarrow ada - smoDE(f, T, \Theta_0)$;
  $\Theta_t \leftarrow \Theta_t - \alpha \nabla_\Theta f^*$;
end
return Θ_{Epoch}

line 3, we use ada-smoDE to generate the optimal solution and its function value f^*. In line 4, Θ is updated by calculating the gradient w.r.t. Θ. Here we talk about how to calculate $\nabla_\Theta f^*$: Note that line 4 aims to minimize f^* which is the same as Eq. (9). Different from Eq. (9), we now need to find optimal Θ but not F, CR. Notice that F, CR is generated by $\psi(\cdot, \Theta)$, thus we can obtain $\nabla_\Theta f^*$ by chain rule:

$$\nabla_\Theta f^* = \frac{\partial f^*}{\partial F} \times \frac{\partial F}{\partial \Theta} + \frac{\partial f^*}{\partial CR} \times \frac{\partial CR}{\partial \Theta}$$

where $\frac{\partial f^*}{\partial F}$ and $\frac{\partial f^*}{\partial CR}$ are obtained by Eq. (12) and Eq. (11) respectively. $\frac{\partial F}{\partial \Theta}$ and $\frac{\partial CR}{\partial \Theta}$ can be obtained by normal back-propagation (BP) calculation in neural networks.

REFERENCES

[1] Aldeida Aleti and Irene Moser. 2016. A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms. *Comput. Surveys* 49, 3 (2016), 1–35.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*. 3981–3989.

[3] J. Brest, M. S. Maučec, and B. Bošković. 2016. iL-SHADE: Improved L-SHADE algorithm for single objective real-parameter optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1188–1195.

[4] J. Brest, M. S. Maučec, and B. Bošković. 2017. Single objective realparameter optimization: Algorithm jSO. In *Proceedings of the Proceedings of the IEEE Congress on Evolutionary Computation*. 1311–1318.

[5] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. 2011. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* 15, 1 (2011), 4–31.

[6] Russell Eberhart and James Kennedy. 1995. A New Optimizer Using Particle Swarm Theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. 39–43.

[7] Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization. *arXiv: Machine Learning* (2018), arXiv:1807.02811.

[8] Yuxuan Fu, Yanmei Kang, and Ruonan Liu. 2020. Novel bearing fault diagnosis algorithm based on the method of moments for stochastic resonant systems. *IEEE Transactions on Instrumentation and Measurement* 70 (2020), 650061.

[9] John Galletly. 1996. Evolutionary Algorithms in Theory and Practice. *Complexity* 2, 8 (1996), 26–27.

[10] L. Gammaitoni, P. Hanggi, P. Jung, and F. Marchesoni. 1998. Stochastic resonance. *Reviews of Modern Physics* 70, 1 (1998), 223–287.

[11] A. A. Hadi, A. W. Mohamed, and K. M. Jambi. 2021. Single-Objective Real-Parameter Optimization: Enhanced LSHADE-SPACMA Algorithm. In *Heuristics for Optimization and Learning. Studies in Computational Intelligence*. Springer.

[12] N. Hansen, S. Muller, and P. Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation* 11, 1 (2003), 1–18.

[13] N. Hansen and A. Ostermeier. 2001. Completely derandomized self adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[14] S Hochreiter and J Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[15] Frank Hutter, Holger H Hoos, and Kevin Leytonbrown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. 507–523.

[16] A. K. Mohamed and A. W. Mohamed. 2019. Real-Parameter Unconstrained Optimization Based on Enhanced AGDE Algorithm. In *Machine Learning Paradigms: Theory and Application. Studies in Computational Intelligence*. Springer.

[17] A. W. Mohamed. 2017. Differential Evolution (DE): A Short Review. *Robotics and Automation Engineering Journal* 2, 1 (2017).

[18] Ali W. Mohamed, Anas A. Hadi, Anas M. Fattouh, and Kamal M. Jambi. 2017. LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems. In *IEEE Congress on Evolutionary Computation*. 145–152.

[19] A. W. Mohamed, A. A. Hadi, and K. M. Jambi. 2019. Novel Mutation Strategy for Enhancing SHADE and LSHADE Algorithms for Global Numerical Optimization. *Swarm and Evolutionary Computation* 50 (2019), 100455.

[20] A. W. Mohamed, A. A. Hadi, and A. K. Mohamed. 2021. Differential Evolution Mutations: Taxonomy, Comparison and Convergence Analysis. *IEEE Access* 9, 99 (2021).

[21] Ali Wagdy Mohamed, Anas A. Hadi, Ali Khater Mohamed, and Noor H. Awad. 2020. Evaluating the Performance of Adaptive GainingSharing Knowledge Based Algorithm on CEC 2020 Benchmark Problems. In *IEEE Congress on Evolutionary Computation*. 1–8.

[22] Kenneth V. Price. 1999. *An Introduction to Differential Evolution*. McGraw-Hill Ltd., UK, GBR, 79–108.

[23] Mudita Sharma, Alexandros Komninos, Manuel Lopezibanez, and Dimitar Kazakov. 2019. Deep reinforcement learning based parameter control in differential evolution. In *Genetic and Evolutionary Computation Conference*. 709–717.

[24] Jianyong Sun, Wei Zheng, Qingfu Zhang, and Zongben Xu. 2022. Graph Neural Network Encoding for Community Detection in Attribute Networks. *IEEE Transactions on Cybernetics* 52, 8 (2022), 7791–7804.

[25] R Sutton and A Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

[26] Ryoji Tanabe and Alex Fukunaga. 2013. Success-History Based Parameter Adaptation for Differential Evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 71–78.

[27] Ryoji Tanabe and Alex Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1658–1665.

[28] Ryoji Tanabe and Alex Fukunaga. 2020. Reviewing and Benchmarking Parameter Control Methods in Differential Evolution. *IEEE Transactions on Systems, Man, and Cybernetics* 50, 3 (2020), 1170–1184.

[29] Sander Van Rijn, Carola Doerr, and Thomas Back. 2018. Towards an Adaptive CMA-ES Configurator. In *Parallel Problem Solving from Nature*. 54–65.

[30] Daan Wierstra, Tom Schaul, Jan Peters, and Jurgen Schmidhuber. 2008. Natural Evolution Strategies. In *world congress on computational intelligence*. 3381–3387.

[31] D. Zaharie. 2002. Critical values for the control parameters of Differential Evolution algorithms. In *Proceedings of MENDEL*. 62–67.

[32] Zhihui Zhan, Jun Zhang, Yun Li, and Henry Shuhung Chung. 2009. Adaptive Particle Swarm Optimization. In *Systems Man and Cybernetics*, Vol. 39. 1362–1381.

[33] Geng Zhang and Yuhui Shi. 2018. Hybrid Sampling Evolution Strategy for Solving Single Objective Bound Constrained Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1–7.

[34] Haotian Zhang, Jialong Shi, Jianyong Sun, and Zongben Xu. 2022. Learning to Balance Exploration and Exploitation in Pareto Local Search for Multi-objective Combinatorial Optimization. In *Genetic and Evolutionary Computation Conference Companion*. 383–386.

[35] Haotian Zhang, Jianyong Sun, Thomas Bäck, Qingfu Zhang, and Zongben Xu. 2023. Controlling Sequential Hybrid Evolutionary Algorithm by Q-Learning. *IEEE Computational Intelligence Magazine* 18, 1 (2023), 84–103.

[36] Haotian Zhang, Jianyong Sun, Kay Chen Tan, and Zongben Xu. 2022. Learning Adaptive Differential Evolution by Natural Evolution Strategies. *IEEE Transactions on Emerging Topics in Computational Intelligence* (2022), 1–15.

[37] Haotian Zhang, Jianyong Sun, and Zongben Xu. 2020. Adaptive Structural Hyper-Parameter Configuration by Q-Learning. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1–8.

[38] Haotian Zhang, Jianyong Sun, and Zongben Xu. 2021. Learning to mutate for Differential Evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1–8. https://doi.org/10.1109/CEC45853.2021.9504990

[39] Jingqiao Zhang and Arthur C Sanderson. 2009. JADE: Adaptive Differential Evolution with Optimal External Archive. *IEEE Transactions On Evolutionary Computation* 13, 5 (2009), 945–458.