International Journal on Artificial Intelligence Tools
Vol. 24, No. 6 (2015) 1550023 (33 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218213015500232

LADE: Learning Automata Based Differential Evolution

Mahshid Mahdaviani

Soft Computing Laboratory, Computer Engineering and Information Technology Department
Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran
mahshid.mahdaviani@gmail.com

Javidan Kazemi Kordestani

Department of Computer Engineering, Science and Research Branch
Islamic Azad University, Tehran, Iran
Javidan.kazemi@gmail.com

Alireza Rezvanian*

Soft Computing Laboratory, Computer Engineering and Information Technology Department
Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran
rezvanian@gmail.com
Department of Computer Engineering, Hamedan Branch
Islamic Azad University, Hamedan, Iran
a.rezvanian@aut.ac.ir

Mohammad Reza Meybodi

Soft Computing Laboratory, Computer Engineering and Information Technology Department
Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran
mmeybodi@aut.ac.ir

Received 12 August 2014
Accepted 7 April 2015
Published 21 December 2015

Many engineering optimization problems do not standard mathematical techniques, and cannot be solved using exact algorithms. Evolutionary algorithms have been successfully used for solving such optimization problems. Differential evolution is a simple and efficient population-based evolutionary algorithm for global optimization, which has been applied in many real world engineering applications. However, the performance of this algorithm is sensitive to appropriate choice of its parameters as well as its mutation strategy. In this paper, we propose two different underlying classes of learning automata based differential evolution for adaptive selection of crossover probability and mutation strategy in differential evolution. In the first class, genomes of the population use the same mutation strategy and crossover probability. In the second class, each genome of the population adjusts its own mutation strategy and crossover probability parameter separately. The performance of the proposed methods is analyzed on ten benchmark functions from

*Corresponding author.

CEC 2005 and one real-life optimization problem. The obtained results show the efficiency of the proposed algorithms for solving real-parameter function optimization problems.

Keywords: Evolutionary algorithms; continuous optimization; differential evolution; learning automata; parameter adaptation.

List of acronyms

EAs	Evolutionary Algorithms
GA	Genetic Algorithm
ACO	Ant Colony Optimization
AIS	Artificial Immune System
PSO	Particle Swarm Optimization
DE	Differential Evolution
LA	Learning Automata
CLA	Cellular Learning Automata
FA	Firefly Algorithm
PC	Parameter vector Change magnitude
FC	Function value Change
CoDE	Composite DE
EDA	Estimation of Distribution Algorithm
CDE	Crowding-based Differential Evolution
QOX	Quantization Orthogonal crossover
VSLA	Variable Structure Learning Automata
GLADE	Group Learning Automata based Differential Evolution
ILADE	Individually Learning Automata based Differential Evolution
NFL	No Free Lunch
Fes	Function Evaluations
LADE	Learning Automata based DE
FMSW	Frequency-Modulated Sound Waves

1. Introduction

Global optimization has been widely applied across different branches of engineering and science. Typical examples of global optimization in real-world applications include: financial planning optimization, chemical engineering design/control, mathematical function optimization and electrical circuit optimization/design. The objective of global optimization is to find the best solution of a given problem, in a set of all feasible solutions, in order to satisfy some optimality measures. Comprehensive study on global optimization can be found in Refs. 1 and 2.

One of the most widely used numerical methods for solving global optimization problems are Evolutionary Algorithms (EAs). EAs are global probabilistic search techniques based on natural evolution that have been used successfully in a variety of applications. Compared to the classical methods of optimization, EAs offer several practical advantages when facing complex optimization problems. Some of these advantages include the simple structure of the procedure, robustness to changing circumstances and the ability to self-adapt the optimum seeking process during the run. Hence, they seem to be a good candidate for solving global optimization problems. Several evolutionary algorithms have been proposed for function optimization. Genetic Algorithm (GA),[3] Ant Colony Optimization (ACO),[4] Artificial Immune System (AIS),[5] Particle Swarm Optimization (PSO),[6] Group Search Optimizer (GSO),[7] water drop algorithms (WDA)[8] and Differential Evolution (DE)[9] are examples of such methods. However, most of these methods suffer from premature convergence and have a slow convergence rate.

Recently, many researchers have incorporated Learning Automata (LA) into mentioned algorithms with the aim to enhance their performance in function optimization.[10–13] For instance, Rastegar *et al.*[14] proposed a combination of EAs and Cellular Learning Automata (CLA) called CLA-EC. They have assigned each genome of the population to a cell of the CLA and equipped each cell with a set of LAs to determine the string genome for that cell. In each cell, they generate a reinforcement signal based on a local rule for selecting an action and updating the internal structure of the LA. Abtahi *et al.*[15] used LA along with co-evolutionary GA to learn whether or not the variables of a given problem are dependent. Then in each case they choose an appropriate approach to solve the problem. Rezvanian *et al.*[13] used LA for tuning the mutation rate of antibodies in order to establish a balance between the process of global and local search in AIS. Hashemi *et al.*[16] introduced two classes of LA based algorithms for adaptive selection of value for inertia weight and acceleration coefficients in PSO. In both classes, they used an LA per w, c_1 and c_2 in order to choose an optimal value for corresponding parameter at each stage of the algorithm. Vafashoar *et al.*[17] proposed a model based on CLA and DE, namely CLA-DE. In CLA-DE, the search dimensions of the problem are iteratively partitioned via LA in the CLA and learning process is guided toward the most admissible partition. Moreover, they used DE to facilitate the incorporation among neighboring cells. Farahani *et al.*[18] applied LA for adjusting the parameters of the Firefly Algorithm (FA). In comparison with standard FA, their proposed method shows a better performance on a set of standard test functions.

Recently, DE has attracted a considerable deal of attention regarding its potential as an optimization technique for numerical problems and several modifications of DE proposed and applied in various domains.[19–22] Compared to some other competitive optimization algorithms, DE exhibits much better performance.[19] That is the reason why in this paper we have chosen DE as the foundation of our work. Despite its effectiveness, the performance of DE is highly sensitive to the value of its control parameters (i.e. F and CR). In this paper, we propose two classes of LA based algorithms to improve the performance of standard DE. The rest of the paper is organized as follows: The general principles of DE are given in Section 2. Section 3 reviews the related works on DE. Section 4 briefly presents LA. The proposed algorithms are introduced in Section 5. Section 6 is devoted to experimental setup. Experimental results are reported in Section 7. Finally, Section 8 concludes the paper.

2. Differential Evolution

DE[23] is among the most powerful stochastic real-parameter optimization algorithms, proposed by Storn and Price.[24, 25] The main idea of DE is to use spatial difference among

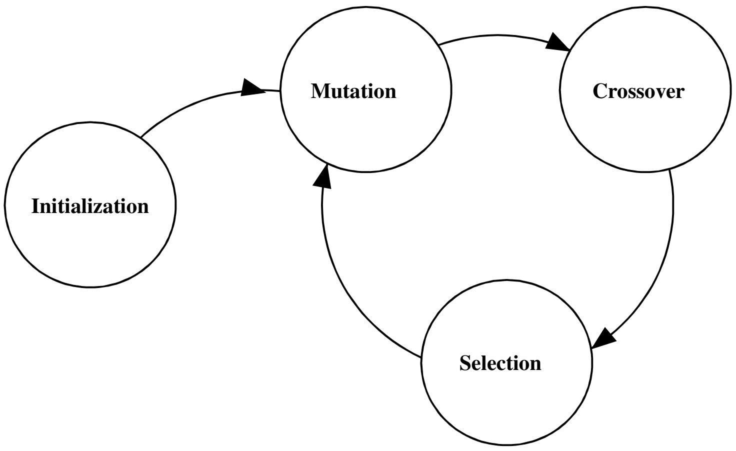


Fig. 1. Schematic view of the main stages of DE algorithm.

the population of vectors to guide the search process toward the optimum solution. The main steps of DE are illustrated in Figure 1.

The rest of this section describes the main operational stages of DE in detail.

2.1. *Initialization of vectors*

DE starts with a population of *NP* randomly generated vectors in a *D*-dimensional search space. Each vector *i*, also known as *genome* or *chromosome*, is a potential solution to an optimization problem which is represented by $\vec{X}_i = (x_{1i}, x_{2i}, \dots, x_{Di})$. The initial population of vectors is simply randomized into the boundary of the search space according to a uniform distribution as follows:

$$\vec{X}_i = l_j + rand_j[0,1] \times (u_j - l_j) \tag{1}$$

where $i \in [1, 2, \dots, NP]$ is the index of *i*th vector of the population, $j \in [1, 2, \dots, D]$ represents *j*th dimension of the search space, $rand_j[0,1]$ is a uniformly distributed random number corresponding to *j*th dimension. Finally, l_j and u_j are the lower and upper bounds of the search space corresponding to *j*th dimension of the search space.

2.2. *Difference-vector based mutation*

After initialization of the vectors in the search space, a mutation is performed on each genome *i* of the population to generate a donor vector $\vec{v}_i = (v_{1i}, v_{2i}, \dots, v_{Di})$ corresponding to target vector $\vec{X}_i$. Several strategies have been proposed for generating donor vector $\vec{v}_i$. In this paper, we use the following mutation strategies to create donor vector[22]:

- *DE/rand/*1:

$$\vec{v}_\mathrm{i} = \vec{x}_1 + \mathcal{F}.(\vec{x}_2 - \vec{x}_3) \tag{2}$$

- *DE/rand–to–best/*2:

$$\vec{v}_\mathrm{i} = \vec{x}_1 + \mathcal{F}.(\vec{x}_\mathrm{best} - \vec{x}_1) + \mathcal{F}.(\vec{x}_2 - \vec{x}_3) + \mathcal{F}.(\vec{x}_4 - \vec{x}_5) \tag{3}$$

where $\vec{v}_i$ is the donor vector corresponding to *i*th genome. $\vec{x}_1 \neq \vec{x}_2 \neq \vec{x}_3 \neq \vec{x}_4 \neq \vec{x}_5$ are five randomly selected vectors from the population. $\mathcal{F}$ is the scaling factor used to control the amplification of difference vector. The effect of different mutation strategies on the performance of DE has been studied in Ref. 9. If the generated mutant vector is out of the search boundary, a repair operator is used to make $\vec{v}_\mathrm{i}$ back to the feasible region. Different strategies have been proposed to repair the out of bound individuals. In this article, if the *j*th element of the *i*th mutant vector, i.e. v_{ij}, is out of the search region $[lb_j, ub_j]$, then it is repaired as follows:

$$v_{ij} = \begin{cases} \dfrac{x_{ij} + lb_j}{2} & if\ v_{ij} < lb_j \\ \dfrac{x_{ij} + ub_j}{2} & if\ v_{ij} > ub_j \end{cases} \tag{4}$$

where x_{ij} is the *j*th element of the *i*th target vector.

2.3. *Crossover*

To introduce diversity to the population of genomes, DE utilizes a crossover operation to combine the components of target vector $\vec{X}_i$ and donor vector $\vec{V}_i$, to form the trial vector $\vec{U}_i$. Two types of crossover are commonly used in the DE community, which are called *binomial crossover* and *exponential crossover.* In this paper, we focus our work on *binomial crossover* which is defined as follows[19]:

$$u_{ij} = \begin{cases} v_{ij} & if\ rand_{ij}[0,1] \leq CR\ or\ j = jrand \\ x_{ij} & otherwise \end{cases} \tag{5}$$

where $rand_{ij}[0,1]$ is a random number drawn from a uniform distribution between 0 and 1, *CR* is the *crossover rate* used to control the approximate number of components transferred to trial vector from donor vector. $jrand$ is a random index in the range $[1, D]$, which ensures the transmission of at least one component of donor vector into the trial vector. The reason why we have chosen the binomial crossover over the other crossover types is that its behavior is less sensitive to the problem size.[26]

2.4. *Selection*

Finally, a *selection* approach is performed on vectors to determine which vector ($\vec{X}_i$ or $\vec{U}_i$) should be survived in the next generation. The most fitted vector is chosen to be the member of the next generation as follows:

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G} & if\ f(\vec{X}_{i,G}) \leq f(\vec{U}_{i,G}) \\ \vec{X}_{i,G} & otherwise \end{cases} \tag{6}$$

Different variations of DE are specified with a general convention **DE/x/y/z**, where **DE** stands for "Differential Evolution", **x** represents a string denoting the base vector to be perturbed, **y** is the number of difference vectors considered for perturbation of **x**, and **z** stands for the type of crossover being used (exponential or binomial).[19] Algorithm 1 shows a sample pseudo-code for DE.

Algorithm 1. Pseudo-code for DE with binomial crossover

```
1.  Setting parameters
2.  Randomly initialize population in the D-dimensional search space
3.  repeat
4.      for each genome i in the population do
5.          select three mutually exclusive random genomes $\vec{x}_1 \neq \vec{x}_2 \neq \vec{x}_3$
6.          generate a donor vector according to Eq. (2)
            $\vec{v}_i = \vec{x}_1 + \mathcal{F} . (\vec{x}_2 - \vec{x}_3)$
7.          repair $\vec{v}_i$ if it violates the boundary conditions
8.          jRand = a random integer in the range of [1, D]
9.          generate a trial vector $\vec{u}_i$ using binomial crossover by Eq. (4)
            $u_{ij} = \begin{cases} v_{ij} & if\ rand_{ij}[0,1] \leq CR\ or\ j = jrand \\ x_{ij} & otherwise \end{cases}$
10.         evaluate the candidate $\vec{u}_i$
11.         replace $\vec{x}_i$ with $\vec{u}_i$, if fitness of $\vec{u}_i$ is better than fitness of $\vec{x}_i$
12.     end-for
13. until a termination condition is met
```

DE has several advantages that make it a powerful tool for optimization tasks[19]: Specifically, (1) DE has a simple structure and is easy to implement; (2) despite its simplicity, DE exhibits a high accuracy; (3) the number of control parameters in DE are very few (i.e. *NP*, *F* and *CR*); (4) due to its low space complexity, DE is suitable for handling large scale problems.

3. Related Works

Since the inception of DE, several improvements have been proposed to enhance the performance of DE. In the rest of this section, we will examine the current studies and advances in the literature in seven major categories.

3.1. *Changing the initialization pattern of the DE*

It has been acknowledged that the initial population of DE has a great influence on its performance.[27] Most of the studies in the literature have generated the initial population of DE according to a uniform random distribution. However, some researchers have tried to accelerate the convergence speed and solution accuracy of DE by applying other types of initialization methods. For example, Rahnamayan *et al.*[28] used opposition-based learning for generating the initial population in DE. Ali *et al.*[27] proposed two initialization methods for DE based on quadratic interpolation and nonlinear simplex method. Both approaches reported a significant improvement over the basic DE.

3.2. *Adjusting the control parameters of DE*

Several attempts have been done in the literature to establish a balance between the exploration and exploitation ability of DE by adjusting its control parameters. In this subsection, we examine three types of parameter adjustment in DE.

3.2.1. *DE with constant or random parameters*

The first group of methods has tried to determine an exact value or a range of values for the parameters of DE (i.e. *F* and *CR*). This class of studies contain strategies in which the value of DE parameters is either constant during the search or is selected from a pre-defined interval in a random manner. Storn and Price[25] suggested a constant range of values for *NP*, *F* and *CR*. According to their experiments, a reasonable value for *NP* is in the range of $5 \times D$ to $10 \times D$ where *D* is the dimensionality of the problem. *F* should be chosen from [0.5, 1] and a good first choice for *CR* is either 0.9 or 1. Das *et al.*[29] proposed a scheme for adjusting the scaling factor *F*, in which the value of *F* varies during the search process in a random manner. In their approach, the value of *F* is chosen randomly within the range [0.5, 1]. Brest *et al.*[21] introduced an algorithm, called jDE, which adjusts the values of *CR* and *F* for each individual separately. They used a random mechanism to generate new values for *F* and *CR* according to a uniform distribution in the range of [0.1, 1.0] and [0.0, 1.0] respectively.

3.2.2. *DE with time-varying parameters*

Apart from DE with constant or random parameters value, another option is to change the value of the parameters as a function of time or iteration number. An example of such strategy is the work by Das *et al.*[29] They introduced a linearly decreasing scaling factor. In their method, the value of *F* is reduced from an initial value ($\mathcal{F}_{\max}$) to a final value ($\mathcal{F}_{\min}$) according to the following scheme[29]:

$$\mathcal{F}_{\text{iter}} = (\mathcal{F}_{\max} - \mathcal{F}_{\min}) \times \frac{(\text{iter}_{\max} - \text{iter})}{\text{iter}_{\max}} \tag{7}$$

where $\mathcal{F}_{\text{iter}}$ is the value of *F* in the current iteration, $\mathcal{F}_{\max}$ and $\mathcal{F}_{\min}$ are the upper and lower value of *F*, respectively. $\text{iter}_{\max}$ is the maximum number of iterations. Higher value of *F* enables the genomes of the population to explore wide areas of the search space during the early stages of the optimization. Moreover, the decreasing scheme for *F* allows the movements of trial solutions in a relatively small region of the search space around the suspected global optimum, at final stages of the search process.

3.2.3. *DE with adaptive parameters*

The last class of methods contains strategies which adjust the value of the DE parameters according to the state of the algorithm. These methods often control the search process via one or more feedbacks. For example Liu and Lampinen[30] proposed a fuzzy adaptive variant of DE, named FDE, for adaptive selection of value of DE parameters. They used a fuzzy system consisting of "9 × 2" rules for dynamic adjustment of F and CR. Each rule of the system has two inputs and one output. Parameter vector change magnitude (PC) and function value change (FC) are input variables of the system, and the value of F or CR is the output variable of the system. Each fuzzy variable has three fuzzy sets: SMALL, MIDLLE and BIG. Different combinations of input variables are used to determine the value of F and CR. For instance, a typical rule of their system is: IF (PC is small) and (FC is big) Then (CR is big). Qin *et al.*[22] proposed a self-adaptive DE, called SaDE, in which the control parameters and trial vector generation strategies are adaptively adjusted based on their previous performance in generating promising solutions. In Ref. 31, Zhang and Sanderson proposed JADE where the values of F and CR are sampled from a normal distribution and a Cauchy distribution at individual level, respectively. In JADE, information from the most recent successful F and CR are used to set the new F and CR. In this paper, two different feedbacks are used to monitor the search progress of DE, one at population level and another at genome level, and adjust the parameter CR, accordingly.

3.3. *Adapting the selection of mutation strategies in DE*

There exist various methods in the literature that have used strategy adaptation for improving the performance DE. For instance Gong *et al.*[32] employed the *probability matching* technique for strategy adaptation in DE. In their approach, at each generation, a mutation strategy is selected for each parent from a strategy pool of four mutation schemes, i.e. "DE/rand/1", "DE/rand/2", "DE/rand-to-best/2" and "DE/current-to-rand/1", based on its probability. Afterwards, the *relative fitness improvement*, which is calculated as the difference of the fitness of the offspring with that of its parent, is gathered in order to update the probability of each mutation strategy. Mallipeddi *et al.*[33] introduced an ensemble of mutation strategies and control parameters of DE, called EPSDE. EPSDE contains two separate pools: a pool of distinct trial vector generation strategies (with "DE/rand/1", "DE/best/2" and "DE/current-to-rand/1") and a pool of values for the control parameters $F \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and $CR \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. In EPSDE, successful combinations of mutation strategies and parameters values are used to increase the probability of generating promising offspring. Wang *et al.*[34] proposed a Composite DE (CoDE) which combines different trial vector generation strategies with some control parameter settings. In CoDE, they constituted a mutation strategy pool (with "DE/rand/1", "DE/rand/2" and "DE/current-to-rand/1") and a parameter candidate pool (with "$F = 1.0$, $CR = 0.1$", "$F = 1.0$, $CR = 0.9$" and "$F = 0.8$, $CR = 0.2$"). At each generation, three offspring, with randomly chosen parameter settings from parameter pool, are generated for each target vector. Then, the best generated offspring is transferred to the next generation, if it is fitter than its parent. In this work we also use strategy adaptation to improve the performance of DE for function optimization.

3.4. *Hybridizing DE with other operators*

A group of studies have combined DE with other optimization methods. For example, Sun *et al.*[35] proposed a hybrid algorithm of differential evolution and Estimation of Distribution Algorithm (EDA), which they have called DE/EDA. In DE/EDA, local information provided by DE is combined with global information extracted by the EDA. Kazemi *et al.*[36] proposed a bi-population hybrid collaborative model of crowding-based differential evolution (CDE) and PSO, namely CDEPSO, for dynamic optimization problems. In CDEPSO, a population of genomes is responsible for locating several promising areas of the search space and keeping diversity throughout the run using CDE. Another population is used to exploit the area around the best found position using the PSO.

3.5. *Utilizing multi-population scheme*

Several studies have confirmed that utilizing multi-population scheme, instead of single-population; can improve the performance of basic DE. For example, Halder *et al.*[37] presented a cluster-based differential evolution with external archive, called CDDE_Ar, for dynamic optimization problems. In CDDE Ar, the entire population is partitioned into several sub-populations according to the spatial locations of the trial solutions. Each sub-population then exploits its respective region using “DE/best/1/bin”.

3.6. *Designing new types of mutation, crossover and selection*

Another group of studies has been focused on designing new mutation, crossover and selection operators. Zhang and Sanderson[31] proposed a new mutation operator named “DE/current-to-pbest”, which is a generalization of “DE/current-to-best”, to establish a balance between the greediness of the mutation and the diversity of the population. Wang *et al.*[20] embedded quantization orthogonal crossover (QOX) with DE/rand/1/bin to enhance the search ability of DE. Das *et al.*[38] introduced a modified selection mechanism to the classical DE. In this work, the probability of accepting the inferior solutions is dynamically altered with iterations via the simulated annealing concepts.

3.7. *Using local neighborhood topologies in DE*

Apart from the attempts for designing new mutation operators, a number of studies have investigated the effect of local communication topologies on the performance of DE. Das *et al.*[39] proposed DE with global and local neighborhoods, called DEGL, to improve the DE/target-to-best/1/bin mutation scheme. In DEGL, genomes of the population are arranged in a ring topology. Each parameter vector then shares information about good regions of the search space with two of its immediate neighbors. This way, the information about the best position of each neighborhood is spread through the population, which decreases the chance of entrapment in local optima.

The present study focuses on parameter adjustment (i.e. *CR*) as well as strategy adaptation to improve the performance of DE.

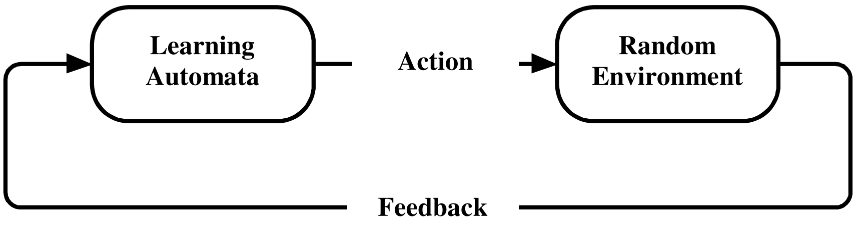


Fig. 2. The interaction between learning automata and environment.

4. Learning Automata

Learning automaton (LA) is an adaptive decision-making device that learns the optimal action out of a set of finite actions through repeated interactions with a random environment.[40,41] At each stage, LA chooses an action, among a set of finite actions, based on a probability distribution over the action-set and apply that to the environment. Then, a feedback is received from the environment by the automaton which is used to update the probabilities of actions. After a certain number of stages, LA is able to select the optimal policy. Interaction between LA and its environment is depicted in Fig. 2.

In this paper, we use Variable Structure Learning Automata (VSLA) to improve the efficiency of DE. VSLA is a type of LA in which probabilities of the actions are updated at each iteration. VSLA can be defined as a quadruple $\{\alpha, \beta, p, T\}$, which $\alpha = \{\alpha_1, \ldots, \alpha_r\}$ is set of automata actions, $\beta = \{\beta_1, \ldots, \beta_m\}$ is the set of automata inputs, $p = \{p_1, \ldots, p_r\}$ is the probability vector corresponds to each action and $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ is the learning algorithm where n is the current iteration of the automata. Various environments can be modeled by β.

In an S-model environment, the output of the environment is a continuous random variable that assumes values in the interval [0, 1]. The automaton in this model updates its action probabilities according to the following equations:

$$p_i(n+1) = p_i(n) - \beta(n) b p_i(n) + [1-\beta(n)] a (1 - p_i(n)), \quad i = j \tag{8}$$

$$p_j(n+1) = p_j(n) - \beta(n)\left[\frac{b}{(r-1)} - b p_j(n)\right] + [1-\beta(n)] a p_j(n), \quad i \neq j \tag{9}$$

where parameters a and b determine reward and penalty in the range of [0, 1].

In a P-model environment, the output of the environment is a binary number with 0 for “desirable” and 1 for “undesirable” response. In this model, LA updates its action probabilities according to following equations:

$$p_j(k+1) = \begin{cases} p_j(k) + a(1 - p_j(k)), & \text{if } i = j \\ p_j(k)(1-a), & \text{if } i \neq j \end{cases} \tag{10}$$

$$p_j(k+1)=\begin{cases} p_j(k)+a(1-b), & \text{if } i=j \\ \dfrac{b}{r-1}+(1-b)p_j(k), & \text{if } i\neq j \end{cases} \tag{11}$$

It is also here that parameters a and b determine reward and penalty in the range of [0, 1]. Three linear reinforcement schemes can be obtained by selecting different values for the reward and penalty parameters. Table 1 indicates different linear reinforcement schemes.

LA have been successfully applied to a number of applications including image processing,[42,43] pattern recognition,[44] wireless sensor networks,[45] parameter adaption,[12,16] function optimization,[46] multi objective optimization,[47] dynamic optimization,[48] Sampling from Complex Networks,[49] graph problems,[50,51] and information retrieval.[52]

Table 1. Different linear reinforcement schemes in learning automata.

Scheme	Parameters Value
L_{RI} (linear reward inaction)	$b=0$
L_{RP} (linear reward penalty)	$a=b$
$L_{R\varepsilon P}$ (linear reward ε-penalty)	$a>>b$

5. Proposed Algorithms

In this section, two classes of LA based approaches for adaptive selection of parameter *CR* and mutation strategy in DE are introduced. The proposed DE algorithms extend the general principles of the standard DE with an extra section for the process of learning the optimal mutation strategy and value for *CR* using LA. In the first class of algorithms, a Group Learning Automata Based Differential Evolution (GLADE) is introduced in which the selected mutation strategy and *CR* are applied to all genomes of the population. Conversely, in the second class, an Individually Learning Automata Based Differential Evolution (ILADE) is proposed in which the mutation strategy and *CR* are chosen for each genome, separately. In both classes we define two types of LA: a LA_{scheme}, which is responsible for selecting an appropriate mutation strategy and a LA_{CR}, which is used to adjust *CR*. The LA_{scheme} and LA_{CR} contain n admissible actions corresponding to mutation strategy and *CR* value, respectively. The LAs in both classes of algorithms have the same characteristics. In the rest of this section, GLADE and ILADE are further explained in detail.

5.1. *Group learning automata based differential evolution*

In this approach, genomes of the population share the same mutation strategy and value for their parameter *CR*. The GLADE approach contains two learning automata (LA_{scheme}

Algorithm 2. Pseudo-code for GLADE

```
1.  Setting parameters $a$, $b$, $F$, population_size
2.  Define learning automata $LA_{scheme}$, $LA_{CR}$ for selecting offspring generation scheme and
    crossover probability $CR$, respectively.
3.  Randomly initialize the population in the D-dimensional search space
4.  repeat
5.      choose an action for each automata according to its probability
        vector
6.      Evolve population of genomes according to selected actions
7.      Set $\beta(n) = 1 - \frac{\text{Number of improved genomes since last iteration}}{\text{population\_size}}$ and update each
        automaton's probability vector using Eqs. (8), (9)
8.  until a termination condition is met
```

and LA_{CR}) that adapt trial vector generation scheme and parameter CR at population level. Algorithm 2 shows the general procedure of GLADE.

At each iteration, LA_{scheme} and LA_{CR} select a trial vector generation strategy and a CR, according to their probability vectors. Then, the population of genomes is evolved by chosen actions and the fraction of improved genomes in the current iteration is used as a feedback to modify the probability vector of each learning automata.

5.2. *Individually learning automata based differential evolution*

In this class of algorithms, each genome of the population adjusts its own mutation strategy and parameter CR separately. In this approach, each genome i of the population uses two LA, i.e. LA_{scheme} and LA_{CR}, to select the mutation strategy and CR, respectively. Hence, the total number of LA in ILADE approach is equal to $NP \times 2$. Pseudo-code for ILADE is presented in Algorithm 3.

In ILADE, in each iteration, a mutation strategy along with a CR is selected for each genome by the corresponding LA. Afterwards, all genomes are evolved using their corresponding trial vector generation strategy and CR value. If the selected mutation strategy and CR for each genome i of the population improve the quality of i, then the related LA will receive a favorable feedback from genome i and the corresponding action of LA_{scheme} and LA_{CR} will be rewarded, otherwise they will be penalized.

Different variations of GLADE and ILADE are obtained by changing the reinforcement schemes in LA_{scheme} and LA_{CR}. These variations are $GLADE_{RI}$, $GLADE_{RP}$, $GLADE_{R\varepsilon P}$, $ILADE_{RI}$, $ILADE_{RP}$ and $ILADE_{R\varepsilon P}$. In these variations when we refer to $ILADE_{R\varepsilon P}$ for example, we mean that both LA_{scheme} and LA_{CR} use $L_{R\varepsilon P}$ reinforcement scheme.

Algorithm 3. Pseudo-code for ILADE

```
1.  Setting parameters $a$, $b$, $F$, population_size
2.  for each genome $i$ in the population do
3.      Define two types of learning automata $LA_{scheme}(i)$, $LA_{CR}(i)$ for
        selecting offspring generation scheme and crossover probability
        $CR$, respectively.
4.  end-for
5.  Randomly initialize population in the D-dimensional search space
6.  repeat
7.      choose an action for each automata according to its probability
        vector
8.      evolve population of genomes according to selected actions
9.      for each genome $i$ in the population do
10.         if $fitness_t(i) < fitness_{t-1}(i)$
11.             Reward selected action of $LA_{scheme}(i)$, $LA_{CR}(i)$ using Eq. (10)
12.         else
13.             Penalize selected action of $LA_{scheme}(i)$, $LA_{CR}(i)$ using Eq. (11)
14.         end-if
15.     end-for
16. until a termination condition is met
```

6. Experimental Setup

6.1. *Benchmark functions used*

There are various benchmark functions in the literature for validating the effectiveness of newly proposed algorithms. Obviously, the No Free Lunch (NFL) theorem states that it is unlikely to develop an omnipotent algorithm for solving all class of problems.[53] Thus, our goal is not to propose a method for solving all benchmark functions, but to improve the efficiency of the DE over a class of benchmark functions. Hence, to study the performance of the proposed automata-based DE variants, a set of experiments was carried out using ten benchmark functions from CEC2005 special session on real-parameter optimization.[54] Among them, five functions are unimodal (F_1–F_5), and the other five functions (F_6–F_{10}) are multimodal functions. Moreover, the performance of the proposed method is also evaluated on one real-life optimization problem called parameter estimation for Frequency-Modulated Sound Waves (FMSW),[55] which will be explained later in sub-section 7.4.

6.2. *Algorithms for comparison*

Several algorithms have been chosen to compare with the proposed approach. These algorithms include two classical DE, three state-of-the-art DE variants and three other well-known methods listed below:

- DE/rand/1/bin ($NP = 100$, $F = 0.5$, $CR = 0.9$).
- DE/rand-to-best/2/bin ($NP = 100$, $F = 0.5$, $CR = 0.9$).
- Self-adapting control parameters in DE (jDE).[21]
- DE with ensemble of parameters and mutation strategies (EPSDE).[33]
- Enhancing the search ability of DE through orthogonal crossover (OXDE).[20]
- Comprehensive learning PSO (CLPSO).[6]
- Global and local real-coded GAs based on parent-centric crossover operators (GL-25).[56]
- Completely derandomized self-adaptation in evolution strategies (CMA-ES).[57]

For Algorithms (3–8), their original settings are used.

6.3. *Parameter settings for LADE variants*

For all variations of the proposed algorithm, the population size *NP* is set to 100. In addition, binomial crossover and parameter $F = 0.5$ are applied. For parameter *CR*, three ranges of discontinues crossover rate are considered as actions for LA_{CR}. Table 2 shows the values of parameter *CR* used in GLADE and ILADE.

Table 2. Different crossover rates used in GLADE and ILADE.

Rate of Crossover	*CR* Values	Initial Probability
Low	{0.1, 0.2}	0.4
Moderate	{0.5}	0.2
High	{0.9, 1.0}	0.4

It is evident that the parameter *CR* plays a critical role on the behavior of DE. A large value of CR can maintain the diversity among the individuals, which is favorable for multi-modal functions. In contrast, a small value of *CR* can make trial vector little different from the target vector. This feature is desirable for optimizing separable problems. The choice of range of values for parameter *CR* is based on the above consideration. Besides, these values have been frequently used in many DE variants, e.g. Ref. 34.

As mentioned before, we use two mutation strategies for LA_{scheme} as follows:

$$DE/rand/1{:}\ \vec{v} = \vec{x}_1 + \mathcal{F}.(\vec{x}_2 - \vec{x}_3) \tag{12}$$

$$DE/rand\text{-}to\text{-}best/2{:}\ \vec{v} = \vec{x}_1 + \mathcal{F}.(\vec{x}_{\text{best}} - \vec{x}_1) + \mathcal{F}.(\vec{x}_2 - \vec{x}_3) + \mathcal{F}.(\vec{x}_4 - \vec{x}_5) \tag{13}$$

Table 3. Different configurations for LA_{scheme} and LA_{CR}.

LA	Learning Scheme	*a*	*b*	Action Set	Initial Probability
LA_{scheme}	L_{RI}	0.01	0.00	{DE/rand/1, DE/random-to-best/2}	{0.5, 0.5}
LA_{scheme}	L_{RP}	0.01	0.01	{DE/rand/1, DE/random-to-best/2}	{0.5, 0.5}
LA_{scheme}	$L_{R\varepsilon P}$	0.01	0.001	{DE/rand/1, DE/random-to-best/2}	{0.5, 0.5}
LA_{CR}	L_{RI}	0.01	0.00	{0.1, 0.2, 0.5, 0.9, 1.0}	{0.2, 0.2, 0.2, 0.2, 0.2}
LA_{CR}	L_{RP}	0.01	0.01	{0.1, 0.2, 0.5, 0.9, 1.0}	{0.2, 0.2, 0.2, 0.2, 0.2}
LA_{CR}	$L_{R\varepsilon P}$	0.01	0.001	{0.1, 0.2, 0.5, 0.9, 1.0}	{0.2, 0.2, 0.2, 0.2, 0.2}

Table 3 represents configurations of LA in different learning schemes.

6.4. *Simulation settings*

All ten benchmark functions (F_1–F_{10}) were tested in 30-dimensions (30D). The maximum number of Function Evaluations (FEs) was set to 300000. All experiments on each function were run 25 times. All the algorithms were implemented and tested using MATLAB R2009a on an Intel Pentium Dual-core Processor with 2.5 GHz CPU and 4 GB of RAM in Windows XP SP3.

7. Results and Discussion

The simulations are organized in three subsections. In the first subsection, the proposed methods are compared to each other and the best performing Learning Automata based DE (LADE) variant is selected for further comparison with other methods. The second subsection compares LADE with alternative DE variants. The computational complexity of the LADE is also presented in the second sub-section. Finally, in the last subsection, the performance of the LADE is compared with other well-known algorithms. The average and standard deviation of the function error values $f(\vec{x}_{best}) - f(\vec{x}_{opt})$ among 25 independent runs recorded for each benchmark function, where $f(\vec{x}_{best})$ the best solution is found by the algorithm in a typical run and $f(\vec{x}_{opt})$ is the optimum value of the test function. Moreover, when comparing with other peer algorithms, the success rate is reported which is the percentage of successful runs among 25 runs. A run is successful if its function error value is smaller than the target error accuracy level ε, which is set to 10^{-6} for test functions F_1–F_5, and 10^{-2} for the other test functions.[54] In order to show the significant statistical difference among the algorithms, a non-parametric statistical test called Wilcoxon rank sum test is conducted for independent samples[58,59] at the 0.05 significance level.

7.1. *Comparison between different LADE variants*

Table 4 presents the function error values obtained by different LADE variants in ten test functions.

Table 4. Average and standard deviation of the function error values of LADE variants over 25 independent runs on ten benchmark functions at 30D, after 300000 FEs.

Fun.	GLADE_{RI}	GLADE_{RP}	$\text{GLADE}_{\text{R}\varepsilon\text{P}}$	ILADE_{RI}	ILADE_{RP}	$\text{ILADE}_{\text{R}\varepsilon\text{P}}$
F_1	0.00E+00±0.00E+00	0.00E+00±0.00E+00	0.00E+00±0.00E+00	0.00E+00±0.00E+00	0.00E+00±0.00E+00	0.00E+00±0.00E+00
F_2	3.98E-08±7.24E-08	1.10E-10±2.06E-10	3.45E-18±7.30E-18	6.16E-10±1.03E-09	6.18E-12±8.75E-12	5.34E-19±1.78E-18
F_3	2.00E+05±1.19E+05	1.16E+05±9.39E+04	1.14E+05±6.61E+04	1.44E+05±1.09E+05	9.19E+04±6.71E+04	6.14E+04±4.28E+04
F_4	1.27E-06±2.20E-06	1.34E-05±2.15E-05	1.57E-02±1.85E-02	2.75E-06±3.26E-06	4.85E-06±5.56E-06	3.00E-08±7.54E-08
F_5	3.78E-02±4.72E-02	5.58E-01±3.59E-01	5.52E-01±6.12E-01	2.38E-02±2.81E-02	4.52E-01±3.84E-01	1.67E-01±1.30E-01
F_6	2.01E-06±7.56E-06	3.36E-05±1.08E-04	3.04E+00±1.39E+00	1.01E-10±2.40E-10	7.39E-07±8.99E-07	3.12E-09±7.75E-09
F_7	2.95E-04±1.47E-03	4.71E-15±6.90E-15	3.94E-04±1.97E-03	6.90E-04±2.41E-03	3.94E-04±1.97E-03	6.90E-04±2.41E-03
F_8	2.09E+01±3.47E-02	2.09E+01±3.70E-02	2.09E+01±5.01E-02	2.09E+01±5.26E-02	2.09E+01±4.89E-02	2.09E+01±3.63E-02
F_9	3.60E-09±1.21E-08	1.26E+01±2.11E+00	1.13E+02±2.14E+01	2.27E-10±3.51E-10	1.17E+01±2.49E+00	1.86E-04±9.31E-04
F_{10}	1.48E+02±1.26E+01	1.51E+02±1.23E+01	1.81E+02±8.62E+00	1.51E+02±9.52E+00	1.50E+02±1.23E+01	1.50E+02±1.24E+01

Table 5. Multiple comparison of mean best fitness, based on Tukey-Kramer method.

Methods	GLADE_{RI}	GLADE_{RP}	$\text{GLADE}_{\text{R}\varepsilon\text{P}}$	ILADE_{RI}	ILADE_{RP}	$\text{ILADE}_{\text{R}\varepsilon\text{P}}$	Score
GLADE_{RI}	0	(+2, –2) **0**	(+5, –2) **+3**	(0, –1) **–1**	(+2, –2) **0**	(0, –2) **–2**	0
GLADE_{RP}	(+2, –2) **0**	0	(+4, 0) **+4**	(0, –2) **–2**	(0, 0) **0**	(0, –2) **–2**	0
$\text{GLADE}_{\text{R}\varepsilon\text{P}}$	(+2, –5) **–3**	(0, –4) **–4**	0	(0, –5) **–5**	(0, –4) **–4**	(0, –5) **–5**	–21
ILADE_{RI}	(+1, 0) **+1**	(+2, 0) **+2**	(+5, 0) **+5**	0	(+2, 0) **+2**	(0, –1) **0**	+10
ILADE_{RP}	(+2, –2) **0**	(0, 0) **0**	(+4, 0) **+4**	(0, –2) **–2**	0	(0, –2) **–2**	0
$\text{ILADE}_{\text{R}\varepsilon\text{P}}$	(+2, 0) **+2**	(+2, 0) **+2**	(+5, 0) **+5**	(+1, 0) **+1**	(+2, 0) **+2**	0	+12

In order to find the best performing algorithm among all LADE variants, Tukey-Kramer multiple comparison[60] has been applied over all test functions. Table 5 shows the result of multiple comparisons in the form of $(b, -w)$. For each cell (row, col) of the table, b and $-w$ represent the number of problems for which LADE_{row} is significantly better than and worse than LADE_{col}, respectively. If the subtraction of b and w (boldface values) is a positive number, it means that LADE_{row} is better than the LADE_{col} and vice versa. In case that this value is equal to zero, none of the algorithms are superior to the other one. The last column of the Table 5 indicates the overall superiority of each method over the other methods, which is obtained by summing the boldface values of each row.

A number of conclusions can be drawn from Table 5:

- $\text{GLADE}_{\text{R}\varepsilon\text{P}}$ is the worst performing algorithm that is outperformed by all the other LADE variants.
- $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ is the best performing algorithm on the set of benchmark functions that is not outperformed by any other LADE variants.
- Overall performance of ILADE versions is better than GLADEs. This is mainly because in ILADE variants each genome of the population adjusts its parameters according to its own progress.

The reason for the superiority of $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ over other variants of ILADE lies in the fact that $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ uses a linear reward ε-penalty reinforcement scheme. As can be seen in Table 5, $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ and ILADE_{RI} are highly competitive. However, $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ can perform slightly better than ILADE_{RI}. The only difference between these two algorithms is in their reinforcement scheme. It can be concluded that the penalty scheme used in $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ is a key feature that can contribute to its better performance.

In the rest of this paper, we compare the performance of $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ with other alternative algorithms.

7.2. *Comparison with other DE variants*

This subsection compares the proposed $\text{ILADE}_{\text{R}\varepsilon\text{P}}$ with other types of DE. Since LADEs use DE/rand/1/bin and DE/rand-to-best/2/bin as their actions for $\text{LA}_{\text{scheme}}$, DE/rand/1/bin and DE/rand-to-best/2/bin are the first candidates for comparison. jDE,[21] EPSDE[33] and OXDE[20] are also chosen as three state-of-the-art DE to compare with the proposed method. Finally, the computational complexity of the proposed method is reported.

7.2.1. *Comparison with DE/rand/1/bin and DE/rand-to-best/2/bin*

Tables 6–8 present the statistical results obtained by DE/rand/1/bin, DE/rand-to-best/2/bin and ILADE$_{R\varepsilon P}$ on a suite of ten benchmark functions. As can be seen from the reported results in Tables 6 and 8, for all tested functions, ILADE$_{R\varepsilon P}$ outperforms DE/rand/1/bin in terms of both rate of convergence and quality of final solution.

DE/rand-to-best/2/bin surpasses ILADE$_{R\varepsilon P}$ only in F_4 (see Tables 7 and 8). In some cases, ILADE$_{R\varepsilon P}$ shows a slower convergence rate than DE/rand-to-best/2/bin, but the difference between the final solutions of ILADE$_{R\varepsilon P}$ and DE/rand-to-best/2/bin is considerable. The obtained results suggest that the proposed automata based approach is a significant improvement over DE/rand/1/bin and DE/rand-to-best/2/bin.

Table 6. Error values of DE/rand/1/bin over 25 independent runs on ten benchmark functions at 30D.

FEs	Function	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
3×10^3	best	1.45E+04	3.38E+04	1.69E+08	4.68E+04	1.56E+04	1.59E+09	2.37E+03	2.09E+01	2.52E+02	3.26E+02
	7	1.77E+04	4.28E+04	2.54E+08	5.50E+04	2.03E+04	2.99E+09	3.06E+03	2.11E+01	2.89E+02	3.79E+02
	13	1.94E+04	4.84E+04	3.07E+08	6.20E+04	2.09E+04	3.92E+09	3.55E+03	2.12E+01	2.96E+02	3.92E+02
	19	2.16E+04	5.21E+04	3.63E+08	6.69E+04	2.20E+04	4.70E+09	3.89E+03	2.12E+01	3.12E+02	4.03E+02
	worst	2.64E+04	5.99E+04	4.52E+08	7.76E+04	2.43E+04	6.51E+09	4.40E+03	2.12E+01	3.26E+02	4.37E+02
	mean	1.96E+04	4.78E+04	3.08E+08	6.13E+04	2.08E+04	3.95E+09	3.49E+03	2.12E+01	2.98E+02	3.88E+02
	std	2.67E+03	7.39E+03	7.62E+07	7.47E+03	2.25E+03	1.32E+09	5.63E+02	7.18E-02	1.82E+01	2.86E+01
3×10^4	best	2.64E+00	2.42E+03	1.30E+07	5.82E+03	3.24E+03	3.11E+03	4.56E+00	2.09E+01	1.75E+02	1.97E+02
	7	5.40E+00	4.44E+03	2.56E+07	8.79E+03	3.85E+03	6.93E+03	8.21E+00	2.10E+01	1.90E+02	2.14E+02
	13	6.42E+00	5.43E+03	2.88E+07	1.07E+04	4.14E+03	1.36E+04	9.51E+00	2.10E+01	2.00E+02	2.19E+02
	19	7.99E+00	7.08E+03	3.78E+07	1.18E+04	4.49E+03	2.09E+04	1.08E+01	2.11E+01	2.07E+02	2.25E+02
	worst	1.23E+01	9.05E+03	4.09E+07	1.58E+04	5.29E+03	3.97E+04	1.29E+01	2.12E+01	2.19E+02	2.35E+02
	mean	6.65E+00	5.62E+03	3.04E+07	1.05E+04	4.13E+03	1.60E+04	9.45E+00	2.10E+01	2.00E+02	2.19E+02
	std	2.02E+00	1.67E+03	7.48E+06	2.25E+03	4.85E+02	1.03E+04	2.13E+00	6.38E-02	1.08E+01	1.02E+01
3×10^5	best	0.00E+00	6.44E-06	7.79E+04	9.01E-04	2.02E-01	2.44E-04	0.00E+00	2.08E+01	8.16E+01	1.42E+02
	7	0.00E+00	1.54E-05	2.83E+05	6.69E-03	3.14E-01	1.64E+00	0.00E+00	2.09E+01	1.08E+02	1.73E+02
	13	0.00E+00	2.84E-05	4.17E+05	1.17E-02	6.52E-01	2.50E+00	0.00E+00	2.09E+01	1.28E+02	1.78E+02
	19	0.00E+00	5.19E-05	6.68E+05	1.86E-02	1.27E+00	2.94E+00	0.00E+00	2.10E+01	1.47E+02	1.84E+02
	worst	0.00E+00	4.01E-04	9.62E+05	1.16E-01	5.73E+00	7.21E+00	1.23E-02	2.10E+01	1.70E+02	1.94E+02
	mean	0.00E+00	5.78E-05	4.83E+05	2.27E-02	9.76E-01	2.48E+00	1.08E-03	2.09E+01	1.29E+02	1.76E+02
	std	0.00E+00	8.49E-05	2.58E+05	3.15E-02	1.13E+00	1.61E+00	3.11E-03	5.94E-02	2.52E+01	1.22E+01

Table 7. Error values of DE/rand-to-best/2/bin over 25 independent runs on ten benchmark functions at 30D.

FEs	Function	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
3×10^3	best	3.67E+03	2.11E+04	8.74E+07	2.21E+04	1.06E+04	1.86E+08	7.58E+02	2.11E+01	2.19E+02	2.64E+02
	7	5.06E+03	2.67E+04	1.07E+08	3.57E+04	1.21E+04	2.69E+08	1.04E+03	2.12E+01	2.71E+02	2.93E+02
	13	6.15E+03	3.11E+04	1.36E+08	4.17E+04	1.32E+04	3.45E+08	1.14E+03	2.12E+01	2.79E+02	3.15E+02
	19	7.53E+03	3.59E+04	1.54E+08	4.70E+04	1.50E+04	4.32E+08	1.28E+03	2.12E+01	2.89E+02	3.29E+02
	worst	8.69E+03	4.44E+04	2.18E+08	6.11E+04	1.85E+04	1.23E+09	1.68E+03	2.13E+01	3.04E+02	3.62E+02
	mean	6.24E+03	3.13E+04	1.39E+08	4.17E+04	1.35E+04	3.88E+08	1.17E+03	2.12E+01	2.77E+02	3.12E+02
	std	1.48E+03	5.69E+03	3.53E+07	8.73E+03	2.02E+03	2.06E+08	2.33E+02	4.65E-02	1.94E+01	2.57E+01
3×10^4	best	1.28E-02	3.67E+02	4.52E+06	1.23E+03	1.22E+03	4.82E+01	1.05E+00	2.09E+01	1.71E+02	2.02E+02
	7	1.79E-02	6.91E+02	6.80E+06	2.06E+03	1.51E+03	6.41E+01	1.08E+00	2.10E+01	1.92E+02	2.13E+02
	13	2.60E-02	8.33E+02	8.53E+06	2.95E+03	1.80E+03	7.97E+01	1.10E+00	2.11E+01	2.01E+02	2.24E+02
	19	2.85E-02	9.94E+02	9.32E+06	3.53E+03	2.35E+03	1.84E+02	1.11E+00	2.11E+01	2.10E+02	2.27E+02
	worst	3.93E-02	1.48E+03	1.17E+07	5.05E+03	3.38E+03	8.32E+02	1.23E+00	2.11E+01	2.22E+02	2.50E+02
	mean	2.45E-02	8.59E+02	8.15E+06	2.86E+03	1.92E+03	1.72E+02	1.10E+00	2.11E+01	2.00E+02	2.22E+02
	std	7.70E-03	2.83E+02	1.80E+06	1.00E+03	5.35E+02	2.09E+02	4.00E-02	5.11E-02	1.29E+01	1.21E+01
3×10^5	best	1.14E-28	2.01E-11	8.14E+03	3.30E-07	7.47E-06	3.21E-22	0.00E+00	2.09E+01	1.36E+02	1.66E+02
	7	3.53E-28	6.07E-11	5.58E+04	1.99E-06	8.24E-05	1.97E-20	0.00E+00	2.09E+01	1.51E+02	1.74E+02
	13	4.54E-28	1.06E-10	8.37E+04	5.98E-06	1.53E-04	8.05E-20	0.00E+00	2.10E+01	1.57E+02	1.84E+02
	19	5.05E-28	2.03E-10	1.28E+05	9.98E-06	2.64E-04	2.58E-19	0.00E+00	2.10E+01	1.67E+02	1.87E+02
	worst	6.06E-28	2.97E-10	3.20E+05	1.91E-05	2.63E-03	3.99E+00	2.22E-02	2.10E+01	1.78E+02	2.06E+02
	mean	4.21E-28	1.35E-10	1.05E+05	6.69E-06	3.05E-04	3.19E-01	2.86E-03	2.10E+01	1.59E+02	1.82E+02
	std	1.20E-28	8.87E-11	7.67E+04	5.54E-06	5.29E-04	1.10E+00	6.42E-03	3.87E-02	9.62E+00	9.63E+00

Table 8. Error values of $ILADE_{R\varepsilon P}$ over 25 independent runs on ten benchmark functions at 30D.

FEs	Function	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
3×10^3	best	8.53E+03	2.83E+04	1.46E+08	3.09E+04	1.23E+04	5.47E+08	1.59E+03	2.10E+01	1.95E+02	2.74E+02
	7	1.09E+04	4.17E+04	1.86E+08	5.04E+04	1.53E+04	1.22E+09	1.71E+03	2.11E+01	2.49E+02	3.24E+02
	13	1.20E+04	4.44E+04	2.39E+08	5.77E+04	1.63E+04	1.45E+09	1.89E+03	2.12E+01	2.61E+02	3.41E+02
	19	1.25E+04	4.87E+04	2.55E+08	6.26E+04	1.80E+04	1.69E+09	2.07E+03	2.12E+01	2.57E+02	3.49E+02
	worst	1.41E+04	5.75E+04	3.87E+08	7.77E+04	2.14E+04	2.36E+09	2.40E+03	2.13E+01	2.85E+02	3.92E+02
	mean	1.16E+04	4.44E+04	2.33E+08	5.67E+04	1.66E+04	1.42E+09	1.90E+03	2.12E+01	2.52E+02	3.38E+02
	std	1.29E+03	6.89E+03	5.41E+07	1.10E+04	2.05E+03	4.17E+08	2.42E+02	7.87E-02	2.11E+01	2.24E+01

FEs	Function	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
3×10^4	best	4.08E-01	8.55E+02	2.42E+06	1.95E+03	2.85E+03	7.68E+02	3.52E+00	2.09E+01	7.22E+01	1.89E+02
	7	4.75E-01	1.28E+03	4.28E+06	4.44E+03	3.43E+03	1.19E+03	4.85E+00	2.10E+01	8.40E+01	2.08E+02
	13	5.49E-01	1.66E+03	5.93E+06	5.46E+03	3.61E+03	2.18E+03	6.15E+00	2.10E+01	9.01E+01	2.14E+02
	19	6.66E-01	1.97E+03	6.73E+06	6.19E+03	3.94E+03	2.90E+03	6.88E+00	2.11E+01	9.28E+01	2.23E+02
	worst	8.97E-01	2.60E+03	1.04E+07	1.05E+04	4.62E+03	5.14E+03	8.72E+00	2.11E+01	1.06E+02	2.37E+02
	mean	5.81E-01	1.63E+03	5.65E+06	5.38E+03	3.69E+03	2.30E+03	6.00E+00	2.10E+01	8.90E+01	2.14E+02
	std	1.23E-01	4.74E+02	1.91E+06	1.67E+03	4.85E+02	1.21E+03	1.31E+00	5.70E-02	7.27E+00	1.15E+01
3×10^5	best	0.00E+00	1.02E-22	6.39E+03	4.08E-11	2.56E-02	3.19E-11	0.00E+00	2.09E+01	8.88E-14	1.19E+02
	7	0.00E+00	4.58E-21	3.20E+04	3.48E-09	8.94E-02	3.19E-10	0.00E+00	2.09E+01	1.24E-10	1.47E+02
	13	0.00E+00	2.03E-20	5.21E+04	1.03E-08	1.30E-01	5.98E-10	0.00E+00	2.10E+01	8.72E-10	1.51E+02
	19	0.00E+00	1.19E-19	7.23E+04	2.01E-08	1.91E-01	1.87E-09	0.00E+00	2.10E+01	9.36E-09	1.58E+02
	worst	0.00E+00	8.27E-18	2.24E+05	3.77E-07	4.98E-01	3.83E-08	9.86E-03	2.10E+01	4.65E-03	1.67E+02
	mean	0.00E+00	5.34E-19	6.14E+04	3.00E-08	1.67E-01	3.12E-09	6.90E-04	2.09E+01	1.86E-04	1.50E+02
	std	0.00E+00	1.78E-18	4.28E+04	7.54E-08	1.30E-01	7.75E-09	2.41E-03	3.63E-02	9.31E-04	1.24E+01

Table 9. Average and standard deviation of the function error values of jDE, EPSDE, OXDE and ILADE$_{\text{ReP}}$ over 25 independent runs on ten benchmark functions at 30D, after 300000 FEs. For successful runs, success rates are shown in parentheses.

Fun.	jDE (2006)	EPSDE (2011)	OXDE (2012)	ILADE$_{\text{ReP}}$
F_1	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	6.27E-29 ± 1.14E-28§ (100%)	0.00E+00 ± 0.00E+00 (100%)
F_2	3.59E-07 ± 3.25E-07§ (92%)	9.77E-26 ± 3.16E-25† (100%)	7.33E-05 ± 8.20E-05§	5.34E-19 ± 1.78E-18 (100%)
F_3	1.55E+05 ± 1.06E+05§	6.93E+05 ± 2.94E+06§	5.39E+05 ± 2.53E+05§	6.14E+04 ± 4.28E+04
F_4	6.70E-02 ± 2.32E-01§	3.88E+01 ± 1.18E+02§ (4%)	4.04E+00 ± 7.94E+00§	3.00E-08 ± 7.54E-08 (100%)
F_5	4.65E+02 ± 3.65E+02§	1.42E+03 ± 7.12E+02§	2.71E+01 ± 4.44E+01§	1.67E-01 ± 1.30E-01
F_6	2.13E+01 ± 2.46E+01§	3.18E-01 ± 1.10E+00§ (92%)	1.11E+00 ± 1.82E+00§ (72%)	3.12E-09 ± 7.75E-09 (100%)
F_7	1.27E-02 ± 7.00E-03§ (80%)	1.62E-02 ± 1.47E-02§ (52%)	1.11E-02 ± 8.72E-03§ (56%)	6.90E-04 ± 2.41E-03 (100%)
F_8	2.09E+01 ± 4.78E-02$^{\approx}$	2.09E+01 ± 4.79E-02$^{\approx}$	2.09E+01 ± 5.95E-02$^{\approx}$	2.09E+01± 3.63E-02
F_9	0.00E+00 ± 0.00E+00† (100%)	3.97E-02 ± 1.98E-01§ (96%)	1.54E+01 ± 4.16E+00§	1.86E-04 ± 9.31E-04 (100%)
F_{10}	5.71E+01 ± 1.03E+01†	4.79E+01 ± 1.35E+01†	6.69E+01 ± 6.42E+01†	1.50E+02 ± 1.24E+01
†	2	2	1	
§	6	6	8	
≈	2	2	1	

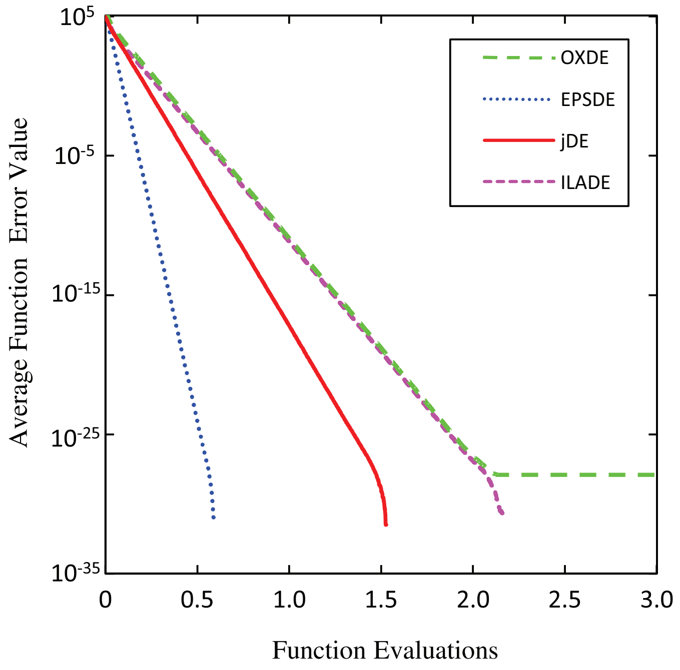
''†'' and ''§'' indicate a 0.05 level of significance by Wilcoxon rank sum test. ''†'', ''§'' and ''≈'' denote that the performance of the corresponding algorithm is better than, worse than, and similar to that of LADE, respectively.

7.2.2. *Comparison with three state-of-the-art DE variants*

The results of ILADE$_{R\varepsilon P}$ and three other state-of-the-art DE variants are given in Table 9. The last three rows of Table 9 summarize the simulation results.

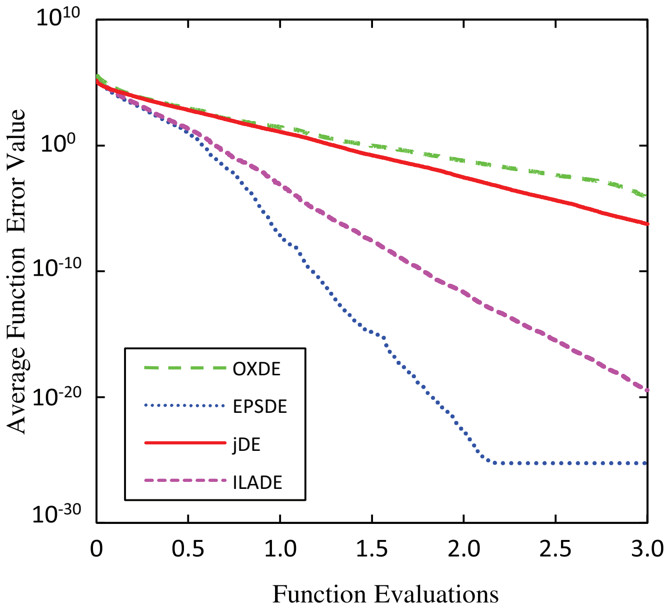
Moreover, the average function error value curves of jDE, EPSDE, OXDE and ILADE$_{R\varepsilon P}$ over 25 independent runs for test functions F_1–F_{10} are depicted in Figs. 3 and 4.

On unimodal test functions (i.e. F_1–F_5), it is observed that ILADE$_{R\varepsilon P}$ is very efficient. It outperforms other methods on three test functions (i.e. F_3–F_5) (see Table 9). Moreover, ILADE$_{R\varepsilon P}$ has the fastest convergence rate on F_3–F_5 (see Fig. 3). jDE and OXDE cannot outperform ILADE$_{R\varepsilon P}$ on any unimodal functions and EPSDE surpasses ILADE$_{R\varepsilon P}$ only in F_2. On the other hand, on multimodal functions (i.e. F_6–F_{10}), ILADE$_{R\varepsilon P}$ is significantly better on F_6 and F_7. On F_8, ILADE$_{R\varepsilon P}$ is the second best method. F_{10} is the only test function that ILADE$_{R\varepsilon P}$ fails to reach to global minimum with a competitive accuracy. A closer look at Table 9 shows that EPSDE can perform better than ILADE$_{R\varepsilon P}$ on functions F_2 and F_{10}. The possible reason is that EPSDE uses a mutation pool with three strategies DE/best/2/bin, DE/rand/1/bin and DE/current-to-rand/1/bin. On the one hand,

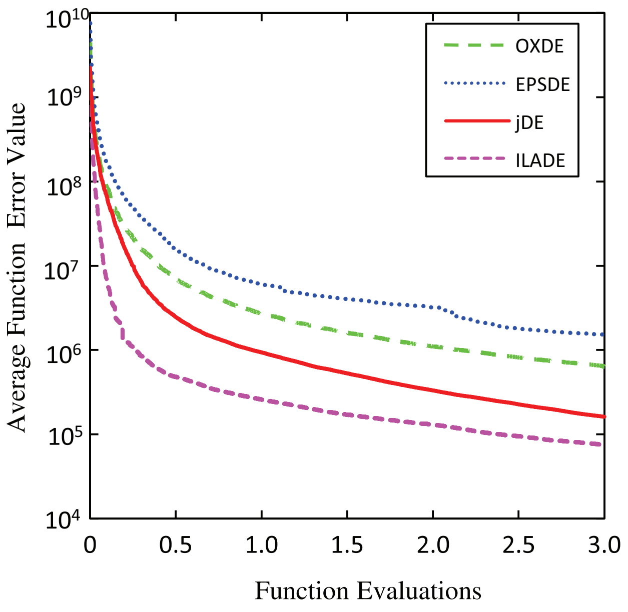


(a)

Fig. 3. (Color online) Evolution of the mean error values versus the number of function evaluations for OXDE, EPSDE, jDE and ILADE on test functions (a) F_1, (b) F_2, (c) F_3, (d) F_4 and (e) F_5.

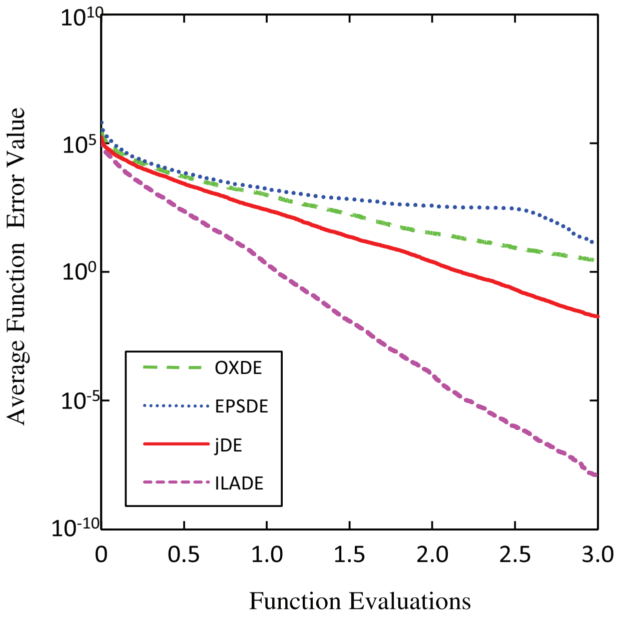


(b)

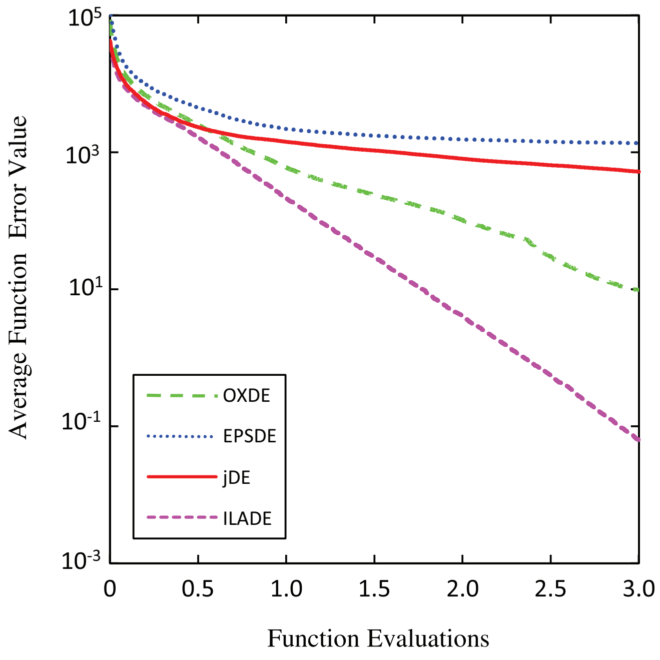


(c)

Fig. 3. (*Continued*)

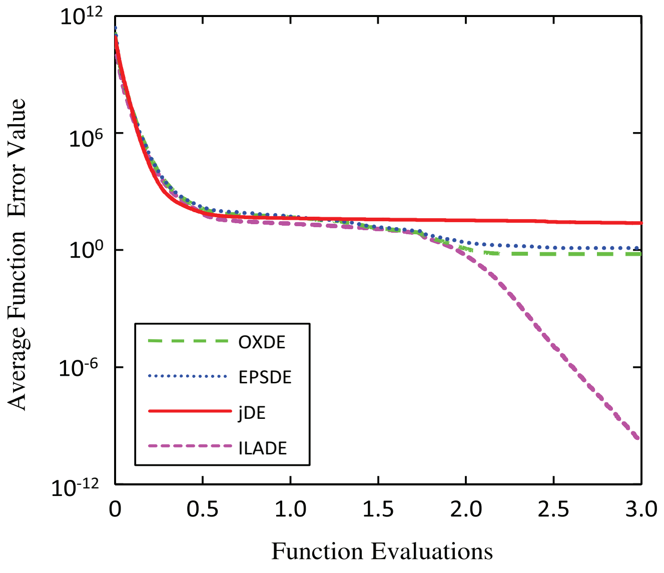


(d)

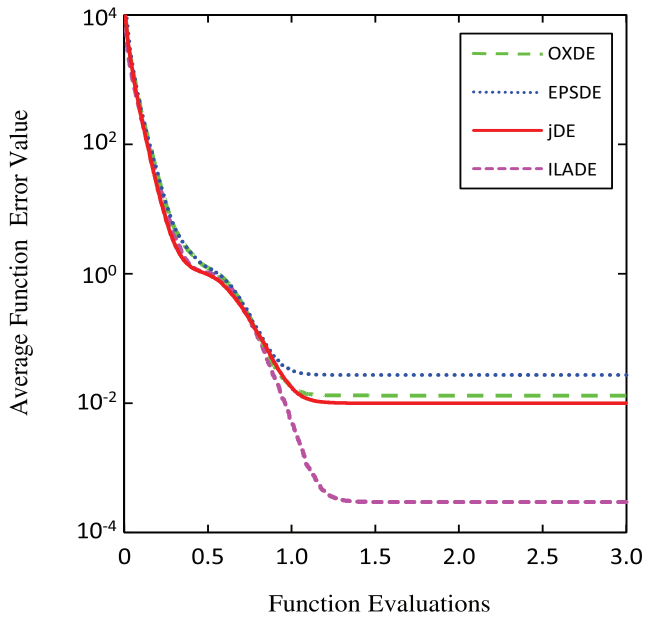


(e)

Fig. 3. (*Continued*)

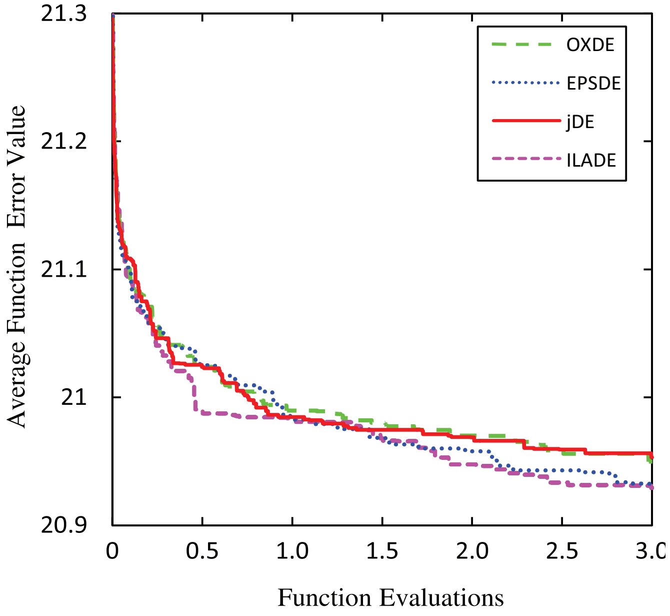


(a)

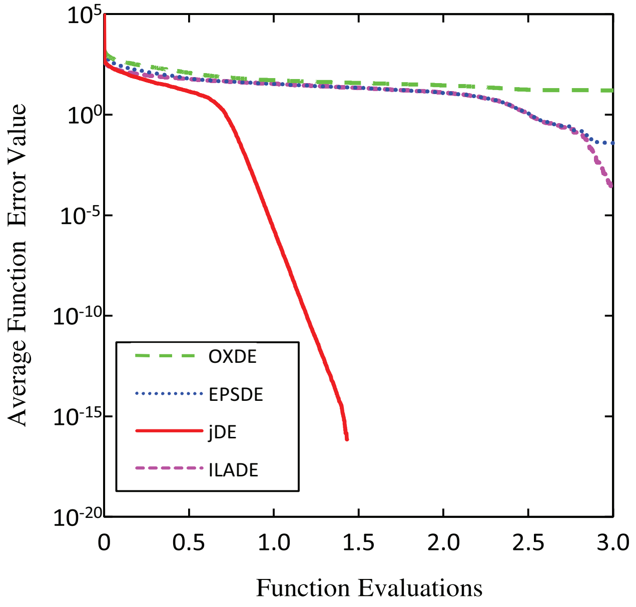


(b)

Fig. 4. (Color online) Evolution of the mean error values versus the number of function evaluations for OXDE, EPSDE, jDE and ILADE on test functions (a) F_6, (b) F_7, (c) F_8, (d) F_9 and (e) F_{10}.

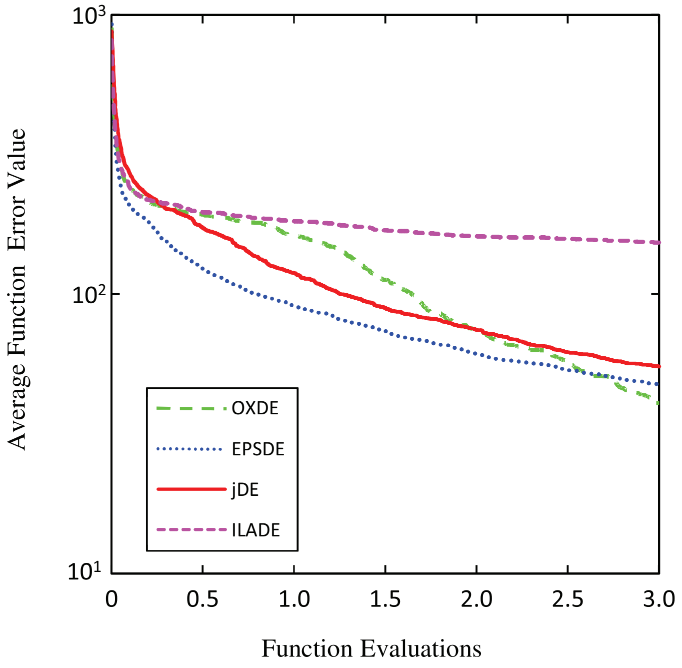


(c)



(d)

Fig. 4. (*Continued*)



(e)

Fig. 4. (*Continued*)

the first two strategies are robust. On the other hand, the third strategy is rotation-invariant. In addition, the value of parameter F is also adapted for each vector during the search process. These characteristics can contribute toward better performance of EPSDE on functions F_2 and F_{10}. Finally, simulation results reveal that the ILADE$_{R\varepsilon P}$ is a viable approach for the optimization of real-parameter functions.

7.2.3. *Computational complexity of ILADE$_{R\varepsilon P}$*

In order to evaluate the runtime complexity of our proposed algorithm, we use the method based on Ref. 54. Table 10 reports the computational complexity of ILADE$_{R\varepsilon P}$ and compares it with EPSDE. It should be noted that the reason why EPSDE was selected as a peer algorithm for comparison is that EPSDE uses both parameter adjustment and strategy adaptation simultaneously, similar to ILADE$_{R\varepsilon P}$. In Table 10, T_0 is the computing time needed for a test program depicted in Fig. 5, T_1 is the computing time of the function F_3 for 200 000 evaluations of a certain dimension D and $\hat{T}_2$ is the average computing time for the 5 runs of the algorithm with 200 000 evaluations of the same D dimensional function F_3.

From Table 10, it is clear that our proposed ILADE$_{R\varepsilon P}$ is more computationally efficient than EPSDE.

Table 10. Computational complexity of ILADE$_{R\varepsilon P}$ in comparison with EPSDE.

Dim	T_0	T_1	$\hat{T}_2$		$(\hat{T}_2 - T_1) / T_0$	
Method	–	–	EPSDE	ILADE$_{R\varepsilon P}$	EPSDE	ILADE$_{R\varepsilon P}$
10	0.17	12.09	26.70	25.28	81.17	72.82
30	0.17	15.00	41.14	27.51	153.76	73.58
50	0.17	17.61	66.23	31.55	286	82

```
for i = 1:1000000
    x = (double) 5.55;
    x = x + x; x = x./2; x = x*x; x = sqrt(x); x = ln(x); x = exp(x); y = x/x;
end
```

Fig. 5. The test program for computing T_0.

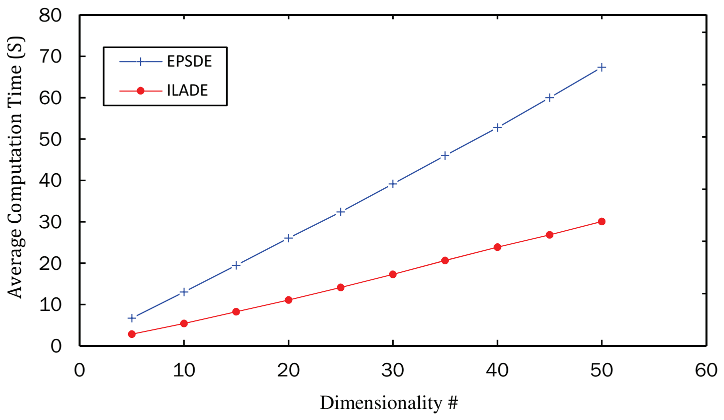


Fig. 6. The effect of dimensionality on the computation time consumed by ILADE$_{R\varepsilon P}$ and EPSDE on benchmark function F_1.

In order to evaluate the runtime complexity of our proposed algorithm, we use the method based on Ref. 54. Table 10 reports the computational complexity of ILADE$_{R\varepsilon P}$ and compares it with EPSDE. It should be noted that the reason why EPSDE was selected as a peer algorithm for comparison is that EPSDE uses both parameter adjustment and strategy adaptation simultaneously, similar to ILADE$_{R\varepsilon P}$. In Table 10, T_0 is the computing time needed for a test program depicted in Fig. 5, T_1 is the computing time of the function F_3 for 200 000 evaluations of a certain dimension D and $\hat{T}_2$ is the average computing time for the 5 runs of the algorithm with 200 000 evaluations of the same D dimensional function F_3.

It can be seen in Fig. 6 that ILADE$_{R\varepsilon P}$ has a smaller increasing rate than the EPSDE. Moreover, ILADE$_{R\varepsilon P}$ is able to save computation time by 43.52% on average, in comparison with EPSDE.

Table 11. Average and standard deviation of the function error values of CMA-ES, CLPSO, GL-25 and ILADE$_{R\varepsilon P}$ over 25 independent runs on ten benchmark functions at 30D, after 300000 FEs. For successful runs, success rates are shown in parentheses.

Fun.	CMA-ES (2001)	CLPSO (2006)	GL-25 (2008)	ILADE$_{R\varepsilon P}$
F$_1$	1.69E-25 ± 3.63E-26§ (100%)	0.00E+00 ± 0.00E+00$^{\approx}$ (100%)	1.13E-27 ± 2.33E-27§ (100%)	0.00E+00 ± 0.00E+00 (100%)
F$_2$	6.14E-25 ± 1.89E-25† (100%)	8.15E+02 ± 1.68E+02§	6.30E+01 ± 1.32E+02§	5.34E-19 ± 1.78E-18 (100%)
F$_3$	5.28E-21 ± 1.86E-21† (100%)	1.54E+07 ± 3.20E+06§	1.94E+06 ± 8.17E+05§	6.14E+04 ± 4.28E+04
F$_4$	6.94E+05 ± 1.54E+06§	6.66E+03 ± 1.37E+03§	9.01E+02 ± 4.06E+02§	3.00E-08 ± 7.54E-08 (100%)
F$_5$	3.17E-10 ± 5.88E-11† (100%)	3.91E+03 ± 3.63E+02§	2.48E+03 ± 2.17E+02§	1.67E-01 ± 1.30E-01
F$_6$	7.97E-01 ± 1.62E+00§ (80%)	7.23E+00 ± 9.71E+00§	2.14E+01 ± 1.34E+00§	3.12E-09 ± 7.75E-09 (100%)
F$_7$	3.45E-03 ± 7.07E-03§ (88%)	4.66E-01 ± 6.07E-02§	1.89E-02 ± 1.85E-02§ (40%)	6.90E-04 ± 2.41E-03 (100%)
F$_8$	2.03E+01 ± 5.81E-01†	2.09E+01 ± 5.31E-02$^{\approx}$	2.09E+01 ± 6.05E-02$^{\approx}$	2.09E+01± 3.63E-02
F$_9$	4.31E+02 ± 9.04E+01§	0.00E+00 ± 0.00E+00† (100%)	2.40E+01 ± 5.10E+00§	1.86E-04 ± 9.31E-04 (100%)
F$_{10}$	4.60E+01 ± 1.26E+01†	9.90E+01 ± 1.12E+01§	1.52E+02 ± 5.63E+01§	1.50E+02 ± 1.24E+01
†	5	1	0	
§	5	7	9	
≈	0	2	1	

"†" and "§" indicate a 0.05 level of significance by Wilcoxon rank sum test. "†", "§" and "≈" denote that the performance of the corresponding algorithm is better than, worse than, and similar to that of LADE, respectively.

7.3. *Comparison with other optimization algorithms*

In this subsection, ILADE$_{R\varepsilon P}$ is compared to three well-known EAs, namely, CMA-ES,[57] CLPSO[6] and GL-25.[56] Table 11 summarizes the simulation results.

From Table 11, it can be concluded that ILADE$_{R\varepsilon P}$ is clearly better than CLPSO and GL-25, and is very competitive with CMA-ES.

7.4. *Real-life optimization problem*

The parameter estimation for FMSW is an important real-life engineering problem which plays a key role in several modern music systems[55]. The parameter estimation for FMSW is a six-dimensional optimization problem in which the goal of optimization is to estimate the $\vec{X} = \{a_1, \omega_1, a_2, \omega_2, a_3, \omega_3\}$ to minimize the following fitness function:

$$f = \sum_{t=0}^{100} (y(t) - y_0(t))^2 \tag{14}$$

In above equation, $y(t)$ is the generated sound wave and $y_0(t)$ is the target sound wave which are calculated as follows:

$$y(t) = a_1 \times \sin(\omega_1 . t . \theta + a_2 \times \sin(\omega_2 . t . \theta + a_3 \times \sin(\omega_3 . t . \theta))) \tag{15}$$

$$y_0(t) = (1.0) \times \sin\Big((5.0).t.\theta + (-1.5) \times \sin\big((4.8).t.\theta + (2.0) \times \sin((4.9).t.\theta)\big)\Big) \quad (16)$$

where $\theta = 2\pi/100$ and the parameters are bounded in [–6.4 6.35].

This problem is a highly complex multimodal instance with minimum value $f_{\min} = 0$. The results of ILADE$_{\mathbf{R\varepsilon P}}$, EA-DE-Mimetic,[61] SAMODE,[62] and DE-Λ_{Cr}[63] on the FMSW have been reported in Table 12. For ILADE$_{\mathbf{R\varepsilon P}}$, the results were reported based on 25 independent runs. Results of the other methods were directly taken from their respective papers.

As can be seen in Table 12, ILADE$_{\mathrm{R\varepsilon P}}$ has a good performance on the estimation of FMSW.

Table 12. Results of different methods for the parameter estimation of FMSW.

FEs	Function value	EA-DE-Mimetic (2011)	SAMODE (2011)	DE-Λ_{Cr} (2011)	ILADE$_{\mathrm{R\varepsilon P}}$
5×10^4	best	1.1674E-11	2.5397E-11	1.6147E-10	**2.5354E-18**
	median	1.0167E+01	8.9616E-06	**3.8779E-09**	1.8904E-03
	worst	1.5440E+01	**1.2547E+01**	1.4813E+01	1.3210E+01
	mean	7.6022E+00	3.0769E+00	**2.1870E+00**	3.0669E+00
	std	6.0205E+00	5.0495E+00	4.6326E+00	4.1835E+00
1×10^5	best	1.1674E-11	1.6373E-30	3.5443E-12	**0.0000E+00**
	median	1.2912E-09	6.9955E-24	5.0886E-11	**0.0000E+00**
	worst	1.1490E+01	**1.0942E+01**	1.1757E+01	1.3111E+01
	mean	3.8056E+00	1.2120E+00	1.0978E+00	**8.3291E-01**
	std	5.2091E+00	3.3762E+00	3.1751E+00	2.7172E+00
1.5×10^5	best	1.1674E-11	0.0000E+00	7.2093E-15	**0.0000E+00**
	median	6.0846E-10	0.0000E+00	1.2362E-11	**0.0000E+00**
	worst	1.1374E+01	1.0942E+01	1.1757E+01	**1.8688 E+00**
	mean	2.0948E+00	1.2120E+00	8.7697E-01	**7.4754 E-02**
	std	4.3063E+00	3.3762E+00	3.0439E+00	3.7377E-01

8. Conclusion

In this paper, learning automata have been applied for adaptive parameter adjustment and strategy selection in DE. Two classes of possible approach have been used for choosing new values for crossover rate as well as mutation strategy. In the first class, i.e. GLADE, all individuals share the same value for *CR*, and use the same mutation strategy. In the second class, i.e. ILADE, each individual acts in a separate manner and chooses its own *CR* and mutation strategy based on its success and failure.

In order to justify the proposed approach, experiments were carried out to compare the performance of the proposed method with several state-of-the-art algorithms (i.e., jDE,[21] EPSDE,[33] OXDE,[20] CMA-ES,[57] CLPSO[6] and GL-25[56]) on ten benchmark functions of CEC2005 special session of real-parameter optimization. The proposed approach was also used to solve the parameter estimation for FMSW.[55]

From the experimental results, following conclusions can be drawn on the real-parameter function optimization problems. The proposed LADE greatly improves the performance of DE in terms of both accuracy and convergence speed. In comparison with other peer algorithms, LADE shows a good performance, and it is computationally efficient. Finally, it is also worth noticing that the results of the paper suggest that adjustment of the parameters and mutation strategy at the individual level can lead to a better performance.

Future research may focus on providing a general framework for improving the performance of EAs using the concept of LA. It is also interesting to study other types of learning for the LA.

References

1. R. Horst, P. M. Pardalos and H. P. Benson, Handbook of global optimization, *Journal of Global Optimization* **9**(2) (1996) 217–218.
2. P. M. Pardalos and M. G. Resende, *Handbook of Applied Optimization* (Oxford University Press, 2001).
3. I. G. Tsoulos, Solving constrained optimization problems using a novel genetic algorithm, *Applied Mathematics and Computation* **208**(1) (2009) 273–283.
4. K. Socha and M. Dorigo, Ant colony optimization for continuous domains, *European Journal of Operational Research* **185**(3) (2008) 1155–1173.
5. Q. Xu, S. Wang, L. Zhang and Y. Liang, A novel chaos danger model immune algorithm, *Communications in Nonlinear Science and Numerical Simulation* **18**(11) (2013) 3046–3060.
6. J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, *IEEE Transactions on Evolutionary Computation* **10**(3) (2006) 281–295.
7. M. Hasanzadeh, S. Sadeghi, A. Rezvanian and M. R. Meybodi, Success rate group search optimiser, *Journal of Experimental & Theoretical Artificial Intelligence* (2015) 1–17, doi: 10.1080/0952813X.2014.971467, in press.
8. N. Siddique and H. Adeli, Water drop algorithms, *International Journal on Artificial Intelligence Tools* **23**(6) (2014) 1430002-1–22.
9. M. Ali, P. Siarry and M. Pant, An efficient differential evolution based algorithm for solving multi-objective optimization problems, *European Journal of Operational Research* **217**(2) (2012) 404–416.
10. J. K. Kordestani, A. Ahmadi and M. R. Meybodi, An improved differential evolution algorithm using learning automata and population topologies, *Applied Intelligence* **41**(4) (2014) 1150–1169.
11. S. Nabizadeh, A. Rezvanian and M. R. Meybodi, A multi-swarm cellular PSO based on clonal selection algorithm in dynamic environments, in *2012 Int. Conf. on Informatics, Electronics & Vision* (2012), pp. 482–486.
12. A. Rezvanian and M. R. Meybodi, An adaptive mutation operator for artificial immune network using learning automata in dynamic environments, in *Proc. of the 2010 Second World Congress on Nature and Biologically Inspired Computing* (2010), pp. 479–483.
13. A. Rezvanian and M. R. Meybodi, LACAIS: Learning automata based cooperative artificial immune system for function optimization, in *Contemporary Computing* (2010), pp. 64–75.

14. R. Rastegar, M. R. Meybodi and A. Hariri, A new fine-grained evolutionary algorithm based on cellular learning automata, *International Journal of Hybrid Intelligent Systems* **3**(2) (2006) 83–98.
15. F. Abtahi, M. R. Meybodi, M. M. Ebadzadeh and R. Maani, Learning automata-based co-evolutionary genetic algorithms for function optimization, in *Proc. of the 6th Int. Symp. on Intelligent Systems and Informatics* (2008), pp. 1–5.
16. A. Hashemi and M. Meybodi, A note on the learning automata based algorithms for adaptive parameter selection in PSO, *Applied Soft Computing* **11**(1) (2011) 689–705.
17. R. Vafashoar, M. R. Meybodi and A. H. Momeni Azandaryani, CLA-DE: A hybrid model based on cellular learning automata for numerical optimization, *Applied Intelligence* **36**(3) (2012) 735–748.
18. S. M. Farahani, A. A. Abshouri, B. Nasiri and M. R. Meybodi, Some hybrid models to improve firefly algorithm performance, *International Journal of Artificial Intelligence* **8**(S12) (2012) 97–117.
19. S. Das and P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, *IEEE Transactions on Evolutionary Computation* **15**(1) (2011) 4–31.
20. Y. Wang, Z. Cai and Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, *Information Sciences* **185**(1) (2012) 153–177.
21. 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 Transactions on Evolutionary Computation* **10**(6) (2006) 646–657.
22. A. K. Qin, V. L. Huang and P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, *IEEE Transactions on Evolutionary Computation* **13**(2) (2009) 398–417.
23. M. Ali, M. Pant, A. Abraham and C. W. Ahn, Swarm directions embedded differential evolution for faster convergence of global optimization problems, *International Journal on Artificial Intelligence Tools* **21**(3) (2012) 1240013.
24. R. Storn and K. Price, Differential evolution — A simple and efficient adaptive scheme for global optimization over continuous spaces, International Computer Science Institute, Berkeley (CA, 1995, Tech. Rep. TR-95–012, 1995).
25. R. Storn and K. Price, Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* **11**(4) (1997) 341–359.
26. D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms, *Applied Soft Computing* **9**(3) (2009) 1126–1138.
27. M. Ali, M. Pant and A. Abraham, Unconventional initialization methods for differential evolution, *Applied Mathematics and Computation* **219**(9) (2013) 4474–4494.
28. S. Rahnamayan, H. R. Tizhoosh and M. M. Salama, Opposition-based differential evolution, *IEEE Transactions on Evolutionary Computation* **12**(1) (2008) 64–79.
29. S. Das, A. Konar and U. K. Chakraborty, Two improved differential evolution schemes for faster global search, in *Proc. of the 2005 Conference on Genetic and Evolutionary Computation* (2005), pp. 991–998.
30. J. Liu and J. Lampinen, A fuzzy adaptive differential evolution algorithm, *Soft Computing* **9**(6) (2005) 448–462.
31. J. Zhang and A. C. Sanderson, JADE: Adaptive differential evolution with optional external archive, *IEEE Transactions on Evolutionary Computation* **13**(5) (2009) 945–958.
32. W. Gong, Á. Fialho and Z. Cai, Adaptive strategy selection in differential evolution, in *Proc. of the 12th Annual Conference on Genetic and Evolutionary Computation* (2010), pp. 409–416.

33. R. Mallipeddi, P. N. Suganthan, Q.-K. Pan and M. F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, *Applied Soft Computing* **11**(2) (2011) 1679–1696.
34. Y. Wang, Z. Cai and Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, *IEEE Transactions on Evolutionary Computation* **15**(1) (2011) 55–66.
35. J. Sun, Q. Zhang and E. P. Tsang, DE/EDA: A new evolutionary algorithm for global optimization, *Information Sciences* **169**(3) (2005) 249–262.
36. J. K. Kordestani, A. Rezvanian and M. R. Meybodi, CDEPSO: A bi-population hybrid approach for dynamic optimization problems, *Applied Intelligence* **40**(4) (2014) 682–694.
37. U. Halder, S. Das and D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, *IEEE Transactions on Cybernetics* **43**(3) (2013) 881–897.
38. S. Das, A. Konar and U. K. Chakraborty, Annealed differential evolution, in *IEEE Congress on Evolutionary Computation* (2007), pp. 1926–1933.
39. S. Das, A. Abraham, U. K. Chakraborty and A. Konar, Differential evolution using a neighborhood-based mutation operator, *IEEE Transactions on Evolutionary Computation* **13**(3) (2009) 526–553.
40. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction* (Printice-Hall, 1989).
41. M. A. L. Thathachar and P. S. Sastry, Varieties of learning automata: An overview, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **32**(6) (2002) 711–722.
42. E. Cuevas, D. Zaldivar and M. Pérez-Cisneros, Seeking multi-thresholds for image segmentation with learning automata, *Machine Vision and Applications* **22**(5) (2011) 805–818.
43. M. Hasanzadeh, S. Sadeghi, A. Rezvanian and M. R. Meybodi, Cellular edge detection: Combining cellular automata and cellular learning automata, *AEU — International Journal of Electronics and Communications* **69**(9) (2015) 1282–1290.
44. M. Ahangaran and H. Beigy, Cellular learning automata with external input and its applications in pattern recognition, in *Fifth Int. Conf. on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control* (2009), pp. 1–4.
45. M. Esnaashari and M. R. Meybodi, A cellular learning automata-based deployment strategy for mobile wireless sensor networks, *Journal of Parallel and Distributed Computing* **71**(7) (2011) 988–1001.
46. Q. H. Wu and H. L. Liao, Function optimisation by learning automata, *Information Sciences* **220** (2013) 379–398.
47. H. L. Liao and Q. H. Wu, Multi-objective optimization by learning automata, *Journal of Global Optimization* **55**(2) (2013) 459–487.
48. A. Rezvanian, M. R. Meybodi and T. Kim, Tracking extrema in dynamic environments using a learning automata-based immune algorithm, in *Grid and Distributed Computing, Control and Automation* (2010), pp. 216–225.
49. A. Rezvanian, M. Rahmati and M. R. Meybodi, Sampling from complex networks using distributed learning automata, *Physica A: Statistical Mechanics and its Applications* **396** (2014) 224–234.
50. A. Rezvanian and M. R. Meybodi, Finding maximum clique in stochastic graphs using distributed learning automata, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **23**(1) (2015) 1–31.
51. A. Mousavian, A. Rezvanian and M. R. Meybodi, Cellular learning automata based algorithm for solving minimum vertex cover problem, in *2014 22nd Iranian Conference on Electrical Engineering* (2014), pp. 996–1000.

52. J. Akbari Torkestani, An adaptive learning automata-based ranking function discovery algorithm, *Journal of Intelligent Information Systems* **39**(2) (2012) 441–459.
53. D. H. Wolpert and W. G. Macready, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation* **1**(1) (1997) 67–82.
54. P. N. Suganthan *et al.*, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, KanGAL Report 2005005, (2005).
55. S. Das and P. N. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems, Jadavpur University, Nanyang Technological University, Kolkata (2010).
56. C. García-Martínez, M. Lozano, F. Herrera, D. Molina and A. M. Sánchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, *European Journal of Operational Research* **185**(3) (2008) 1088–1113.
57. N. Hansen and A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, *Evolutionary Computation* **9**(2) (2001) 159–195.
58. F. Wilcoxon, Individual comparisons by ranking methods, *Biometrics Bulletin* **1**(6) (1945) 80–83.
59. S. García, D. Molina, M. Lozano and F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization, *Journal of Heuristics* **15**(6) (2009) 617–644.
60. L. E. Toothaker, *Multiple comparison procedures* (Sage, 1993).
61. H. K. Singh and T. Ray, Performance of a hybrid EA-DE-memetic algorithm on CEC 2011 real world optimization problems, in *2011 IEEE Congress on Evolutionary Computation* (2011), pp. 1322–1326.
62. F. Qian, B. Xu, R. Qi and H. Tianfield, Self-adaptive differential evolution algorithm with α-constrained-domination principle for constrained multi-objective optimization, *Soft Computing* **16**(8) (2012) 1353–1372.
63. G. Reynoso-Meza, J. Sanchis, X. B. Ferragud and J. M. H. Durá, Hybrid DE algorithm with adaptive crossover operator for solving real-world numerical optimization problems, in *IEEE Congress on Evolutionary Computation* (2011), pp. 1551–1556.