Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization

A. K. Qin, V. L. Huang, and P. N. Suganthan

***Abstract*—Differential evolution (DE) is an efficient and powerful population-based stochastic search technique for solving optimization problems over continuous space, which has been widely applied in many scientific and engineering fields. However, the success of DE in solving a specific problem crucially depends on appropriately choosing trial vector generation strategies and their associated control parameter values. Employing a trial-and-error scheme to search for the most suitable strategy and its associated parameter settings requires high computational costs. Moreover, at different stages of evolution, different strategies coupled with different parameter settings may be required in order to achieve the best performance. In this paper, we propose a self-adaptive DE (SaDE) algorithm, in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Consequently, a more suitable generation strategy along with its parameter settings can be determined adaptively to match different phases of the search process/evolution. The performance of the SaDE algorithm is extensively evaluated (using codes available from P. N. Suganthan) on a suite of 26 bound-constrained numerical optimization problems and compares favorably with the conventional DE and several state-of-the-art parameter adaptive DE variants.**

***Index Terms*—Differential evolution (DE), global numerical optimization, parameter adaptation, self-adaptation, strategy adaptation.**

I. Introduction

EVOLUTIONARY ALGORITHMs (EAs), inspired by the natural evolution of species, have been successfully applied to solve numerous optimization problems in diverse fields. However, when implementing the EAs, users not only need to determine the appropriate encoding schemes and evolutionary operators, but also need to choose the suitable parameter settings to ensure the success of the algorithm, which may lead to demanding computational costs due to the time-consuming trial-and-error parameter and operator tuning process. To overcome such inconvenience, researchers have actively investigated the adaptation of parameters and operators in EAs [1]–[3]. Different categorizations of parameter adaptation methods have been presented in [4]–[6]. Angeline [4] summarized two types of parameter updating rules in an adaptive EA, namely, absolute and empirical rules. Absolute updating rules usually prespecify how the parameter modifications would be made, while empirical updating rules adapt parameters according to the competition inherent in EAs. Literature [5] divided the parameter adaptation techniques into three categories: deterministic, adaptive, and self-adaptive control rules. Deterministic rules modify the parameters according to certain predetermined rationales without utilizing any feedback from the search process. Adaptive rules incorporate some form of the feedback from the search procedure to guide the parameter adaptation. Self-adaptive rules directly encode parameters into the individuals and evolve them together with the encoded solutions. Parameter values involved in individuals with better fitness values will survive, which fully utilize the feedback from the search process. Generally speaking, self-adaptive rules can also refer to those rules that mainly utilize the feedback from the search process such as fitness values to guide the updating of parameters.

The differential evolution (DE) algorithm, proposed by Storn and Price [7], is a simple yet powerful population-based stochastic search technique, which is an efficient and effective global optimizer in the continuous search domain. DE has been successfully applied in diverse fields such as mechanical engineering [13], [14], communication [11], and pattern recognition [10]. In DE, there exist many trial vector generation strategies out of which a few may be suitable for solving a particular problem. Moreover, three crucial control parameters involved in DE, i.e., population size NP, scaling factor F, and crossover rate (CR), may significantly influence the optimization performance of the DE. Therefore, to successfully solve a specific optimization problem at hand, it is generally required to perform a time-consuming trial-and-error search for the most appropriate strategy and to tune its associated parameter values. However, such a trial-and-error searching process requires high computational costs. Moreover, as evolution proceeds, the population of DE may move through different regions in the search space, within which certain strategies associated with specific parameter settings may be more effective than others. Therefore, it is desirable to adaptively determine an appropriate strategy and its associated parameter values at different stages of evolution/search process. In this paper, we propose a self-adaptive DE (SaDE) algorithm to avoid the expensive computational costs spent on searching for the most appropriate trial vector generation strategy as well as its associated parameter values by a trial-and-error procedure. Instead, both strategies and their associated parameters are gradually

Manuscript received January 16, 2007; revised July 20, 2007, March 24, 2008, and May 08, 2008. First published September 26, 2008; current version published April 01, 2009. This work was supported by the A*Star (Agency for Science, Technology and Research, Singapore) under Grant 052 101 0020.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore (e-mail: qinkai@pmail.ntu.edu.sg; huangling@pmail.ntu.edu.sg; epnsugan@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2008.927706

self-adapted by learning from their previous experiences in generating promising solutions. Consequently, a more suitable generation strategy along with its parameter settings can be determined adaptively to match different search/evolution phases. Specifically, at each generation, a set of trial vector generation strategies together with their associated parameter values will be separately assigned to different individuals in the current population according to the selection probabilities learned from the previous generations.

The remainder of this paper is organized as follows. The conventional DE and related work are reviewed in Sections II and III, respectively. Section IV describes the proposed SaDE. Experimental results demonstrating the performance of SaDE in comparison with the conventional DE and several state-of-the-art adaptive DE variants over a suite of 26 bound constrained numerical optimization problems are presented in Section V. Section VI concludes this paper.

II. DE Algorithm

DE algorithm aims at evolving a population of NP D-dimensional parameter vectors, so-called individuals, which encode the candidate solutions, i.e., $\mathbf{X}_{i,G} = \{x_{i,G}^1, \ldots, x_{i,G}^D\}, i = 1, \ldots, \mathrm{NP}$ towards the global optimum. The initial population should better cover the entire search space as much as possible by uniformly randomizing individuals within the search space constrained by the prescribed minimum and maximum parameter bounds $\mathbf{X}_{\min} = \{x_{\min}^1, \ldots, x_{\min}^D\}$ and $\mathbf{X}_{\max} = \{x_{\max}^1, \ldots, x_{\max}^D\}$. For example, the initial value of the jth parameter in the ith individual at the generation $G = 0$ is generated by

$$x_{i,0}^j = x_{\min}^j + \mathrm{rand}(0,1) \cdot \left(x_{\max}^j - x_{\min}^j\right), \qquad j = 1, 2, \ldots, D \tag{1}$$

where $\mathrm{rand}(0,1)$ represents a uniformly distributed random variable within the range $[0,1]$.

A. Mutation Operation

After initialization, DE employs the mutation operation to produce a mutant vector $\mathbf{V}_{i,G}$ with respect to each individual $\mathbf{X}_{i,G}$, so-called target vector, in the current population. For each target vector $\mathbf{X}_{i,G}$ at the generation G, its associated mutant vector $\mathbf{V}_{i,G} = \{v_{i,G}^1, v_{i,G}^2, \ldots, v_{i,G}^D\}$ can be generated via certain mutation strategy. For example, the five most frequently used mutation strategies implemented in the DE codes[1] are listed as follows:

1) "DE/rand/1":

$$\mathbf{V}_{i,G} = \mathbf{X}_{r_1^i,G} + F \cdot \left(\mathbf{X}_{r_2^i,G} - \mathbf{X}_{r_3^i,G}\right) \tag{2}$$

2) "DE/best/1":

$$\mathbf{V}_{i,G} = \mathbf{X}_{\mathrm{best},G} + F \cdot \left(\mathbf{X}_{r_1^i,G} - \mathbf{X}_{r_2^i,G}\right) \tag{3}$$

3) "DE/rand-to-best/1":

$$\mathbf{V}_{i,G} = \mathbf{X}_{i,G} + F \cdot (\mathbf{X}_{\mathrm{best},G} - \mathbf{X}_{i,G}) + F \cdot \left(\mathbf{X}_{r_1^i,G} - \mathbf{X}_{r_2^i,G}\right) \tag{4}$$

4) "DE/best/2":

$$\mathbf{V}_{i,G} = \mathbf{X}_{\mathrm{best},G} + F \cdot \left(\mathbf{X}_{r_1^i,G} - \mathbf{X}_{r_2^i,G}\right) + F \cdot \left(\mathbf{X}_{r_3^i,G} - \mathbf{X}_{r_4^i,G}\right) \tag{5}$$

5) "DE/rand/2":

$$\mathbf{V}_{i,G} = \mathbf{X}_{r_1^i,G} + F \cdot \left(\mathbf{X}_{r_2^i,G} - \mathbf{X}_{r_3^i,G}\right) + F \cdot \left(\mathbf{X}_{r_4^i,G} - \mathbf{X}_{r_5^i,G}\right). \tag{6}$$

The indices $r_1^i, r_2^i, r_3^i, r_4^i, r_5^i$ are mutually exclusive integers randomly generated within the range $[1, \mathrm{NP}]$, which are also different from the index i. These indices are randomly generated once for each mutant vector. The scaling factor F is a positive control parameter for scaling the difference vector. $\mathbf{X}_{\mathrm{best},G}$ is the best individual vector with the best fitness value in the population at generation G.

B. Crossover Operation

After the mutation phase, crossover operation is applied to each pair of the target vector $\mathbf{X}_{i,G}$ and its corresponding mutant vector $\mathbf{V}_{i,G}$ to generate a trial vector: $\mathbf{U}_{i,G} = (u_{i,G}^1, u_{i,G}^2, \ldots, u_{i,G}^D)$. In the basic version, DE employs the binomial (uniform) crossover defined as follows:

$$u_{i,G}^j = \begin{cases} v_{i,G}^j, & \text{if } (\mathrm{rand}_j[0,1) \le \mathrm{CR}) \text{ or } (j = j_{\mathrm{rand}}) \\ x_{i,G}^j, & \text{otherwise} \end{cases}, \qquad j = 1, 2, \ldots, D. \tag{7}$$

In (7), the crossover rate CR is a user-specified constant within the range $[0,1)$, which controls the fraction of parameter values copied from the mutant vector. j_{rand} is a randomly chosen integer in the range $[1, D]$. The binomial crossover operator copies the jth parameter of the mutant vector $\mathbf{V}_{i,G}$ to the corresponding element in the trial vector $\mathbf{U}_{i,G}$ if $\mathrm{rand}_j[0,1) \le \mathrm{CR}$ or $j = j_{\mathrm{rand}}$. Otherwise, it is copied from the corresponding target vector $\mathbf{X}_{i,G}$. There exists another exponential crossover operator, in which the parameters of trial vector $\mathbf{U}_{i,G}$ are inherited from the corresponding mutant vector $\mathbf{V}_{i,G}$ starting from a randomly chosen parameter index till the first time $\mathrm{rand}_j[0,1) > \mathrm{CR}$. The remaining parameters of the trial vector $\mathbf{U}_{i,G}$ are copied from the corresponding target vector $\mathbf{X}_{i,G}$. The condition $j = j_{\mathrm{rand}}$ is introduced to ensure that the trial vector $\mathbf{U}_{i,G}$ will differ from its corresponding target vector $\mathbf{X}_{i,G}$ by at least one parameter. DE's exponential crossover operator is functionally equivalent to the circular two-point crossover operator.

C. Selection Operation

If the values of some parameters of a newly generated trial vector exceed the corresponding upper and lower bounds, we randomly and uniformly reinitialize them within the prespecified range. Then, the objective function values of all trial vectors are evaluated. After that, a selection operation is performed. The objective function value of each trial vector $f(\mathbf{U}_{i,G})$ is compared to that of its corresponding target vector $f(\mathbf{X}_{i,G})$ in the current population. If the trial vector has less or equal objective function value than the corresponding target vector, the trial vector will replace the target vector and enter the population of the next generation. Otherwise, the target vector will remain in

[1]Publicly available online at http://www.icsi.berkeley.edu/~storn/code.html

TABLE I
ALGORITHMIC DESCRIPTION OF DE

```
Step 1 Set the generation number $G = 0$, and randomly initialize a population of NP individuals
       $\mathbf{P}_G = \{\mathbf{X}_{1,G},...,\mathbf{X}_{NP,G}\}$ with $\mathbf{X}_{i,G} = \{x_{i,G}^1, ..., x_{i,G}^D\}$, $i = 1,..., NP$ uniformly distributed in the range
       $[\mathbf{X}_{\min}, \mathbf{X}_{\max}]$, where $\mathbf{X}_{\min} = \{x_{\min}^1,..., x_{\min}^D\}$ and $\mathbf{X}_{\max} = \{x_{\max}^1,..., x_{\max}^D\}$
Step 2 WHILE stopping criterion is not satisfied
       DO
       Step 2.1 Mutation step
            /*Generate a mutated vector $\mathbf{V}_{i,G} = \{v_{i,G}^1, ..., v_{i,G}^D\}$ for each target vector $\mathbf{X}_{i,G}$ */
          FOR $i$ = 1 to NP
              Generate a mutated vector $\mathbf{V}_{i,G} = \{v_{i,G}^1, ..., v_{i,G}^D\}$ corresponding to the target vector $\mathbf{X}_{i,G}$
            via one of the equations (2)-(6).
          END FOR
       Step 2.2 Crossover step
         /*Generate a trial vector $\mathbf{U}_{i,G} = \{u_{i,G}^1, ..., u_{i,G}^D\}$ for each target vector $\mathbf{X}_{i,G}$ */
         a) Binomial crossover
         FOR $i$ = 1 to NP
            $j_{rand} = \lfloor rand[0,1) * D \rfloor$
            FOR $j$ = 1 to $D$
               $u_{i,G}^j = \begin{cases} v_{i,G}^j, & \text{if } (rand[0,1) \le CR) \text{ or } (j = j_{rand}) \\ x_{i,G}^j, & \text{otherwise} \end{cases}$
            END FOR
         END FOR
         b) Exponential crossover
         FOR $i$ = 1 to NP
            $j = \lfloor rand[0,1) * D \rfloor$, $L=0$
            $U_{i,G} = X_{i,G}$
            DO
               $u_{i,G}^j = v_{i,G}^j$
               $j = \langle j+1 \rangle_D$ *
               $L=L+1$
            WHILE ( $rand[0,1)<CR$ & $L<D$ )
          END FOR
       Step 2.3 Selection step
          /*Selection*/
          FOR $i$ =1 to NP
            Evaluate the trial vector $\mathbf{U}_{i,G}$
            IF $f(\mathbf{U}_{i,G}) \le f(\mathbf{X}_{i,G})$, THEN $\mathbf{X}_{i,G+1} = \mathbf{U}_{i,G}$, $f(\mathbf{X}_{i,G+1}) = f(\mathbf{U}_{i,G})$
               IF $f(\mathbf{U}_{i,G}) < f(\mathbf{X}_{best,G})$, THEN $\mathbf{X}_{best,G} = \mathbf{U}_{i,G}$, $f(\mathbf{X}_{best,G}) = f(\mathbf{U}_{i,G})$
               END IF

            END IF
          END FOR
       Step 2.4 Increment the generation count $G = G + 1$
Step 3 END WHILE
```

* Acute brackets $\langle\ \rangle_D$ denote modulo function with modulus D.

the population for the next generation. The selection operation can be expressed as follows:

$$\mathbf{X}_{i,G+1} = \begin{cases} \mathbf{U}_{i,G}, & \text{if } f(\mathbf{U}_{i,G}) \le f(\mathbf{X}_{i,G}) \\ \mathbf{X}_{i,G}, & \text{otherwise.} \end{cases} \quad (13)$$

The above 3 steps are repeated generation after generation until some specific termination criteria are satisfied. The algorithmic description of DE is summarized in Table I.

III. Previous Work Related to DE

The performance of the conventional DE algorithm highly depends on the chosen trial vector generation strategy and associated parameter values used. Inappropriate choice of strategies and parameters may lead to premature convergence or stagnation, which have been extensively demonstrated in [8], [16], [17], [24], and [29]. In the past decade, DE researchers have suggested many empirical guidelines for choosing trial vector generation strategies and their associated control parameter settings. Storn and Price [15] suggested that a reasonable value for NP should be between $5D$ and $10D$, and a good initial choice of F was 0.5. The effective range of F values was suggested between 0.4 and 1. The first reasonable attempt of choosing CR value can be 0.1. However, because the large CR value can speed up convergence, the value of 0.9 for CR may also be a good initial choice if the problem is near unimodal or fast convergence is desired. Moreover, if the population converges prematurely, either F or NP can be increased.

It was recommended in [20] to use the trial vector generation strategy DE/current-to-rand/1 and parameter setting $\mathrm{NP} = 20D, K = 0.5, F = 0.8$. If the DE converges prematurely, one should increase the value of NP and F or decrease the value of K. If the population stagnates, one should increase the value of NP or F, or randomly choose K within the range $[0, 1]$. If none of the above configuration works, one may try the strategy DE/rand/1/bin along with a small {\rm CR} value. Gämperle *et al.* [17] examined different parameter settings for DE on Sphere, Rosenbrock, and Rastrigin functions. Their experimental results showed that the searching capability and convergence speed are very sensitive to the choice of control parameters NP, F, and CR. They recommended that the population size NP be between $3D$ and $8D$, the scaling factor F equal 0.6, and the crossover rate CR be between $[0.3, 0.9]$.

Recently, Rönkkönen *et al.* [21] suggested using F values between $[0.4, 0.95]$ with $F = 0.9$ being a good initial choice. The CR values should lie in $[0, 0.2]$ when the function is separable while in $[0.9, 1]$ when the function's parameters are dependent. However, when solving a real engineering problem, the characteristics of the problem are usually unknown. Consequently, it is difficult to choose the most appropriate CR value in advance.

In DE literatures, various conflicting conclusions have been drawn with regard to the rules for manually choosing the strategy and control parameters, which undesirably confuse scientist and engineers who are about to utilize DE to solve scientific and engineering problems. In fact, most of these conclusions lack sufficient justifications as their validity is possibly restricted to the problems, strategies, and parameter values considered in the investigations.

Therefore, researchers have developed some techniques to avoid manual tuning of the control parameters. For example, Das *et al.* [29] linearly reduced the scaling factor F with increasing generation count from a maximum to a minimum value, or randomly varied F in the range $(0.5, 1)$. They also employed a uniform distribution between 0.5 and 1.5 (with a mean value of 1) to obtain a new hybrid DE variant [30]. In addition, several researchers [18], [24]–[26] focused on the adaptation of the control parameters F and CR. Liu and Lampinen introduced fuzzy adaptive differential evolution (FADE) using fuzzy logic controllers whose inputs incorporate the relative function values and individuals of successive generations to adapt the parameters for the mutation and crossover operations [18]. Based on the experimental results on test functions, the FADE algorithm outperformed the conventional DE on higher dimensional problems. Zaharie proposed a parameter adaptation for DE (ADE) based on the idea of controlling the population diversity, and implemented a multipopulation approach [24]. Following the same ideas, Zaharie and Petcu designed an adaptive Pareto DE algorithm for multiobjective optimization and analyzed its parallel implementation [25]. Abbass [26] self-adapted the crossover rate of DE for multiobjective optimization problems, by encoding the crossover rate into each individual, to simultaneously evolve with other parameter. The scaling factor F is generated for each variable from a Gaussian distribution $N(0, 1)$.

Since our preliminary self-adaptive DE work presented in [19], some new research papers focusing on the adaptation of control parameters in DE were published. Omran *et al.* [27] introduced a self-adapted scaling factor parameter F analogous to the adaptation of crossover rate CR in [26]. The crossover rate CR in [27] is generated for each individual from a normal distribution $N(0.5, 0.15)$. This approach (called SDE) was tested on four benchmark functions and performed better than other versions of DE. Besides adapting the control parameters F or CR, Teo proposed differential evolution with self adapting populations (DESAP) [22], based on Abbass's self-adaptive Pareto DE. Recently, Brest *et al.* [28] encoded control parameters F and CR into the individuals and adjusted by introducing two new parameters τ_1 and τ_2. In their algorithm (called jDE), a set of F values were assigned to individuals in the population. Then, a random number rand was uniformly generated in the range of $[0, 1]$. If $\mathrm{rand} < \tau_1$, the F was reinitialized to a new random value in the range of $[0.1, 1.0]$, otherwise it was kept unchanged. The CR was adapted in the same manner but with a different reinitialization range of $[0, 1]$. Brest *et al.* further compared the performance of several self-adaptive and adaptive DE algorithms in [42]. Recently, the self-adaptive neighborhood search DE algorithm was adopted into a novel cooperative coevolution framework [39]. Moreover, researchers improved the performance of DE by implementing opposition-based learning [40] or local search [41].

IV. SaDE Algorithm

To achieve the most satisfactory optimization performance by applying the conventional DE to a given problem, it is common to perform a trial-and-error search for the most appropriate trial vector generation strategy and fine-tune its associated control parameter values, i.e., the values of CR, F, and NP. Obviously, it may expend a huge amount of computational costs. Moreover, during different stages of evolution, different trial vector generation strategies coupled with specific control parameter values can be more effective than others. Motivated by these observations, we develop a SaDE algorithm, in which both trial vector generation strategies and their associated control parameter values can be gradually self-adapted according to their previous experiences of generating promising solutions.

The core idea behind the proposed SaDE algorithm is elucidated as follows.

A. Trial Vector Generation Strategy Adaptation

DE realizations employing different trial vector generation strategies usually performs differently when solving different optimization problems. Instead of employing the computationally expensive trial-and-error search for the most suitable strategy and its associated parameter values, we maintain a strategy candidate pool including several effective trial vector generation strategies with effective yet diverse characteristics. During evolution, with respect to each target vector in the current population, one strategy will be chosen from the candidate pool according to a probability learned from its previous experience of generating promising solutions and applied to perform the mutation operation. The more successfully one strategy behaved in previous generations to generate promising solutions, the more probably it will be chosen in the current generation to generate solutions. In the following, we investigate several effective trial vector generation strategies commonly referred to in DE literatures and choose some of them to construct the strategy candidate pool.

- Strategies relying on the best solution found so far such as "DE/rand-to-best/1/bin," "DE/best/1/bin," and "DE/best/2/bin," usually have the fast convergence speed and perform well when solving unimodal problems. However, they are more likely to get stuck at a local optimum and thereby lead to a premature convergence when solving multimodal problems.
- The "DE/rand/1/bin" strategy usually demonstrates slow convergence speed and bears stronger exploration capability. Therefore, it is usually more suitable for solving multimodal problems than the strategies relying on the best solution found so far.
- The "DE/best/1/bin" strategy is a degenerated case of the "DE/rand-to-best/1/bin" strategy with F equal to 1.
- Two-difference-vectors-based strategies may result in better perturbation than one-difference-vector-based strategies. Storn [11] claimed that according to the central limit theorem, the random variation of the summation of difference vectors of all target vector pairs in the current population was shifted slightly towards the Gaussian direction, which is the most commonly used mutation operator in EAs. The advantage of using two-difference-vectors-based strategies was also discussed in [35] in the particle swarm optimization (PSO) context, which empirically demonstrated that the statistical distribution of the summation of all one-difference vectors had a triangle shape, while the statistical distribution of the summation of all two-difference vectors had a bell shape that was generally regarded as a better perturbation mode.
- DE/current-to-rand/1 is a rotation-invariant strategy. Its effectiveness has been verified when it was applied to solve multiobjective optimization problems [34].

Our preliminary study in [19] only included two trial vector generation strategies into the strategy candidate pool, i.e., "DE/rand/1/bin" and "DE/rand-to-best/2/bin," which were frequently employed in many DE literatures. We incorporate two additional strategies: "DE/rand/2/bin" and "DE/current-to-rand/1" into the pool. The former strategy can have a better exploration capability due to the Gaussian-like perturbation while the latter one enables the algorithm to solve rotated problems more effectively. The four trial vector generation strategies constituting the strategy candidate pool in the proposed SaDE algorithm are listed as follows. The binomial-type crossover operator is utilized in the first three strategies due to its popularity in many DE literatures [7], [8], as shown in the equation at the bottom of the page.

Generally speaking, a good candidate pool should be restrictive so that the unfavorable influences of less effective strategies can be suppressed. Moreover, a set of effective strategies contained in a good candidate pool should have diverse characteristics, that is, the used strategies should demonstrate distinct capabilities when dealing with a specific problem at different stages of evolution. The theoretical study on the choice of the optimal pool size and the selection of strategies used in the pool are attractive research issues and deserve further investigations.

In the SaDE algorithm, with respect to each target vector in the current population, one trial vector generation strategy is selected from the candidate pool according to the probability learned from its success rate in generating improved solutions within a certain number of previous generations. The selected strategy is subsequently applied to the corresponding target vector to generate a trial vector. More specifically, at each generation, the probabilities of choosing each strategy in the candidate pool are summed to 1. These probabilities are

DE/rand/1/bin:

$$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot \left(x_{r_2,j} - x_{r_3,j}\right), & \text{if rand}\,[0,1) < \text{CR or } j = j_{\text{rand}} \\ x_{i,j}, & \text{otherwise} \end{cases}$$

DE/rand-to-best/2/bin:

$$u_{i,j} = \begin{cases} x_{i,j} + F \cdot (x_{\text{best},j} - x_{i,j}) + F \cdot \left(x_{r_1,j} - x_{r_2,j}\right) + F \cdot \left(x_{r_3,j} - x_{r_4,j}\right), & \text{if rand}\,[0,1) < \text{CR or } j = j_{\text{rand}} \\ x_{i,j}, & \text{otherwise} \end{cases}$$

DE/rand/2/bin:

$$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot \left(x_{r_2,j} - x_{r_3,j}\right) + F \cdot \left(x_{r_4,j} - x_{r_5,j}\right), & \text{if rand}\,[0,1) < \text{CR or } j = j_{\text{rand}} \\ x_{i,j}, & \text{otherwise} \end{cases}$$

DE/current-to-rand/1:

$$\mathbf{U}_{i,G} = \mathbf{X}_{i,G} + K \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{i,G}) + F \cdot \left(\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G}\right).$$

Success Memory

Index	Strategy 1	Strategy 2	$\cdots$	Strategy K
1	$ns_{1,G-LP}$	$ns_{2,G-LP}$	$\cdots$	$ns_{k,G-LP}$
2	$ns_{1,G-LP+1}$	$ns_{2,G-LP+1}$	$\cdots$	$ns_{k,G-LP+1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
LP	$ns_{1,G-1}$	$ns_{2,G-1}$	...	$ns_{k,G-1}$

Failure Memory

Index	Strategy 1	Strategy 2	$\cdots$	Strategy K
1	$nf_{1,G-LP}$	$nf_{2,G-LP}$	$\cdots$	$nf_{k,G-LP}$
2	$nf_{1,G-LP+1}$	$nf_{2,G-LP+1}$	$\cdots$	$nf_{k,G-LP+1}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
LP	$nf_{1,G-1}$	$nf_{2,G-1}$	...	$nf_{k,G-1}$

Fig. 1. Success memory and failure memory.

gradually adapted during evolution in the following manner. Assume that the probability of applying the kth strategy in the candidate pool to a target vector in the current population is $p_k, k = 1, 2, \ldots, K$, where K is the total number of strategies contained in the pool. The probabilities with respect to each strategy are initialized as $1/K$, i.e., all strategies have the equal probability to be chosen. We use the stochastic universal selection method [31] to select one trial vector generation strategy for each target vector in the current population. At the generation G, after evaluating all the generated trial vectors, the number of trial vectors generated by the kth strategy that can successfully enter the next generation is recorded as $\mathrm{ns}_{k,G}$ while the number of trial vectors generated by the kth strategy that are discarded in the next generation is recorded as $\mathrm{nf}_{k,G}$. We introduce *success and failure memories* to store these numbers within a fixed number of previous generations hereby named learning period (LP). As illustrated in Fig. 1, at the generation G, the number of trial vectors generated by different strategies that can enter or fail to enter the next generation over the previous LP generations are stored in different columns of the *success and failure memories*. Once the memories overflow after LP generations, the earliest records stored in the memories, i.e., $\mathrm{ns}_{G-\mathrm{LP}}$ or $\mathrm{nf}_{G-\mathrm{LP}}$ will be removed so that those numbers calculated in the current generation can be stored in the memories, as shown in Fig. 2.

After the initial LP generations, the probabilities of choosing different strategies will be updated at each subsequent generation based on the *success and failure memories*. For example, at the generation G, the probability of choosing the kth $(k = 1, 2, \ldots, K)$ strategy is updated by

$$p_{k,G} = \frac{S_{k,G}}{\sum_{k=1}^{K} S_{k,G}}$$

where

$$S_{k,G} = \frac{\sum_{g=G-\mathrm{LP}}^{G-1} ns_{k,g}}{\sum_{g=G-\mathrm{LP}}^{G-1} ns_{k,g} + \sum_{g=G-\mathrm{LP}}^{G-1} nf_{k,g}} + \varepsilon, \quad (k = 1, 2, \ldots, K; G > \mathrm{LP}) \qquad (14)$$

where $S_{k,G}$ represents the success rate of the trial vectors generated by the kth strategy and successfully entering the next generation within the previous LP generations with respect to generation G. The small constant value $\varepsilon = 0.01$ is used to avoid the possible null success rates. To ensure that the probabilities of choosing strategies are always summed to 1, we further divide $S_{k,G}$ by $\sum_{k=1}^{K} S_k$ to calculate $p_{k,G}$. Obviously, the larger the success rate for the kth strategy within the previous LP generations is, the larger the probability of applying it to generate the trial vectors at the current generation is.

B. Parameter Adaptation

In the conventional DE, the choice of numerical values for the three control parameters F, CR, and NP highly depends on the problem under consideration. Some empirical guidelines for choosing reasonable parameter settings have been discussed in Section III. In the proposed SaDE algorithm, we leave NP as a user-specified parameter because it highly replies on the complexity of a given problem. In fact, the population size NP does not need to be fine-tuned and just a few typical values can be tried according to the preestimated complexity of the given problem. Between other two parameters, CR is usually more sensitive to problems with different characteristics, e.g., the unimodality and multimodality, while F is closely related to the convergence speed. In our SaDE algorithm, the parameter F is approximated by a normal distribution with mean value 0.5 and standard deviation 0.3, denoted by $N(0.5, 0.3)$. A set of F values are randomly sampled from such normal distribution and applied to each target vector in the current population. It is easy to verify that values of F must fall into the range $[-0.4, 1.4]$ with the probability of 0.997. By doing so, we attempt to maintain both exploitation (with small F values) and exploration (with large F values) power throughout the entire evolution process. Following suggestions in [20], the control parameter K in the strategy "DE/current-to-rand/1" is hereby randomly generated within $[0, 1]$ so as to eliminate one additional parameter.

As demonstrated by a suite of extensive experiments in [8], the proper choice of CR can lead to successful optimization performance while a wrong choice may deteriorate the performance. In fact, good values of CR generally fall into a small range for a given problem, with which the algorithm can perform consistently well. Therefore, we consider gradually adjusting the range of CR values for a given problem according to previous CR values that have generated trial vectors successfully entering the next generation. Specifically, we assume that CR obeys a normal distribution with mean value CRm and standard deviation $\mathrm{Std} = 0.1$, denoted by $N(\mathrm{CRm}, \mathrm{Std})$ where CRm is initialized as 0.5. The Std should be set as a small value to guarantee that most CR values generated by N $(\mathrm{CRm}, \mathrm{Std})$ are between $[0, 1]$, even when CRm is near 0 or 1. Hence, the value of Std is set as 0.1. Our experiments showed that minor changes to the Std of the Gaussian distribution do not influence the performance of SaDE significantly.

In our preliminary work in [19], the same CRm value was used for all the trial vector generation strategies. However, it is possible that different strategies can perform well by using different ranges of CR values. Hence, it is reasonable to adapt the value of CRm with respect to each trial vector generation strategy. Without loss of generality, with respect to the kth

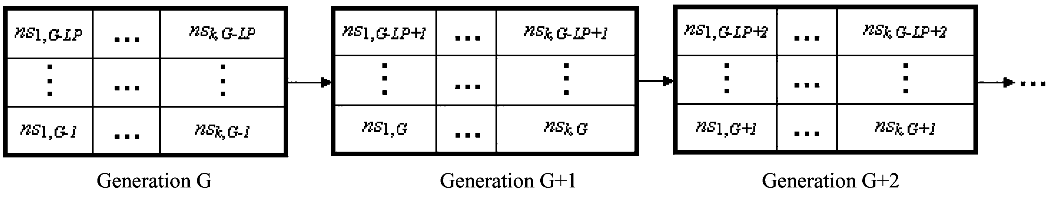


Fig. 2. Progress of success memory.

strategy, the value of CRm_k is initialized to 0.5. A set of CR values are randomly generated according to $N(\mathrm{CRm}_k, 0.1)$ and then applied to those target vectors to which the kth strategy is assigned. To adapt the crossover rate CR, we establish memories named $\mathrm{CRMemory}_k$ to store those CR values with respect to the kth strategy that have generated trial vectors successfully entering the next generation within the previous LP generations. Specifically, during the first LP generations, CR values with respect to kth strategy are generated by N $(\mathrm{CRm}_k, 0.1)$. At each generation after LP generations, the median value stored in $\mathrm{CRMemory}_k$ will be calculated to overwrite CRm_k. Then, CR values can be generated according to N $(\mathrm{CRm}_k, 0.1)$ when applying the kth strategy. After evaluating the newly generated trial vectors, CR values in $\mathrm{CRMemory}_k$ that correspond to earlier generations will be replaced by promising CR values obtained at the current generation with respect to the kth strategy.

By incorporating the aforementioned trial vector generation strategy and control parameter adaptation schemes into the conventional DE framework, a SaDE algorithm is developed. In the SaDE algorithm, both trial vector generation strategies and their associated parameter values are gradually self-adapted by learning their previous experiences of generating promising solutions. Consequently, a more suitable strategy along with its parameter setting can be determined adaptively to suit different phases of the search process. Extensive experiments described in Section IV verify the promising performance of the SaDE to handle problems with distinct properties such as unimodality and multimodality. The algorithmic description of the SaDE is presented in Table II.

TABLE II
ALGORITHMIC DESCRIPTION OF SADE

```
Step 1  Set the generation counter $G = 0$, and randomly initialize a population of NP individuals
        $\mathbf{P}_G = \{\mathbf{X}_{1,G}, \ldots, \mathbf{X}_{NP,G}\}$ with $\mathbf{X}_{i,G} = \{x_{i,G}^1, \ldots, x_{i,G}^D\}$, $i = 1, \ldots, NP$ uniformly distributed in the range
        $[\mathbf{X}_{\min}, \mathbf{X}_{\max}]$, where $\mathbf{X}_{\min} = \{x_{\min}^1, \ldots, x_{\min}^D\}$ and $\mathbf{X}_{\max} = \{x_{\max}^1, \ldots, x_{\max}^D\}$. Initialize the median value
        of CR ($CRm_k$), strategy probability ($p_{k,G}$, $k = 1, \ldots, K$, $K$ is the no. of available strategies), learning
        period (LP)
Step 2  Evaluate the population
Step 3  WHILE stopping criterion is not satisfied
        DO
        Step 3.1  Calculate strategy probability $p_{k,G}$ and update the Success and Failure Memory
                  IF $G>LP$
                      For $k=1$ to $K$
                          Update the $p_{k,G}$ by equation (14)
                          Remove $ns_{k,G-LP}$ and $nf_{k,G-LP}$ out of the Success and Failure Memory
                      respectively.
                      END
                  END
        Step 3.2  Assign trial vector generation strategy and parameter to each target vector $\mathbf{X}_{i,G}$
                  /* Assign trial vector generation strategy */
                  Using stochastic universal sampling to select one strategy k for each target vector $\mathbf{X}_{i,G}$

                  /* Assign control parameter F */
                  FOR $i=1$ to NP
                      $F_i = Normrnd(0.5, 0.3)$
                  END FOR

                  /* Assign control parameter CR */
                  IF $G>=LP$
                      FOR $k = 1$ to $K$, $CRm_k = median(CRMemory_k)$, END FOR
                  END IF
                  FOR $k = 1$ to $K$
                      FOR $i=1$ to NP
                          $CR_{k,i} = Normrnd(CRm_k, 0.1)$
                          WHILE $CR_{m,i} < 0$ or $CR_{m,i} > 1$
                          $CR_{k,i} = Normrnd(CRm_k, 0.1)$
                          END
                      END FOR
                  END FOR

        Step 3.3  Generate a new population where each trial vector $\mathbf{U}_{i,G}^k$ is generated according to
        associated trial vector generation strategy k and parameters $F_i$ and $CR_{k,i}$ in Step 3.2.
        Step 3.4  Randomly reinitialize the trial vector $\mathbf{U}_{i,G}^k$ within the search space if any variable is
        outside its boundaries.
        Step 3.5  Selection:
                  FOR $i=1$ to NP
                      Evaluate the trial vector $\mathbf{U}_{i,G}^k$
                      IF $f(U_{i,G}^k) \le f(X_{i,G})$
                          $X_{i,G+1} = U_{i,G}^k$, $f(X_{i,G+1}) = f(U_{i,G}^k)$
                          $ns_{k,G} = ns_{k,G} + 1$
                          Store $CR_{k,i}$ into $CRMemory_k$
                          IF $f(U_{i,G}) < f(X_{best,G})$
                              $X_{best,G} = U_{i,G}$, $f(X_{best,G}) = f(U_{i,G})$
                          END IF
                      ELSE
                          $nf_{k,G} = nf_{k,G} + 1$
                      END IF
                  END FOR
                  Store $ns_{k,G}$ and $nf_{k,G}$ ($k = 1, \ldots, K$) into the Success and Failure Memory
                  respectively.
        Step 3.6  Increment the generation count $G = G+1$
Step 4  END WHILE
```

V. NUMERICAL EXPERIMENTS AND RESULTS

A. Test Functions

As discussed in [32], many benchmark numerical functions commonly used to evaluate and compare optimization algorithms may suffer from two problems. First, global optimum lies at the center of the search range. Second, local optima lie along the coordinate axes or no linkage among the variables/dimensions exists. To solve these problems, we can shift or rotate the conventional benchmark functions. For benchmark functions suffering from the first problem, we may shift the global optimum to a random position so that the global optimum position has different numerical values for different dimensions, i.e., $F(\mathbf{x}) = f(\mathbf{x} - \mathbf{o}_{\text{new}} + \mathbf{o}_{\text{old}})$, where $F(\mathbf{x})$ is the new function, $f(\mathbf{x})$ is the old function, $\mathbf{o}_{\text{old}}$ is the old global optimum, and $\mathbf{o}_{\text{new}}$ is the new global optimum with different values for different dimensions and not lying at the center of the search range. For the second problem, we can rotate the function $F(\mathbf{x}) = f(\mathbf{M}\mathbf{x})$, where $\mathbf{M}$ is an orthogonal rotation matrix, to avoid local optima lying along the coordinate axes while retaining the properties of the test function. We hereby shift nine commonly used benchmark functions $(f_1 - f_5, f_7, f_9, f_{11-12})$, and further rotate three of them (f_6, f_8, f_{10}).

Two composition functions f_{13} and f_{14} are chosen from [32], which are constructed by using some basic benchmark functions to obtain more challenging problems. Gaussian function is used to combine the simple benchmark functions and blur the functions' structures. The composition functions are asymmetrical multimodal problems, with different properties in different areas. The detailed principle of constructing this class of functions is presented in [32], which is not repeated here.

In the following, 14 test functions are listed, among which functions $f_1 - f_4$ are unimodal and functions $f_5 - f_{14}$ are multimodal. These 14 test functions $(f_1 - f_{14})$ are dimensionwise scalable.

1) Shifted sphere function

$$f_1(x) = \sum_{i=1}^{D} z_i^2$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

2) Shifted Schwefel's Problem 1.2

$$f_2(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j \right)^2$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

3) Rosenbrock's function

$$f_3(x) = \sum_{i=1}^{D-1} \left(100 \left(x_i^2 - x_{i+1} \right)^2 + (x_i - 1)^2 \right).$$

4) Shifted Schwefel's Problem 1.2 with noise in fitness

$$f_4(x) = \left(\sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j \right)^2 \right) (1 + 0.4|N(0,1)|)$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

5) Shifted Ackley's function

$$f_5(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{D} z_i^2} \right) - \exp\left(\frac{1}{D} \sum_{i=1}^{D} \cos(2\pi z_i) \right) + 20 + e,$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

6) Shifted rotated Ackley's function[2]

$$f_6(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{D} z_i^2} \right) - \exp\left(\frac{1}{D} \sum_{i=1}^{D} \cos(2\pi z_i) \right) + 20 + e,$$
$$\mathbf{z} = \mathbf{M}(\mathbf{x} - \mathbf{o}), \qquad \text{cond}(\mathbf{M}) = 1$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

7) Shifted Griewank's function

$$f_7(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000}$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

8) Shifted rotated Griewank's function

$$f_8(x) = \sum_{i=1}^{D} \frac{z_i^2}{4000}$$
$$\mathbf{z} = \mathbf{M}(\mathbf{x} - \mathbf{o}), \qquad \text{cond}\ (\mathbf{M}) = 3$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

9) Shifted Rastrigin's function

$$f_9(x) = \sum_{i=1}^{D} (z_i^2 - 10\cos(2\pi z_i) + 10)$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

10) Shifted rotated Rastrigin's function

$$f_{10}(x) = \sum_{i=1}^{D} (z_i^2 - 10\cos(2\pi z_i) + 10)$$
$$\mathbf{z} = \mathbf{M}(\mathbf{x} - \mathbf{o}), \qquad \text{cond}\ (\mathbf{M}) = 2$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

11) Shifted noncontinuous Rastrigin's function

$$f_{11}(x) = \sum_{i=1}^{D} (y_i^2 - 10\cos(2\pi y_i) + 10)$$
$$y_i = \begin{cases} z_i, & |z_i| < 1/2 \\ \text{round}(2z_i)/2, & |z_i| >= 1/2 \end{cases}, \qquad \text{for } i = 1, 2, \ldots, D$$
$$\mathbf{z} = \mathbf{x} - \mathbf{o}$$
$$\mathbf{o} = [o_1, o_2, \ldots o_D] : \text{the shifted global optimum.}$$

[2]$\text{Cond}(\mathbf{M})$ means the condition number of rotation matrix $\mathbf{M}$.

TABLE III
GLOBAL OPTIMUM, SEARCH RANGES, AND INITIALIZATION RANGES OF THE TEST FUNCTIONS

f	Dimension	Global optimum x^*	$f(x^*)$	Search Range	Initialization Range
f_1		***o***	0	$[-100, 100]^D$	$[-100, 100]^D$
f_2		***o***	0	$[-100, 100]^D$	$[-100, 100]^D$
f_3		(1,1,…,1)	0	$[-100, 100]^D$	$[-100, 100]^D$
f_4		***o***	0	$[-32, 32]^D$	$[-32, 32]^D$
f_5		***o***	0	$[-32, 32]^D$	$[-32, 32]^D$
f_6		***o***	0	$\Re$	$[0,600]^D$
f_7	10 and 30	***o***	0	$\Re$	$[0,600]^D$
f_8		***o***	0	$[-5, 5]^D$	$[-5, 5]^D$
f_9		***o***	0	$[-5, 5]^D$	$[-5, 5]^D$
f_{10}		***o***	0	$[-5, 5]^D$	$[-5, 5]^D$
f_{11}		(420.96, …, 420.96)	0	$[-500, 500]^D$	$[-500, 500]^D$
f_{12}		(420.96, …, 420.96)	0	$[-500, 500]^D$	$[-500, 500]^D$
f_{13}		***o₁***	0	$[-5, 5]^D$	$[-5, 5]^D$
f_{14}		***o₁***	0	$[-5, 5]^D$	$[-5, 5]^D$
f_{15}	30	(0, …, 0)	0	$[-10, 10]^D$	
f_{16}	30	(0, …, 0)	0	$[-100, 100]^D$	
f_{17}	30	(1, …, 1)	0	$[-50, 50]^D$	
f_{18}	30	(1, …, 1)	0	$[-50, 50]^D$	
f_{19}	4	(0.1928,0.1908,0.1231,0.1358)	0.0003075	$[-5, 5]^D$	
f_{20}	2	(0.8983, -0.7126), (-0.08983 0.7126)	-1.0316285	$[-5, 5]^D$	Same as search range
f_{21}	2	(-3.142, 12.275), (3.142, 2.275), (9.425, 2.425)	0.398	$[-5, 10]\times[0, 15]$	
f_{22}	4	(0.114, 0.556, 0.852)	-3.86	$[0, 1]^D$	
f_{23}	6	(0.201, 0.150, 0.477, 0.275, 0.311, 0.657)	-3.32	$[0, 1]^D$	
f_{24}	4	[4, 4, 4, 4]	-10.2	$[0, 10]^D$	
f_{25}	4	[4, 4, 4, 4]	-10.4	$[0, 10]^D$	
f_{26}	4	[4, 4, 4, 4]	-10.5	$[0, 10]^D$	

***o** is the shifted vector.*
***o₁** is the shifted vector for the first basic function in the Composition function.*

12) Schwefel's function

$$f_{12}(x) = 418.9829 \times D - \sum_{i=1}^{D} x_i \sin\left(|x_i|^{1/2}\right).$$

13) Composition function 1 (CF1) in [32].
The function f_{13} (CF1) is composed by using ten sphere functions. The global optimum is easy to find once the global basin is found. The details of constructing such functions are presented in [32] and [37].

14) Composition function 6 (CF6) in [32].
The function f_{14} (CF6) is composed by using ten different benchmark functions, i.e., two rotated Rastrigin's functions, two rotated Weierstrass functions, two rotated Griewank's functions, two rotated Ackley's functions, and two rotated Sphere functions.
To make our test suite more comprehensive, we also chose an additional set of 12 test functions from [33] and [38].

15) Schwefel's Problem 2.22

$$f_{15}(x) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|.$$

16) Schwefel's Problem 2.21

$$f_{16}(x) = \max_i \{|x_i|, 1 \le i \le D\}.$$

17) Generalized penalized function 1

$$f_{17}(x) = \frac{\pi}{D}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2 \times \left[1 + 10\sin^2(\pi y_{i+1})\right] + (y_D - 1)^2\right\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

$$u(x_i,10,100,4)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a.\end{cases}$$

18) Generalized penalized function 2

$$f_{18}(x)=0.1\left\{\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_D-1)[1+\sin^2(2\pi x_D)]\right\}+\sum_{i=1}^{D}u(x_i,5,100,4).$$

19) Kowalik's function

$$f_{19}(x)=\sum_{i=1}^{11}\left[a_i-\frac{x_1\left(b_i^2+b_ix_2\right)}{b_i^2+b_ix_3+x_4}\right]^2.$$

20) Six-hump camel-back function

$$f_{20}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4.$$

21) Branin function

$$f_{21}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10.$$

22) Hartman's function 1

$$f_{22}(x)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2\right].$$

23) Hartman's function 2

$$f_{23}(x)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{6}a_{ij}(x_j-p_{ij})^2\right].$$

24) Shekel's family

$$f(x)=-\sum_{i=1}^{m}[(x-a_i)^T(x-a_i)+c_i]^{-1},$$

with $m=5,7,$ and 10 for $f_{24}(x), f_{25}(x)$ and $f_{26}(x)$.

B. Algorithms for Comparison

Experiments were conducted on a suite of 26 numerical functions to evaluate nine algorithms including the proposed SaDE algorithm. For functions f_1-f_{14}, both 10-dimensional (10-D) and 30-dimensional (30-D) functions were tested. The maximum number of function evaluations (FEs) is set to 100 000 when solving 10-D problems, and 300 000 when solving 30-D counterpart. For the remaining functions $f_{15}-f_{26}$, the function dimensions are listed in Table III, and the maximum number of FEs is set to 500 000 [38]. All experiments were run 30 times, independently. The nine algorithms in comparison are listed as follows:

- DE/rand/1/bin: $F=0.9, \mathrm{CR}=0.1$;
- DE/rand/1/bin: $F=0.9, \mathrm{CR}=0.9$;
- DE/rand/1/bin: $F=0.5, \mathrm{CR}=0.3$;
- DE/rand-to-best/1/bin: $F=0.5, \mathrm{CR}=0.3$;
- DE/rand-to-best/2/ bin with $F=0.5, \mathrm{CR}=0.3$;
- SaDE algorithm;
- adaptive DE algorithm [24];
- SDE algorithm [27];
- jDE [28].

Here, "DE/rand/1/bin" is chosen because it employs a most commonly used trial vector generation strategy, with three commonly suggested sets of control parameters: 1) $F=0.9, \mathrm{CR}=0.1$; 2) $F=0.9, \mathrm{CR}=0.9$; and 3) $F=0.5, \mathrm{CR}=0.3$ [15], [21], [33]. "DE/rand-to-best/1/bin" and "DE/rand-to-best/2/bin" employ more reliable trial vector generation strategies, and the control parameters are both set as $F=0.5, \mathrm{CR}=0.3$ in our experiments [8], [9].

We also choose three most representative adaptive DE variants proposed recently to compare with our proposed SaDE algorithm. The population sizes are all set to 50 and the learning periods are 50 for both 10-D and 30-D functions.

C. Experimental Results and Discussions

Tables IV and V report the mean and standard deviation of function values as well as the success rates by applying the nine algorithms to optimize the 10-D and 30-D numerical functions f_1-f_{14}, respectively. The best results are typed in bold. The success of an algorithm means that this algorithm can result in a function value no worse than the prespecified optimal value, i.e., $f(x*)+1\mathrm{e}-5$ for all problems with the number of FEs less than the prespecified maximum number. The success rate is calculated as the number of successful runs divided by the total number of runs.

Figs. 3 and 4 illustrate the convergence characteristics in terms of the best fitness value of the median run of each algorithm for functions f_1–f_{14} with $D=10$. Because the convergence graphs of the 30-D problems are similar to their 10-D counterparts, they are omitted here.

For functions f_{15}–f_{26}, because most algorithms can locate the global optima with 100% success rate within the specified maximum FEs, it is unnecessary to present the mean and standard deviation of function values. Instead, to compare the convergence speed, we report the average number of function evaluations (NFE) required to find the global optima when an algorithm solves the problem with 100% success rate.

1) Comparing SaDE With Conventional DE: In this section, we intend to show how well the proposed SaDE algorithm performs when compared to the conventional DE.

For the 10-D f_1–f_{14}, Tables IV and VI show that f_1 and f_5 are easily optimized by five DE variants and the SaDE algorithm with 100% success rate. Except these two functions, the

TABLE IV
RESULTS FOR 10-D PROBLEMS

Algorithm	f_1. Sphere			f_2. Schwefel's Problem1.2			f_3. Rosenbrock		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	**0**	**0**	100%	8.89E-01†	4.96E-01	0%	9.01E-01†	7.94E-01	0%
DE/rand/1/bin (F=0.9 CR=0.9)	4.95E-13	5.27E-13	100%	1.44E-05†	1.13E-05	43%	7.11E-03†	2.74E-02	0%
DE/rand/1/bin (F=0.5 CR=0.3)	**0**	**0**	100%	9.63E-09	5.99E-09	100%	1.76E+00†	1.54E+00	0%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	**0**	**0**	100%	**0**	**0**	100%	2.57E+00†	1.86E+00	0%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	**0**	**0**	100%	9.45E-13	9.90E-13	100%	2.37E+00†	2.23E+00	0%
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE(Zaharie)	**0**	**0**	100%	1.44E-04†	2.48E-04	3%	1.56E+00†	2.64E+00	0%
SDE	**0**	**0**	100%	**0**	**0**	100%	2.05E+00†	1.68E+00	0%
jDE	**0**	**0**	100%	**0**	**0**	100%	1.34E-13	7.32E-13	100%

Algorithm	f_4. Schwefel's Problem 1.2 with noise			f_5. Ackley			f_6. Rotated Ackley		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	2.41E+01†	1.28E+01	0%	**0**	**0**	**100%**	3.81E-05†	0.00013	90%
DE/rand/1/bin (F=0.9 CR=0.9)	2.42E-04†	1.38E-04	0%	4.59E-07†	2.41E-07	100%	6.86E-07†	3.89E-07	100%
DE/rand/1/bin (F=0.5 CR=0.3)	5.42E-06	4.44E-06	83%	**0**	**0**	**100%**	3.32E-15	9.01E-16	100%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	**0**	**0**	**100%**	4.97E-15	1.77E-15	100%	4.26E-15	1.45E-15	100%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	1.04E-08	1.20E-08	100%	3.55E-15	1.87E-15	100%	3.55E-15	0	100%
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE(Zaharie)	7.00E-02†	5.84E-02	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
SDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**
jDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**

Algorithm	f_7. Griewank			f_8. Rotated Griewank			f_9. Rastrigin		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	**0**	**0**	**100%**	1.22E-01†	2.77E-02	0%	**0**	**0**	**100%**
DE/rand/1/bin (F=0.9 CR=0.9)	3.05E-01†	2.02E-01	0%	2.41E-01†	2.00E-01	0%	8.71E+00†	5.53E+00	0%
DE/rand/1/bin (F=0.5 CR=0.3)	**0**	**0**	**100%**	1.60E-01†	3.75E-02	0%	**0**	**0**	**100%**
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	4.67E-03†	8.13E-03	70%	2.91E-01†	3.14E-01	0%	6.63E-02	2.52E-01	93%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	**0**	**0**	**100%**	1.44E-01†	3.97E-02	0%	**0**	**0**	**100%**
SaDE	**0**	**0**	**100%**	**1.37E-02**	1.18E-02	**20%**	**0**	**0**	**100%**
ADE(Zaharie)	2.55E-07	1.40E-06	100%	7.93E-02†	4.24E-02	0%	**0**	**0**	**100%**
SDE	7.39E-03†	7.59E-03	40%	3.81E-02†	3.06E-02	0%	6.96E-01†	8.72E-01	50%
jDE	5.75E-04	2.21E-03	93%	2.26E-02†	1.77E-02	7%	**0**	**0**	**100%**

Algorithm	f_{10}. Rotated Rastrigin			f_{11}. Non-continuous Rastrigin			f_{12}. Schwefel		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	1.33E+01†	3.00E+00	0%	0	0	100%	0	0	100%
DE/rand/1/bin (F=0.9 CR=0.9)	1.63E+01†	1.10E+01	0%	8.20E+00†	3.37E+00	0%	2.82E+00†	1.41E+01	13%
DE/rand/1/bin (F=0.5 CR=0.3)	1.65E+01†	2.99E+00	0%	0	0	100%	0	0	100%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	1.00E+01†	2.32E+00	0%	1.33E-01†	3.46E-01	87%	1.18E+01†	3.61E+01	90%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	1.63E+01†	3.36E+00	0%	0	0	100%	6.06E-14	2.31E-13	100%
SaDE	**3.80E+00**	1.35E+00	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE(Zaharie)	9.41E+00†	2.20E+00	0%	0	0	100%	0	0	100%
SDE	7.79E+00†	3.18E+00	0%	1.22E+00†	9.99E-01	27%	2.37E+01†	4.82E+01	80%
jDE	5.78E+00†	2.10E+00	0%	0	0	100%	0	0	100%

Algorithm	f_{13}. Composition Function1			f_{14}. Composition Function6		
	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	3.41E-02†	5.98E-02	0%	6.62E+00†	1.33E+00	0%
DE/rand/1/bin (F=0.9 CR=0.9)	4.27E-13	3.74E-13	100%	2.17E-01	5.64E-01	53%
DE/rand/1/bin (F=0.5 CR=0.3)	6.50E-01	3.56E+00	97%	9.48E+00†	2.49E+01	0%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	1.83E+01†	4.62E+01	80%	2.36E+01†	4.31E+01	17%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	1.02E-09	3.95E-09	100%	1.37E+00†	8.26E-01	0%
SaDE	**0**	**0**	**100%**	**2.54E-01**	5.21E-01	**80%**
ADE(Zaharie)	1.48E-01	5.97E-01	90%	5.87E+00†	4.85E+00	0%
SDE	3.00E+01†	4.66E+01	70%	8.24E+00†	2.50E+01	23%
jDE	1.33E+01†	3.46E+01	87%	1.27E+00†	3.20E+00	53%

†indicates that SaDE performs better than the other algorithm with 95% certainty by *t*-test.
‡ means that corresponding algorithm is better than SaDE.

TABLE V
RESULTS FOR 30-D PROBLEMS

Algorithm	f_1. Sphere			f_2. Schwefel's Problem1.2			f_3. Rosenbrock		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	**0**	**0**	**100%**	3.62E+03†	8.22E+02	0%	3.39E+01†	1.51E+01	0%
DE/rand/1/bin (F=0.9 CR=0.9)	4.50E-02†	6.15E-02	0%	1.73E+03†	1.40E+03	0%	1.04E+02†	6.25E+01	0%
DE/rand/1/bin (F=0.5 CR=0.3)	**0**	**0**	**100%**	1.38E+03†	2.53E+02	0%	2.14E+01†	1.98E+00	0%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	1.90E-07	1.04E-06	97%	1.92E+01†	2.27E+01	0%	6.37E+01†	4.01E+01	0%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	**0**	**0**	**100%**	1.04E+02†	8.25E+01	0%	2.08E+01†	1.19E+01	0%
SaDE	**0**	**0**	**100%**	**0**	**0**	100%	**3.99E-01**	1.22E+00	**90%**
ADE(Zaharie)	**0**	**0**	**100%**	3.04E+02†	6.86E+01	0%	4.69E+01†	2.64E+01	0%
SDE	4.56E-01†	2.08E+00	50%	1.58E+00†	4.48E+00	0%	7.73E+03†	3.27E+04	0%
jDE	**0**	**0**	**100%**	8.91E-11	1.27E-10	100%	5.77E-01	1.38E+00	40%

Algorithm	f_4. Schwefel's Problem 1.2 with noise			f_5. Ackley			f_6. Rotated Ackley		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	1.24E+04†	2.15E+03	0%	**0**	**0**	**100%**	3.81E-05†	0.00013	90%
DE/rand/1/bin (F=0.9 CR=0.9)	8.98E+03†	5.74E+03	0%	3.86E-02†	2.18E-02	0%	7.64E-02†	5.11E-02	0%
DE/rand/1/bin (F=0.5 CR=0.3)	5.19E+03†	1.24E+03	0%	4.03E-15	1.23E-15	100%	3.67E-15	6.49E-16	100%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	2.36E+00	5.47E+00	0%	3.10E-02†	1.70E-01	93%	4.82E-03†	2.64E-02	97%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	1.51E+03†	2.07E+02	0%	7.58E-15	1.80E-15	100%	7.34E-15	1.30E-15	100%
SaDE	3.37E+00	1.37E+01	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE(Zaharie)	6.75E+04†	1.02E+04	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
SDE	4.67E+02†	5.09E+02	0%	2.19E-01†	3.87E-01	40%	1.01E-01†	3.04E-01	63%
jDE	**2.15E-01**‡	4.91E-01	0%	**0**	**0**	**100%**	**0**	**0**	**100%**

Algorithm	f_7. Griewank			f_8. Rotated Griewank			f_9. Rastrigin		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	**0**‡	**0**	**100%**	9.12E-02†	3.08E-02	0%	**0**	**0**	**100%**
DE/rand/1/bin (F=0.9 CR=0.9)	1.52E-01†	1.15E-01	0%	9.01E-01†	1.40E-01	0%	8.54E+01†	3.30E+01	0%
DE/rand/1/bin (F=0.5 CR=0.3)	**0**‡	**0**	**100%**	**2.24E-05**‡	**1.19E-04**	**93%**	3.10E+01†	3.24E+00	0%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	1.08E+01	1.00E+01	0%	1.82E+02†	5.47E+01	0%	9.58E+00†	3.88E+00	0%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	**0**‡	**0**	**100%**	3.97E-03	1.85E-02	37%	4.03E+01†	3.73E+00	0%
SaDE	2.38E-03	5.03E-03	80%	8.54E-03	9.09E-03	40%	**0**	**0**	**100%**
ADE(Zaharie)	**0**‡	**0**	**100%**	2.93E-03	5.65E-03	10%	2.32E-01†	5.01E-01	80%
SDE	1.59E+00†	2.23E+00	13%	1.39E+00†	4.24E+00	13%	1.09E+01†	4.23E+00	0%
jDE	**0**‡	**0**	**100%**	5.17E-03	6.64E-03	57%	**0**	**0**	**100%**

Algorithm	f_{10}. Rotated Rastrigin			f_{11}. Non-continuous Rastrigin			f_{12}. Schwefel		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	1.68E+02†	1.43E+01	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
DE/rand/1/bin (F=0.9 CR=0.9)	2.45E+02†	2.20E+01	0%	6.93E+01†	2.37E+01	0%	4.98E+03†	1.75E+03	0%
DE/rand/1/bin (F=0.5 CR=0.3)	1.87E+02†	1.09E+01	0%	2.88E+01†	1.94E+00	0%	**0**	**0**	**100%**
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	1.44E+02†	1.09E+01	0%	1.18E+01†	2.70E+00	0%	4.18E+02†	1.75E+02	3%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	1.88E+02†	7.15E+00	0%	3.17E+01†	2.25E+00	0%	2.48E+03†	4.01E+02	0%
SaDE	**1.67E+01**	**5.26E+00**	0%	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE(Zaharie)	1.21E+02†	1.28E+01	0%	**0**	**0**	**100%**	3.95E+00	2.16E+01	97%
SDE	3.63E+01†	6.78E+00	0%	1.56E+01†	3.52E+00	0%	4.68E+02†	2.08E+02	0%
jDE	3.65E+01†	8.29E+00	0%	6.67E-02	2.54E-01	93%	6.32E+01†	1.07E+02	63%

Algorithm	f_{13}. Composition Function1			f_{14}. Composition Function6		
	Mean	Std	Success Rate	Mean	Std	Success Rate
DE/rand/1/bin (F=0.9 CR=0.1)	3.78E-03†	1.15E-02	47%	1.33E+01†	3.08E+00	0%
DE/rand/1/bin (F=0.9 CR=0.9)	5.06E-03†	5.54E-03	0%	1.56E+01†	5.46E+00	0%
DE/rand/1/bin (F=0.5 CR=0.3)	4.49E-06	4.62E-06	100%	1.15E+01†	1.54E+00	0%
DE/rand-to-best/1/bin (F=0.5 CR=0.3)	5.00E+01†	8.20E+01	53%	2.86E+01†	7.69E+01	0%
DE/rand-to-best/2/bin (F=0.5 CR=0.3)	2.17E-09	1.19E-08	100%	1.22E+01†	2.99E+00	0%
SaDE	**0**	**0**	**100%**	**2.17E+00**	**9.23E-01**	0%
ADE(Zaharie)	1.67E-03	9.13E-03	97%	4.18E+00†	1.41E+00	0%
SDE	1.29E-01†	3.62E-01	67%	1.16E+01†	1.86E+01	0%
jDE	**0**	**0**	**100%**	1.24E+01†	3.06E+01	0%

performances of DE/rand/1 $(\mathrm{CR} = 0.1)$ and DE/rand/1 $(\mathrm{CR} = 0.9)$ are almost contrary. For example, DE/rand/1/bin $(\mathrm{CR} = 0.9)$ obtains better results on unimodal functions f_2 and f_3 where DE/rand/1/bin $(\mathrm{CR} = 0.1)$ performs poorly. On the contrary, DE/rand/1/bin $(\mathrm{CR} = 0.1)$ performs efficiently and robustly on multimodal functions f_7, f_9, f_{11}, and f_{12}, while DE/rand/1/bin $(\mathrm{CR} = 0.9)$ fails totally on f_7, f_9, and f_{11}, only 13% success rate on f_{12}. Therefore, different values of CR for DE/rand/1/bin demonstrate diverse performances on different problems. DE/rand-to-best/1/bin and DE/rand-to-best/2/bin are more reliable than DE/rand/1/bin. However, they also totally or partially fail in optimizing some problems where SaDE can locate the global optima in every run. Overall, SaDE obtains a smaller mean value and a higher success rate than the five DE variants for all problems.

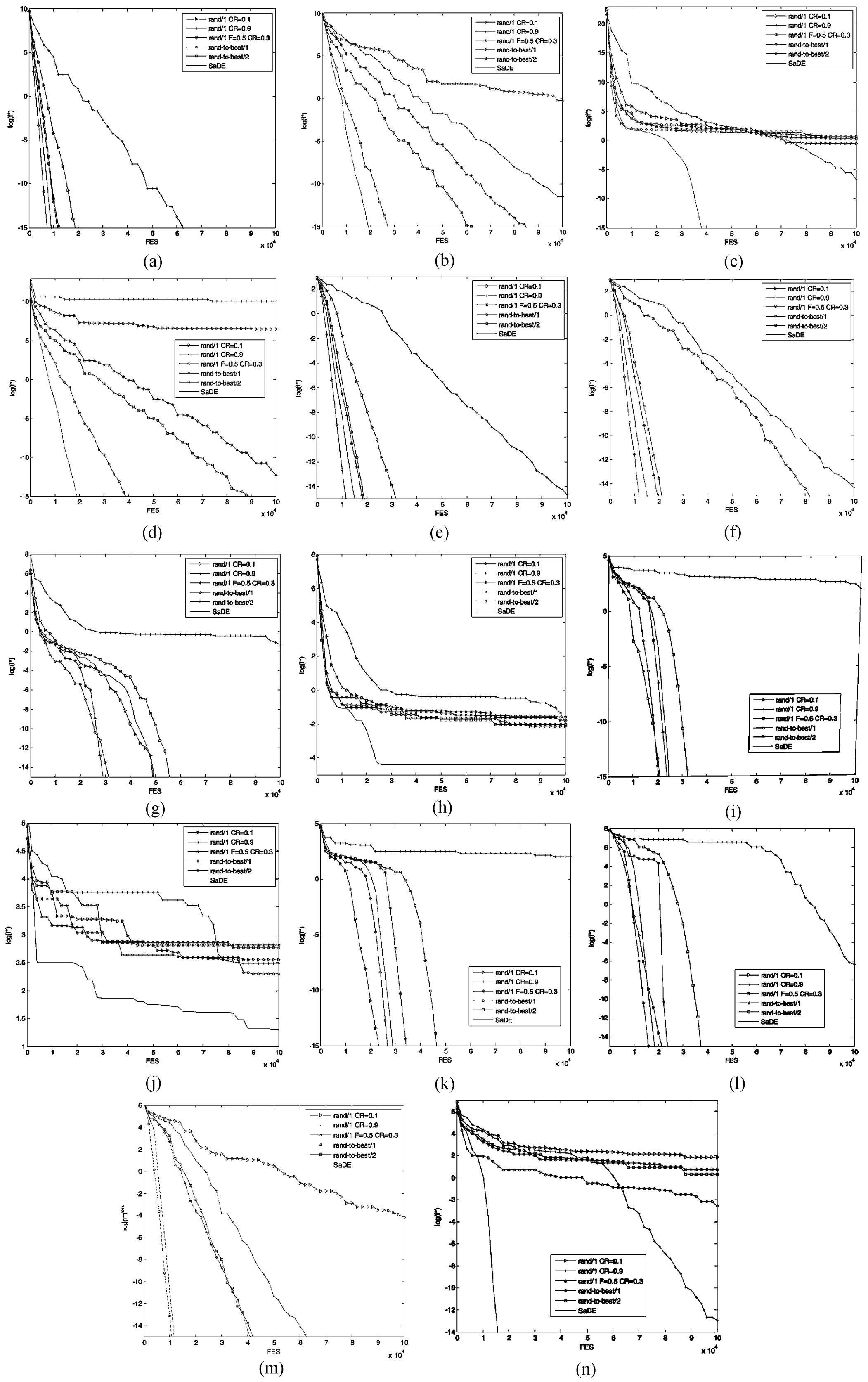


Fig. 3. Median convergence characteristics of rand/1 $\mathrm{CR} = 0.1$, rand/1 $\mathrm{CR} = 0.9$, rand/1 $F = 0.5, \mathrm{CR} = 0.3$, rand-to-best/1, rand-to-best/2, and SaDE on 10-D test functions f_1–f_{14}. (a) f_1: sphere; (b) f_2: Schwefel's 1.2; (c) f_3: Rosenbrock; (d) f_4: Schwefel's Problem 1.2 with noise in fitness; (e) f_5: Ackley; (f) f_6: rotated Ackley; (g) f_7: Griewank; (h) f_8: rotated Griewank; (i) f_9: Rastrigin; (j) f_{10}: rotated Rastrigin; (k) f_{11}: noncontinuous Rastrigin; (l) f_{12}: Schwefel; (m) f_{13}: composition function 1; and (n) f_{14}: composition function 6.

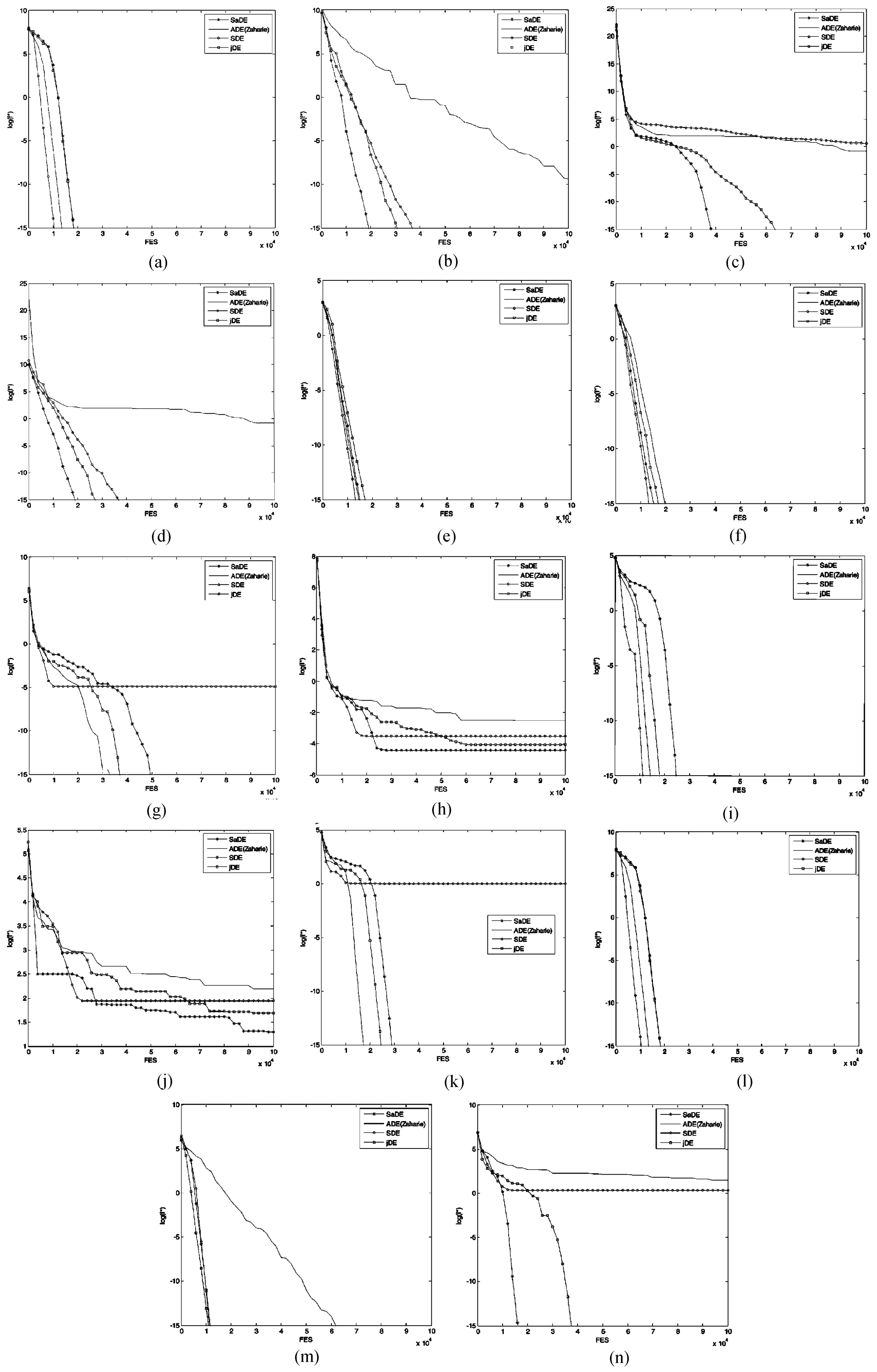


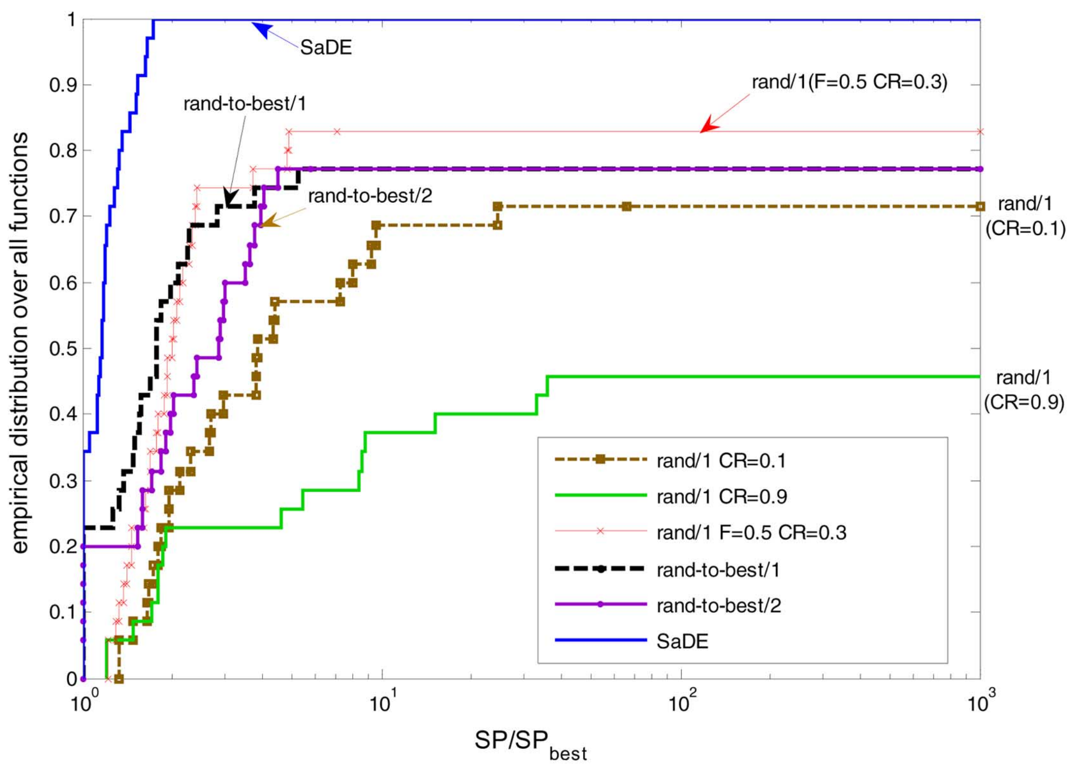
Fig. 4. Median convergence characteristics of SaDE, ADE (Zaharie), SDE, and jDE on 10-D test functions f_1–f_{14}. (a) f_1: sphere; (b) f_2: Schwefel's 1.2; (c) f_3: Rosenbrock; (d) f_4: Schwefel's Problem 1.2 with noise in fitness; (e) f_5: Ackley; (f) f_6: rotated Ackley; (g) f_7: Griewank; (h) f_8: rotated Griewank; (i) f_9: Rastrigin; (j) f_{10}: rotated Rastrigin; (k) f_{11}: noncontinuous Rastrigin; (l) f_{12}: Schwefel; (m) f_{13}: composition function 1; and (n) f_{14}: composition function 6.

TABLE VI
COMPARISON OF $\mathrm{Rand/1(F = 0.9CR = 0.1), rand/1(F = 0.9CR = 0.9), rand/1(F = 0.5, CR = 0.3)}$, RAND-TO-BEST/1, RAND-TO-BEST/2, AND SADE

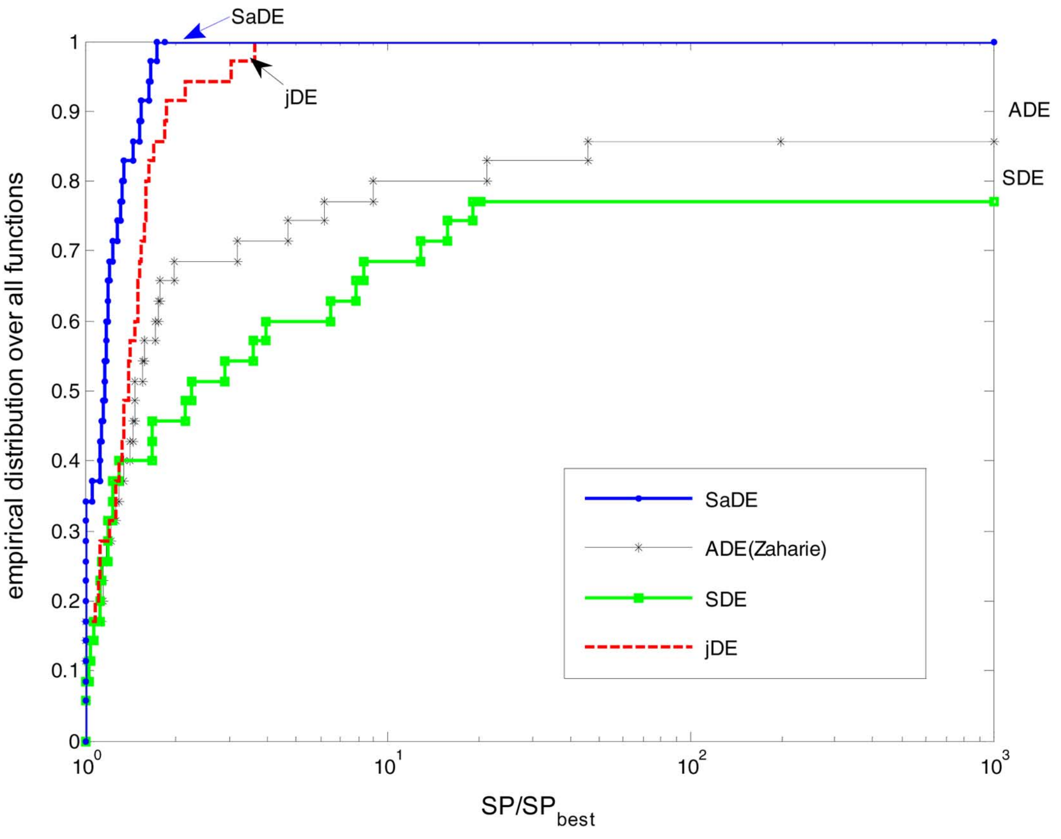
	rand/1/bin (F=0.9 CR=0.1)		rand/1/bin (F=0.9 CR=0.9)		rand/1/bin (F=0.5 CR=0.3)		rand-to-best/1/bin (F=0.5 CR=0.3)		rand-to-best/2/bin (F=0.5 CR=0.3)		SaDE	
10D	NFE	SR	NFE	SR	NFE	SR	NFE	SR	NFE	SR	NFE	SR
f_1	16770	100%	53298	100%	10291	100%	**6318**	100%	10058	100%	8375	100%
f_2	-	0%	-	43%	72436	100%	23383	100%	53658	100%	**14867**	100%
f_3	-	0%	-	0%	-	0%	-	0%	-	0%	**42446**	100%
f_4	-	0%	-	0%	-	83%	30925	100%	71278	100%	**15754**	100%
f_5	25335	100%	82919	100%	15157	100%	**9436**	100%	15045	100%	12123	100%
f_6	-	90%	85272	100%	16682	100%	**9923**	100%	16980	100%	12244	100%
f_7	41247	100%	-	0%	**29961**	100%	-	70%	59205	100%	35393	100%
f_9	**19200**	100%	-	0%	23155	100%	-	93%	30621	100%	23799	100%
f_{11}	**20898**	100%	-	0%	29559	100%	-	87%	46592	100%	26945	100%
f_{12}	19094	100%	-	13%	**14698**	100%	-	90%	33091	100%	16663	100%
f_{13}	-	0%	53131	100%	-	97%	-	80%	38290	100%	**9740**	100%
30D												
f_1	66339	100%	-	0%	34687	100%	-	97%	31470	100%	**20184**	100%
f_2	-	0%	-	0%	-	0%	-	0%	-	0%	**118743**	100%
f_5	92421	100%	-	0%	49822	100%	-	93%	45948	100%	**26953**	100%
f_6	-	0%	-	0%	55108	100%	-	97%	49961	100%	**33014**	100%
f_7	80741	100%	-	0%	39436	100%	-	0%	41314	100%	-	80%
f_9	90391	100%	-	0%	-	0%	-	0%	-	0%	**58723**	100%
f_{11}	108406	100%	-	0%	-	0%	-	0%	-	0%	**77920**	100%
f_{12}	78056	100%	-	0%	73756	100%	-	3%	-	0%	**44283**	100%
f_{13}	-	47%	-	0%	34012	100%	-	53%	38386	100%	**19031**	100%
f_{15}	80944	100%	-	0%	39460	100%	41729	100%	**18617**	100%	25137	100%
f_{16}	-	0%	-	0%	149511	100%	112445	100%	-	0%	**88934**	100%
f_{17}	62766	100%	-	90%	32420	100%	32442	100%	14289	100%	**18742**	100%
f_{18}	62388	100%	-	87%	31346	100%	28690	100%	-	93%	**19390**	100%
f_{19}	-	90%	7782	100%	31121	100%	33900	100%	-	93%	**6426**	100%
f_{20}	2744	100%	2628	100%	2178	100%	2086	100%	**1416**	100%	2076	100%
f_{21}	3659	100%	2713	100%	2246	100%	-	97%	**1593**	100%	2614	100%
f_{22}	1314	100%	996	100%	926	100%	**672**	100%	1004	100%	802	100%
f_{23}	5656	100%	-	60%	3970	100%	-	67%	4759	100%	**3080**	100%
f_{24}	31615	100%	7720	100%	10468	100%	-	90%	12381	100%	**4947**	100%
f_{25}	35503	100%	6596	100%	8653	100%	5085	100%	12921	100%	**4173**	100%
f_{26}	33905	100%	6974	100%	8742	100%	4853	100%	14933	100%	**4267**	100%

Furthermore, the convergence map of DE/rand/1/bin $(F = 0.9, \mathrm{CR} = 0.1)$, DE/rand/1/bin $(F = 0.9, \mathrm{CR} = 0.9)$, DE/rand/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$, DE/rand-to-best/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$, DE/rand-to-best/2/bin $(F = 0.5, \mathrm{CR} = 0.3)$, and SaDE in Fig. 3 shows that the SaDE algorithm always converges faster than others on six problems $(f_2, f_3, f_4, f_8, f_{10},$ and $f_{14})$, while slightly slower than DE/rand-to-best/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$ or DE/rand/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$ on the remaining problems. It can be observed that on f_9 DE/rand/1/bin $(\mathrm{CR} = 0.1)$, DE/rand-to-best/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$ and DE/rand/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$ converge fast, followed by SaDE. This is because a small CR value $(\mathrm{CR} = 0.1$ or $0.3)$ is an effective value to optimize Rastrigin problem, while our SaDE with an initial $\mathrm{CR} = 0.5$ needs some generations to self-adapt the parameters to suitable values. This issue will be discussed in Section V-D.

For the 30-D problems f_1–f_{14}, DE/rand/1/bin $(\mathrm{CR} = 0.9)$ has great difficulty in finding the global optima on all problems. DE/rand/1/bin $(\mathrm{CR} = 0.1)$ yields 100% success rate for $f_1, f_5, f_7, f_9, f_{11},$ and f_{12}. The SaDE algorithm performs much better with success rates of 100% on most problems, and 90% on f_3 where other DE variants totally fail in finding the global optima. When the dimension of variable was increased to 30, some problems such as functions $f_4, f_{10},$ and f_{14} become so difficult that all the six approaches could not find the optimal solutions within the maximum FEs. Regarding the speed of



(a)



(b)

Fig. 5. Empirical distribution of normalized success performance on all 26 test problems.

different algorithms in reaching the optimal solutions, we further compare the NFEs on problems with 100% success rate in Table VI, and find that SaDE converges the fastest among these algorithms on eight problems except f_7.

TABLE VII
RESULTS OF SaDE AND FADE ALGORITHMS

Test Function	Max. Gen.	Global Min.	SaDE		FADE	
			Mean	Var	Mean	Var
1.Sphere	5000	0	**0**	0	2.35E-10	2.97E-21
2.Rosenbrock	7000	0	**0**	0	4.16E+01	1.82E-02
3.Step	5000	0	**0**	**0**	0	0
4.Quartic	5000	15	**1.56E+01**	2.25E-01	1.90E+01	2.93E-01
6.Rastrigin	10000	0	**0**	0	2.58E+02	9.17E+02
7.Ackley	5000	0	**0**	0	5.90E-02	1.23E-06
10.Griewangk	5000	0	**0**	0	5.78E-01	3.50E-01

TABLE VIII
SaDE'S RESULTS WITH DIFFERENT LEARNING PERIOD (LP) ON TEST FUNCTIONS f_8, f_{10}, AND f_{14} (10-D)

LP	f_8. Rotated Griewank			f_{10}. Rotated Rastrigen			f_{14}. Composition Function 6		
	Mean	Std	Success Rate	Mean	Std	Success Rate	Mean	Std	Success Rate
20	6.67E-03	6.41E-03	27%	4.18E+00	1.21E+00	0%	1.01E+01	3.05E+01	83%
30	9.65E-03	7.61E-03	17%	3.98E+00	1.35E+00	0%	1.66E-01	4.32E-01	87%
40	5.17E-03	6.27E-03	43%	4.24E+00	9.96E-01	0%	3.45E+00	1.82E+01	87%
50	1.37E-02	1.18E-02	20%	3.80E+00	1.35E+00	0%	2.54E-01	5.21E-01	80%
60	1.16E-02	1.62E-02	27%	3.89E+00	1.24E+00	0%	9.22E-02	3.54E-01	93%

For the test problems f_{15}–f_{26}, both SaDE and DE/rand/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$ successfully solve all problems in each run, while SaDE shows an overall better convergence speed than DE/rand/1/bin $(F = 0.5, \mathrm{CR} = 0.3)$.

2) Comparing SaDE With Other Adaptive DE Variants: The performance of the SaDE is compared with three other adaptive DE variants: ADE [24], SDE [27], and jDE [28] on 10-D and 30-D problems f_1–f_{14}.

For 10-D problems, the best results of unimodal functions f_1–f_4 are obtained by SaDE and jDE, where SaDE demonstrates a slight superiority on efficiency, which can be observed from the convergence graphs [Fig. 4(a)–(d)]. Among the multimodal functions, f_5 Ackley and f_6 rotated Ackley are easily solved by all the adaptive DE variants with 100% success rate. For f_7 Griewank and f_9 Rastrigin, before rotation, SaDE, ADE, and jDE have high success rates as shown in Table IV. After rotation (f_8 and f_{10}), only SaDE and jDE successfully find the global optimum on f_8 in some run, and SaDE offers a higher success rate than jDE. f_{10} is so difficult that no algorithm could find the global optimum. For the last two composition problems f_{13} and f_{14}, SaDE is able to locate the optimal solutions with smaller standard deviations and higher success rates, demonstrating better efficiency and stability than the other three algorithms.

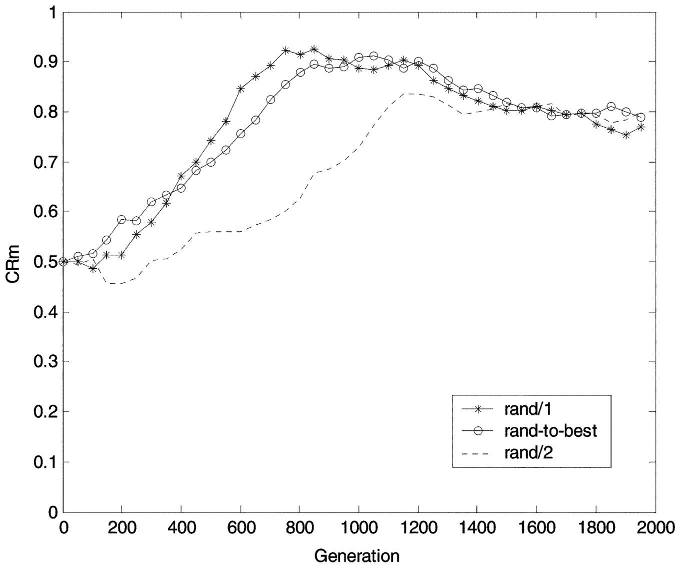
From the results of 30-D problems shown in Table V, we can observe that the algorithms achieved similar ranking as in the 10-D problems, and were not very successful in optimizing f_4, f_{10}, and f_{14}. However, with respect to the mean and standard deviations, SaDE obtains smallest values on f_{10} and f_{14}, and the second smallest values on f_4 where jDE gets the smallest values.

3) Overall Performance Comparison: In this part, we intend to further compare the overall performances of nine algorithms on all functions by plotting empirical distribution of normalized success performance [36]. The success performance is defined as success performance (SP) = mean (FEs for successful runs)*(# of total runs)/(# of successful runs).

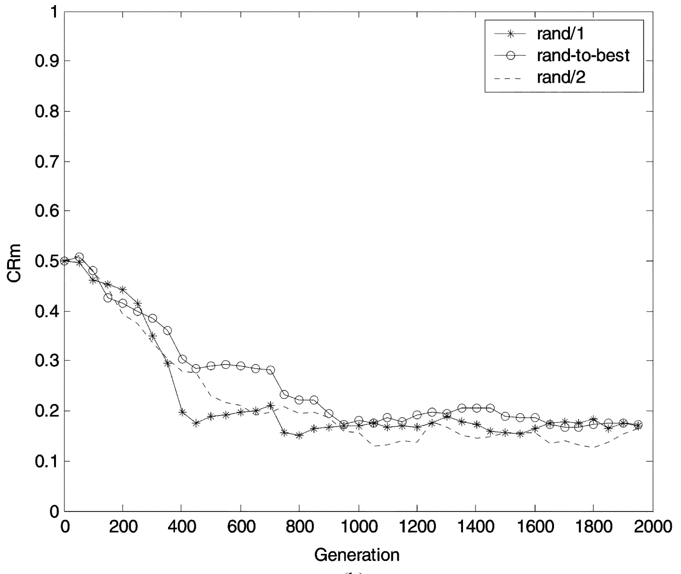
We first calculated the success performance of nine algorithms on each test function, and then normalized the SP by dividing all SPs by the SP of the best algorithm on the respective function. Results of all functions are used where at least one algorithm was successful in at least one run. Therefore, for 10-D f_1–f_{14} problems, we plot the results of all functions except f_{10}, and exclude functions f_4, f_{10}, and f_{14} for 30-D f_1–f_{14} problems. The test problems f_{15}–f_{26} are all plotted. Small values of SP and large values of the empirical distribution in graphs are preferable. The first one that reaches the top of the graph will be regarded as the best algorithm.

From Fig. 5, we can observe that SaDE outperforms other approaches on overall performance on $36 (= 40 - 4)$ test problems.

4) Comparison With FADE: The FADE algorithm was tested on a set of standard test functions in [18], including 2, 3, 30, and 50 dimensions. Because the low-dimensional (2 or 3) test functions are easy to solve for both conventional DE and FADE, we hereby only compare our SaDE with the FADE algorithm on 50-D test functions chosen from [18]. The parameter settings are the same as in [18]: population size $\mathrm{NP} = 10 \cdot D$, and maximum generations are listed in Table VII. The averaged results of 100 independent runs are summarized in the table (results for FADE are taken from [18]), which show that the proposed SaDE algorithm obviously performs better than the FADE algorithm.

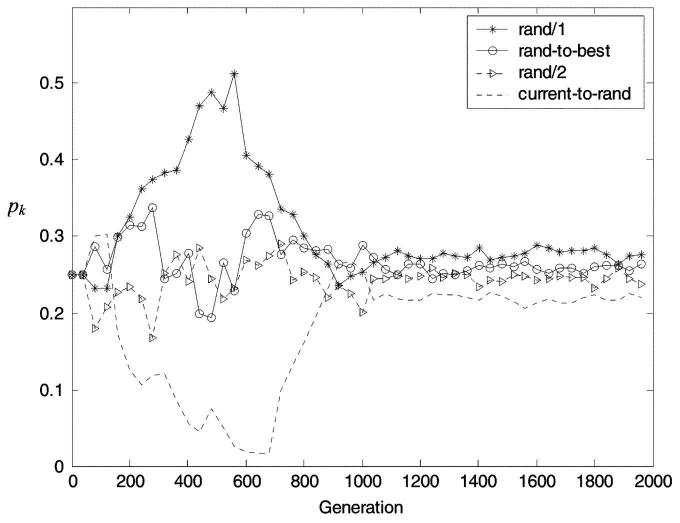


(a)

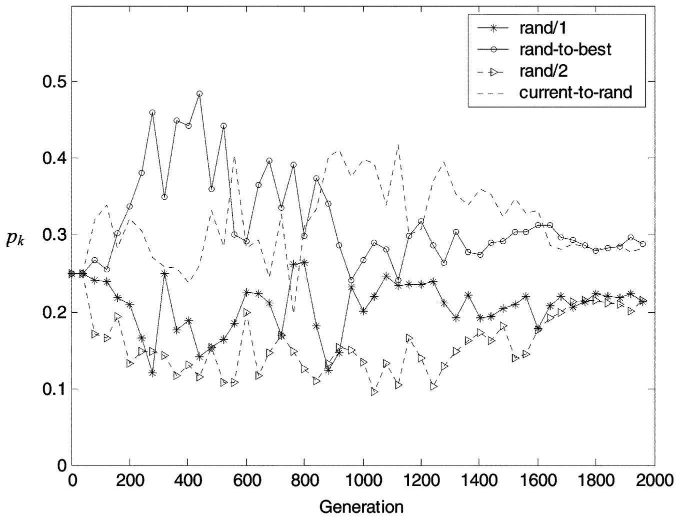


(b)

Fig. 6. Self-adaptation characteristics of CRm on Rosenbrock and Rastrigin functions (10-D). (a) Rosenbrock function. (b) Rastrigin function.



(a)



(b)

Fig. 7. Self-adaptation characteristics of strategies (10-D). (a) Griewank. (b) Rotated Griewank.

D. Analysis of Self-Adaptation Property of SaDE

1) Self-Adaptation of Crossover Probability: Usually, when DE solves problems, there is no specific value of CR and F, which is effective in all search phases. Instead, possibly several combinations of different CR and F values can be effective in different search phases. However, Rosenbrock and Rastrigin function are exceptional. From the experiments of DE/rand/1/bin on these two functions, we know that usually DE/rand/1/bin with a large CR could obtain a good result on Rosenbrock function and a small CR is beneficial to Rastrigin function, which could also be concluded from comparing the results of DE/rand/1/bin $\mathrm{CR} = 0.1$ and $\mathrm{CR} = 0.9$. As we know the suitable CR values for Rosenbrock and Rastrigin functions, we can observe the variation of CR values in SaDE algorithm to check whether the self-adaptation of CR is effective.

Because the CR value in SaDE is mostly depended on the mean value CRm of normal distribution, we plot the CRm value when the SaDE algorithm optimized Rosenbrock and Rastrigin functions as the generation increases in Fig. 6. We found that for Rosenbrock function, the CRm values of strategy “rand/1/bin,” “rand-to-best/2/bin,” and “rand/2/bin” keep increasing during evolution, as we expected. On the contrary, for Rastrigin function, the CRm values of three strategies all keep decreasing during evolution. Therefore, we can say that our proposed SaDE algorithm self-adapts the crossover probability effectively.

2) Self-Adaptation of Trial Vector Generation Strategy: To investigate the self-adaptive selection of trial vector generation strategy in SaDE algorithm, we plot the variations of the four different strategies’ probabilities as the evolution progresses. As we know that the strategy “current-to-rand” is rotationally invariant [20] and has superior performance on rotated problem as stated in Section II, this strategy should occupy more proportion if it yields good results when dealing with rotated problems. As shown in Fig. 7, for Griewank’s function, the strategy “current-to-rand” occupied a very small proportion during the whole evolution progress. At the beginning, we assign each strategy equal probability. Because the strategy “current-to-rand” could

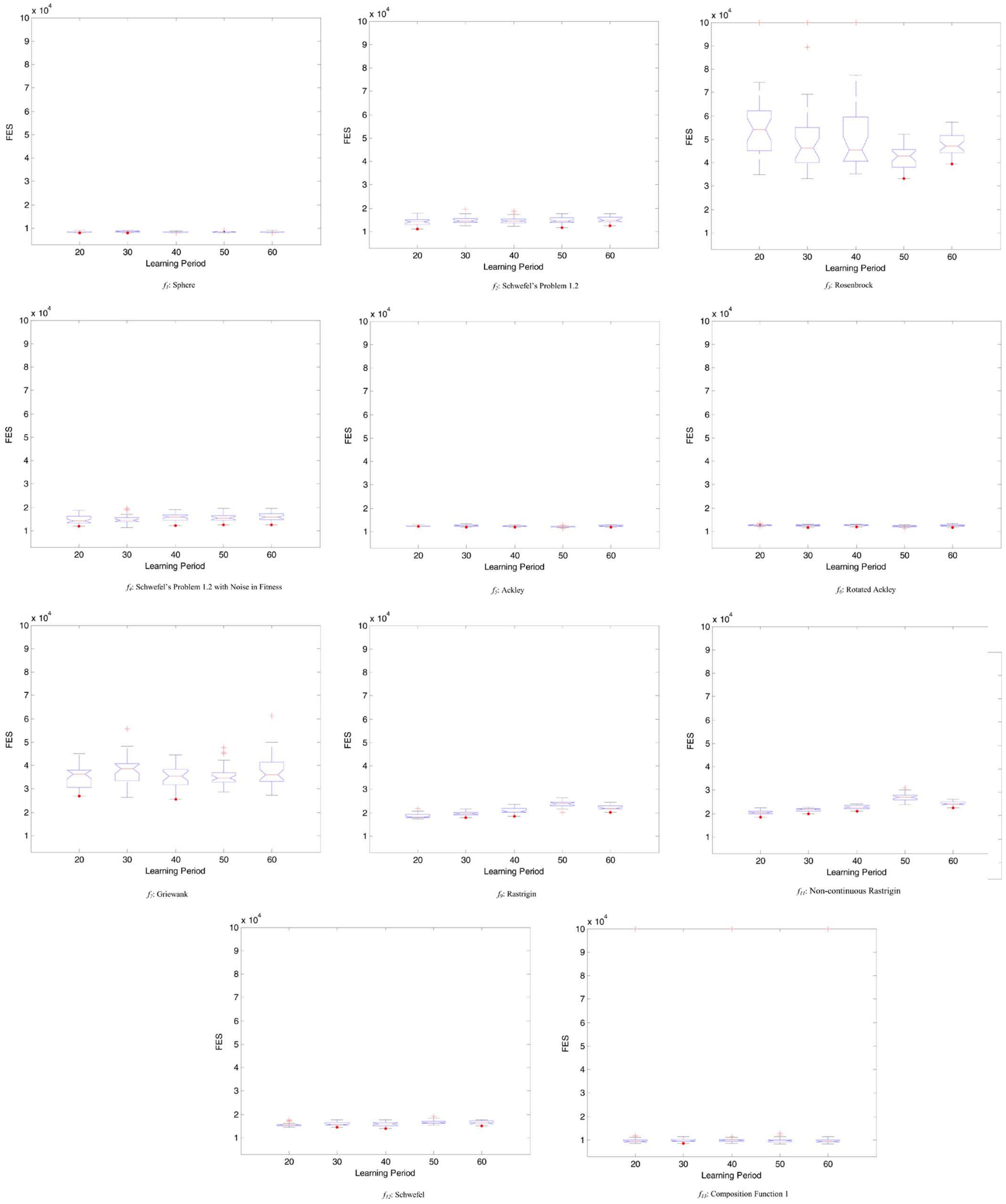


Fig. 8. SaDE's results with different learning period LP on 11 test functions (10-D).

not yield better results, our self-adaptive mechanism of learning is unlikely to choose the strategy "current-to-rand" in next few generations. As this strategy always performed badly, its proportion almost lowered to 0. The individuals were mostly mutated by the other three strategies that always occupy proportions above 0.25 as shown in Fig. 7. After 800 generations, the algorithm converged, and these four strategies performed competitively so as to all occupy around 0.25 proportion. On the contrary, after we rotated the Griewank's function, the strategy "current-to-rand" demonstrated a good performance at the beginning so as to occupy a higher proportion in 300 generations. During about 300–550 generations, the proportion of strategy "current-to-rand" slightly decreased under 0.25 because relatively inferior solutions are found by this strategy during this stage of evolution. After 550 generations, strategy "current-to-rand" occupied a higher proportion again until the algorithm converged and four strategies were almost stable.

E. Parameter Study

1) Learning Period (LP): The learning period parameter LP needs to be optimized. In this section, we use 10-D test functions f_1–f_{14} to investigate the impact of this parameter on SaDE algorithm. The SaDE algorithm runs 30 times on each function with five different learning periods LP of 20, 30, 40, 50, and 60. Because functions f_{1-7}, f_9, and f_{11-13} are solved with 100% success rate with five different LP values, we here omit the results of mean and standard deviation for these functions; only the results of f_8, f_{10}, and f_{14} are shown in Table VIII. However, for functions f_{1-7}, f_9, and f_{11-13}, we investigate the influence of LPs on convergence speed of SaDE algorithm, by comparing the FEs that SaDE algorithm cost to obtain the global optimum with different LPs.

Fig. 8 shows the box plots of FEs that SaDE algorithm requires to reach the global optimum with five different LPs. The box has lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to show the extent of the remaining data. Outliers are data with values beyond the ends of the whiskers. If there is no data outside the whisker, a dot is placed at the bottom whisker. From Fig. 8, we found that SaDE algorithm succeeds on all functions with similar FEs by using five different LPs. Therefore, the convergence speed of SaDE algorithm is less sensitive to the parameter LP parameter values between 20 and 60.

VI. Conclusion

This paper presented a SaDE algorithm, which eliminates the time-consuming exhaustive search for the most suitable trial vector generation strategy and the associated control parameters F and CR. In SaDE, trial vector generation strategies together with their two control parameters will be probabilistically assigned to each target vector in the current population according to the probabilities gradually learned from the experience to generate improved solutions. We have investigated the self-adaptive characteristics of the CR value and trial vector generation strategies. Experiments showed that the SaDE algorithm could evolve suitable strategies and parameter values as evolution progresses. The sensitivity analysis of LP parameter indicated that it had insignificant impact on the performance of the SaDE.

We have compared the performance of SaDE with the conventional DE and three adaptive DE variants over a suite of 26 bound constrained numerical optimization problems, and concluded that SaDE was more effective in obtaining better quality solutions, which are more stable with the relatively smaller standard deviation, and had higher success rates.

References

[1] A. Tuson and P. Ross, "Adapting operator settings in genetic algorithms," *Evolut. Comput.*, vol. 6, no. 2, pp. 161–184, 1998.

[2] J. Gomez, D. Dasgupta, and F. Gonzalez, "Using adaptive operators in genetic search," in *Proc. Genetic Evolut. Comput. Conf.*, Chicago, IL, Jul. 2003, pp. 1580–1581.

[3] B. R. Julstrom, "What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm," in *Proc. 6th Int. Conf. Genetic Algorithms*, Pittsburgh, PA, Jul. 15–19, 1995, pp. 81–87.

[4] P. J. Angeline, "Adaptive and self-adaptive evolutionary computation," in *Computational Intelligence: A Dynamic System Perspective*, M. Palaniswami, Y. Attikiouzel, R. J. Marks, D. Fogel, and T. Fukuda, Eds. New York: IEEE Press, 1995, pp. 152–161.

[5] J. E. Smith and T. C. Fogarty, "Operator and parameter adaptation in genetic algorithms," *Soft Comput.*, vol. 1, no. 2, pp. 81–87, Jun. 1997.

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

[7] R. Storn and K. V. Price, "Differential evolution-A simple and efficient heuristic for global optimization over continuous Spaces," *J. Global Optim.*, vol. 11, pp. 341–359, 1997.

[8] K. Price, R. Storn, and J. Lampinen, *Differential Evolution—A Practical Approach to Global Optimization*. Berlin, Germany: Springer-Verlag, 2005.

[9] V. Feoktistov, *Differential Evolution: In Search of Solutions*. Berlin, Germany: Springer-Verlag, 2006.

[10] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed-forward neural networks," *Neural Process. Lett.*, vol. 7, no. 1, pp. 93–105, 2003.

[11] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. Biennial Conf. North Amer. Fuzzy Inf. Process. Soc.*, Berkeley, CA, 1996, pp. 519–523.

[12] R. Storn, "Differential evolution design of an IIR-filter," in *Proc. IEEE Int. Conf. Evolut. Comput.*, Nagoya, Japan, May 1996.

[13] T. Rogalsky, R. W. Derksen, and S. Kocabiyik, "Differential evolution in aerodynamic optimization," in *Proc. 46th Annu. Conf. ofCan. Aeronaut. Space Inst.*, Montreal, QC, Canada, May 1999, pp. 29–36.

[14] R. Joshi and A. C. Sanderson, "Minimal representation multisensor fusion using differential evolution," *IEEE Trans. Syst. Man Cybern. A, Syst. Humans*, vol. 29, no. 1, pp. 63–76, Jan. 1999.

[15] R. Storn and K. Price, "Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces," TR-95-012, 1995 [Online]. Available: http://http.icsi.berkeley.edu/~storn/litera.html

[16] J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," in *Proc. 6th Int. Mendel Conf. Soft Comput.*, P. Ošmera, Ed., 2002, pp. 76–83.

[17] R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," in *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, A. Grmela and N. E. Mastorakis, Eds. Interlaken, Switzerland: WSEAS Press, 2002, pp. 293–298.

[18] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Comput.*, vol. 9, no. 6, pp. 448–462, Apr. 2005.

[19] A. K. Qin and P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proc. IEEE Congr. Evolut. Comput.*, Edinburgh, Scotland, Sep. 2005, pp. 1785–1791.

[20] K. V. Price, "An introduction to differential evolution," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London, U.K.: McGraw-Hill, 1999, pp. 79–108.

[21] J. Rönkkönen, S. Kukkonen, and K. V. Price, "Real-parameter optimization with differential evolution," in *Proc. IEEE Congr. Evolut. Comput.*, Edinburgh, Scotland, Sep. 2005, pp. 506–513.

[22] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Comput.*, vol. 10, no. 8, pp. 637–686, 2006.

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

[24] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proc. Mendel 9th Int. Conf. Soft Comput.*, R. Matousek and P. Osmera, Eds., Brno, Czech Republic, Jun. 2003, pp. 41–46.

[25] D. Zaharie and D. Petcu, "Adaptive pareto differential evolution and its parallelization," in *Proc. 5th Int. Conf. Parallel Process. Appl. Math.*, Czestochowa, Poland, Sep. 2003, pp. 261–268.

[26] H. A. Abbass, "The self-adaptive Pareto differential evolution algorithm," in *Proc. Congr. Evolut. Comput.*, Honolulu, HI, May 2002, pp. 831–836.

[27] M. G. H. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution," in *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer-Verlag, 2005, pp. 192–199.

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

[29] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in *ACM-SIGEVO Proc. Genetic Evolut. Comput. Conf.*, Washington, DC, pp. 991–998.

[30] U. K. Chakraborty, S. Das, and A. Konar, "Differential evolution with local neighborhood," in *Proc. Congr. Evolut. Comput.*, Vancouver, BC, Canada, 2006, pp. 2042–2049.

[31] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms*, Cambridge, MA, 1987, pp. 14–21.

[32] J. J. Liang, P. N. Suganthan, and K. Deb, "Novel composition test functions for numerical global optimization," in *Proc. IEEE Swarm Intell. Symp.*, Pasadena, CA, Jun. 2005, pp. 68–75.

[33] J. Vesterstrøm and R. Thomson, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Proc. IEEE Congr. Evolut. Comput.*, Portland, OR, June 2004, pp. 1980–1987.

[34] A. Iorio and X. Li, "Solving rotated multi-objective optimization problems using differential evolution," in *Proc. Australian Conf. Artif. Intell.*, Cairns, Dec. 2004, pp. 861–872.

[35] W. J. Zhang and X. F. Xie, "DEPSO: Hybrid particle swarm with differential evolution operator," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, Washington, DC, 2003, pp. 3816–3821.

[36] N. Hansen, "Compilation of results on the 2005 CEC benchmark function set," May 4, 2006 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/compareresults.pdf

[37] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technol. Univ., Singapore, Tech. Rep. KanGAL #2005005, May 2005, IIT Kanpur, India.

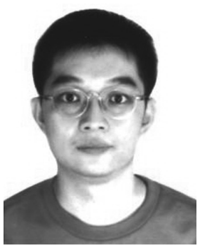
[38] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evolut. Comput.*, vol. 3, no. 2, pp. 82–102, Jul. 1999.

[39] Z. Y. Yang, E. K. Tang, and X. Yao:, "Large scale evolutionary optimization using cooperative coevolution," *Inf. Sci.*, accepted for publication.

[40] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 1, pp. 64–79, Feb. 2008.

[41] N. Noman and H. Iba, "Accelerating differential evolution using an adaptive local search," *IEEE Trans. Evolut. Comput.*, vol. 12, no. 1, pp. 107–125, Feb. 2008.

[42] J. Brest, B. Boskovic, S. Greiner, V. Zumer, and M. S. Maucec, "Performance comparison of self-adaptive and adaptive differential evolution algorithms," *Soft Comput.*, vol. 11, no. 7, pp. 617–629, May 2007.



A. K. Qin received his B.E. Degree from Department of Automatic Control Engineering of Southeast University, Nanjing, P.R. China in 2001. He completed his Ph.D. degree in the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore in 2007. His research interests include pattern recognition, machine learning, neural network, genetic and evolutionary algorithms, computer vision and bioinformatics.

V. L. Huang received the B. E. degree from Huazhong University of Sci. & Tech. Wuhan, P. R. China in 2002. She has been worked toward the Ph.D. degree in the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore since 2004. Her research interests include evolutionary algorithms, differential evolution, and applications of evolutionary algorithms.

P. N. Suganthan received the B.A degree, Postgraduate Certificate and M.A. degree in electrical and information engineering from the University of Cambridge, UK in 1990, 1992 and 1994, respectively. He obtained his Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore in 1996. He was a predoctoral Research Assistant in the Department of Electrical Engineering, University of Sydney in 1995–96 and a lecturer in the Department of Computer Science and Electrical Engineering, University of Queensland in 1996–99. Between 1999 and 2003, he was an Assistant Professor in the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore where he is now an Associate Professor. He is an associate editor of the *IEEE Transactions on Evolutionary Computation and Pattern Recognition* Journal. His research interests include evolutionary computation, applications of evolutionary computation, neural networks, pattern recognition and bioinformatics. He is a senior member of the IEEE.