An Improved Differential Evolution Algorithm Using Learning Automata and Population Topologies

Javidan Kazemi Kordestani [a,*], Ali Ahmadi [b], Mohammad Reza Meybodi [c]

[a] *Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran*

[b] *Electrical & Computer College, K. N. Toosi University of Technology, Shariati St., Seyedkhandan, Tehran, Iran*

[c] *Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran*

Javidan.kazemi@gmail.com[*], ahmadi@eetd.kntu.ac.ir, mmeybodi@aut.ac.ir

Abstract

A new variant of Differential Evolution (DE), called ADE-Grid, is presented in this paper which adapts the mutation strategy, crossover rate (*CR*) and scale factor (*F*) during the run. In ADE-Grid, learning automata (LA), which are powerful decision making machines, are used to determine the proper value of the parameters *CR* and *F*, and the suitable strategy for the construction of a mutant vector for each individual, adaptively. The proposed automata based DE is able to maintain the diversity among the individuals and encourage them to move toward several promising areas of the search space as well as the best found position. Numerical experiments are conducted on a set of twenty four well-known benchmark functions and one real-world engineering problem. The performance comparison between ADE-Grid and other state-of-the-art DE variants indicates that ADE-Grid is a viable approach for optimization. The results also show that the proposed ADE-Grid improves the performance of DE in terms of both convergence speed and quality of final solution.

Keywords: *Global optimization, Differential evolution, Learning automata, Parameter adjustment, Strategy adaptation*

1. Introduction

The rapid development of the technology gives rise to a set of complex real-world problems where several objectives may need to be satisfied. As the complexity of the real-world optimization problems constantly increases, it is always necessary to provide new optimization techniques and to improve the existing methods. Analytical methods encounter many difficulties when applying to complex optimization problems [1]; thus it is not possible to apply them for many practical cases. Evolutionary Algorithms (EAs), however, have been proven to work better than analytical methods [1]. Among existing EAs, Differential Evolution (DE) is one of the most powerful stochastic real-parameter optimization algorithms, which was originally proposed by Storn and Price [2, 3]. Over the past decade, DE has attracted a great deal of attention by the researchers because of its simplicity, effectiveness and robustness.

DE and its variants have been successfully applied to a wide range of problems including numerical optimization [4–6], dynamic optimization problems [7–9], multi-objective optimization [10–12], etc.

Despite its successful applications, the performance of DE is severely dependant on the choice of its parameters, i.e. the crossover rate *CR*, scale factor *F*, and population size *NP* [13]. Moreover, the selection of the proper trial vector generation strategy is very critical to the performance of DE [14]. In this paper, we propose an automata-based approach for automatic tuning of the crossover rate, scale factor and strategy selection in DE. In the proposed approach, each genome of the population is placed on a distinct cell of a typical cubic grid structure. Three mutation strategies are integrated in the trial vector generation stage of DE with the aim to control the degree of diversity and greediness of the genomes. The motivation behind this is to encourage the genomes of the population to move toward several promising areas of the search space using their neighborhood cells. Moreover, each genome of the population chooses its crossover rate, scale factor and mutation strategy based on its own progress. The rest of this paper is structured as follows: Section 2 gives an introduction to the basic DE algorithm. In Section 3, we review some of the existing works in brief. A brief description about LA is given in Section 4. Our proposed algorithms are presented in Section 5. Extensive experiments have been carried out to evaluate the performance of our proposed ADE-Grid in Section 6. Finally, Section 7 concludes the paper with some future works.

2. Differential evolution

The main idea of DE is to use spatial difference among the population of vectors to guide the search process toward the optimum solution. In short, almost all DE variants work according to the following steps:

i. *Initialization*: A number of *NP* points are randomly sampled from the search space to form the initial population.
ii. Repeat steps (iii), (iv) and (v) for each vector $\vec{x}_i$ ($i \epsilon \{1, 2, \dots, NP\}$) of the current population.
iii. *Mutation*: A mutant vector $\vec{v}_i$ is generated for $\vec{x}_i$ according to a specified mutation strategy.
iv. *Repair*: if the mutant vector $\vec{v}_i$ is out of the feasible region, a repair operator is utilized to make it feasible.
v. *Crossover*: A crossover operator is applied to combine the information from $\vec{x}_i$ and $\vec{v}_i$ and form a trial vector $\vec{u}_i$.
vi. *Selection*: The most fitted vector among $\vec{x}_i$ and $\vec{u}_i$ is transferred to the next generation.
vii. If the termination condition is not met go to step (ii).

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) [15].
DE has several advantages that make it a powerful tool for optimization tasks. Specifically, (1) DE has a simple structure and is easy to implement; (2) despite its simplicity, DE exhibits a high performance; (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 [16]. 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. [17, 18] used opposition-based learning for generating the initial population in DE. Ali et al. [16] 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 contains 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 [3] 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. [19] 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. [20] 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. In another work, Brest and Sepesy Maučec [21] improved the efficiency and robustness of the jDE using a population size reduction scheme.

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. [19]. 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:

$$\mathcal{F}_{iter} = (\mathcal{F}_{max} - \mathcal{F}_{min}) \times \frac{(iter_{max} - iter)}{iter_{max}} \quad (1)$$

where $\mathcal{F}_{iter}$ 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, and $iter_{max}$ 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 [22] 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*, *MIDDLE* 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. [14] proposed a self-adaptive DE, SaDE, in which the control parameters and trial vector generation strategies are adaptively adjusted based on their previous performance in generating promising solutions. Zhang and Sanderson [23] 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 utilized to set the new *F* and *CR*.

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. [24] 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. [4] introduced an ensemble of mutation strategies and control parameters of DE (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 utilized to increase the probability of generating promising offspring. Wang et al. [5] 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.

3.4. Hybridizing DE with other operators

A group of studies have combined DE with other optimization methods. For example, Sun et al. [25] 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. Another example is DEPSO proposed by Zang and Xie [26]. DEPSO is a hybrid algorithm of differential evolution and particle swarm optimization (PSO) [27], in which PSO and DE alternate at the odd and at the even iterations. The results indicate that DEPSO has a better convergence than both the DE and PSO over the tested constrained optimization problems. Kazemi Kordestani, Rezvanian, and

Meybodi [9] proposed a bi-population hybrid collaborative model of crowding-based differential evolution (CDE) and particle swarm optimization, 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. [28] presented a cluster-based differential evolution with external archive (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. For example, Storn and Price suggested five different mutation strategies which are "DE/rand/1", "DE/best/1", "DE/current-to-best/1", "DE/best/2", and "DE/rand/2" [29, 30]. Zhang and Sanderson [23] 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. [31] embedded quantization orthogonal crossover (QOX) with DE/rand/1/bin to enhance the search ability of DE. Das et al. [32] 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. This idea has achieved good results in PSO and various topological variants of the classical PSO have been introduced and studied in the literature. For example, Kennedy [33] has examined the effect of four alternative topologies, i.e. ring, wheel, star and random edges, on the behavior of PSO. His findings show that less-interconnected topologies, because of their lower spread of information, can produce better results on multimodal functions. On the other hand, star topology can achieve better solutions on unimodal test functions. In another work, Kennedy and Mendes [34] inferred that von Neumann topology is the best, on average, population topology. For further study on the impact of population topology in the field of PSO, interested readers are referred to [35, 36]. Inspired by the idea of neighborhood topology in PSO, Omran et al. [37] have investigated the effect of ring and von Neumann neighborhood topologies on the behavior of DE. Their results show that using ring topology with DE (DE/lbest/1) can generally improve its performance. Das et al. [38] proposed DE with global and local neighborhoods (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.

Besides the above mentioned methods, a number of methods exist in the literature that have used other type of techniques for improving the performance of DE. For example, Han et al. [39] proposed a dynamic group-based differential evolution (GDE), in which the individuals are mutated based on their fitness value. In GDE, all individuals are grouped into two categories: superior group, i.e. individuals with higher fitness, and inferior group, i.e. individuals with lower fitness. Each group is evolved with a different mutation operation. Superior group performs "DE/best/1" to exploit the neighborhood of the best solution. Inferior group performs "DE/rand/1" to explore for the potential solutions.

In this paper, we have combined the desirable attributes of *parameter adjustment*, *strategy adaptation*, and *population topology* into a single algorithm. The aim of our proposal is to accomplish a two-fold task to improve the performance of DE: (*a*) to conduct the search process of individuals toward different areas of the search space and learn the successful trajectories by LA, (*b*) to establish a balance between the exploration and exploitation of DE by adjusting its parameters F and CR during the run. It should be noted that the difference between our work and the previously mentioned DEs with population topologies is that their main objective is to reduce the chance of getting trapped in local optima by establishing a social network among the individuals. Thus, the population topology is the major component of those methods. In contrast, ADE-Grid aims to control the trajectory of individuals toward promising areas of the search space. For this purpose, neighborhood topology is incorporated as a mutation strategy in trial vector generation process, where LA is applied for adaptive selection of the mutation strategy for each individual, according to its success. Moreover, the parameters F and CR are also adjusted for each individual based on its situation in the search space.

4. Learning automata

Learning Automata are stochastic learning tools that can learn the optimal policy through iterative interaction with an unknown random environment [40, 41]. The learning process of the automaton can be divided into three steps. First, the automaton selects an action from a set of finite actions and applies it to the environment. Second, the environment evaluates the effectiveness of the selected action and generates a response for the learning automaton. Finally, the automaton uses the feedback of the environment to update its internal action probabilities. By repeating this three-step procedure, the learning automaton will be gradually guided toward selecting the optimal action. In this work, variable-structure leaning automata (VSLA) are used to improve the performance of DE by adjusting its parameters (i.e. F and CR) as well as selecting its trial vector generation strategy.

VSLA can be represented with a quadruple $\{\alpha, \beta, p, T\}$, where $\alpha = \{\alpha_1,\ldots, \alpha_r\}$ is the set of actions, $\beta = \{\beta_1,\ldots, \beta_m\}$ is the set of inputs, and $p = \{p_1,\ldots, p_r\}$ is the probability vector which determines the selection probability of each action, and T is the learning algorithm which is used to govern the reinforcement scheme, i.e. $p(n+1) = T\,[\alpha(n), \beta(n), p(n)]$. The framework for the learning process of a VSLA in a stationary environment is shown in Algorithm 1.

Algorithm 1. The framework for the learning process of learning automata in a random unknown environment

```
1. Let $\alpha = \{\alpha_1,\ldots, \alpha_r\}$ be the action set, where $r$ is the number of actions.
2. Let $p = \{p_1,\ldots, p_r\}$ be the action probability vector.
3. while (the automaton converges to one of its actions)
4.     The learning automaton chooses one of its actions based on the probability distribution $p$.
5.     The environment evaluates the action and calculates a reinforcement signal $\beta \in \{0,1\}$.
6.     The environment feedbacks $\beta$ to the learning automaton.
7.     The automaton updates its probability vector.
8. end-while
```

Learning automata have been successfully applied to a vast variety of science and engineering applications. In the following, we will review some of the existing applications of LA. One of the applications of LA is for parameter adaption. For example, Rezvanian and Meybodi [42] applied learning automata for tuning the mutation rate of antibodies in artificial immune system to establish a balance between the global and local search ability of the algorithm. Hashemi and Meybodi [43] introduced two classes of learning automata based algorithms, one at swarm degree and another at particle degree, for adaptive selection of the values for inertia weight (w) and acceleration coefficients (c_1 and c_2) in particle swarm optimization. Another well-known application of LA is in wireless sensor networks (WSN). Moghis, Meybodi, and Esnaashari [44] applied learning automata for channel assignment in WSN. Esnaashari and Meybodi [45] proposed a cellular learning automata-based deployment strategy for attaining high coverage in mobile WSN. Apart from mentioned applications, LA has been successfully applied to vertex coloring problem [46], traffic signal control [47], information retrieval [48], multicast routing problem [49], etc.

5. Adaptive Differential Evolution with Grid structure (ADE-Grid)

This section provides a complete description of the proposed method for solving the following continuous global optimization problem:

$$\begin{aligned}&\text{Minimize: } f(\vec{x}), \vec{x} = (x_1, x_2, \dots, x_D) \in s \\ &\qquad \text{Subject to: } x_i \in [lb_i, ub_i]\end{aligned} \tag{2}$$

where $f(\vec{x})$ is a continuous real-valued objective function to be minimized, and s is the solution space. Finally, lb_i and ub_i are the box constraints corresponding to ith dimension of the search space, that is, $\forall\ i \in \{1, \dots, D\}, -\infty < l_i < u_i < \infty$.

5.1. Initialization

The proposed ADE-Grid 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 $\vec{X}_i = (x_{i1}, x_{i2}, \dots, x_{id})$. The initial population of vectors is simply randomized into the boundary of the search space according to a uniform distribution as follows:

$$\vec{X}_i = lb_j + rand_j[0,1] \times (ub_j - lb_j) \tag{3}$$

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

After generating the population of genomes in the search space, an $n\times n$, 10×10 in this study, grid with checkerboard-like lattice structure is created. Each genome of the population is then assigned to a distinct cell of the grid (Fig. 1). The assignment of the genomes to the cells of the grid remains unchanged during the course of optimization.

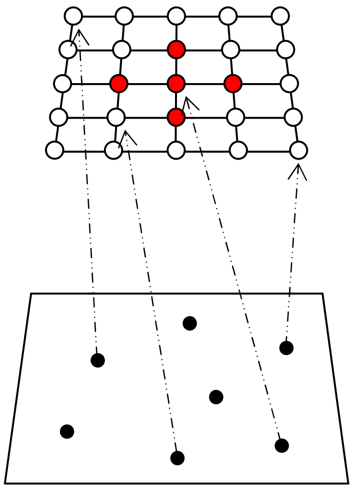


Fig. 1. A typical example for assignment of the genomes in the population to the cells of the grid.

5.2. Choosing 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 $\vec{v}_i = (v_{i1}, v_{i2}, \dots, v_{id})$ corresponding to the target vector $\vec{X}_i$. Several strategies have been proposed for generating donor vector $\vec{v}_i$. The effect of different mutation strategies on the performance of DE has been studied in [50]. In this paper, we aim at conducting the search process of the DE toward the most promising areas in the search space by rewarding the successful trajectories of the individuals using the concept of learning automata. Thus, in order to choose a proper mutation strategy among the set of strategies, a learning automaton was placed upon each genome of the population, i.e. $|LA_{mutation}|=|popsize|=100$. $LA_{mutation}$ has three actions, each of which corresponds to a distinct trial vector generation strategy. The initial selection probability of each mutation strategy is (1/3), where 3 is the number of actions. In this paper, we use the following mutation strategies as the actions of the $LA_{mutation}$ to generate donor vector:

$DE/rand\ cell/1$:

$$\vec{v}_{\mathrm{i}} = \vec{x}_1 + F.(\vec{x}_2 - \vec{x}_3) \tag{4}$$

$DE/rand - to - best\ neghborhood\ cell/1$:

$$\vec{v}_{\mathrm{i}} = \vec{x}_1 + F.\left(\vec{x}_{\mathrm{best\_neghborhood}} - \vec{x}_1\right) + F.(\vec{x}_2 - \vec{x}_3) \tag{5}$$

$DE/rand - to - best\ cell/2$:

$$\vec{v}_{\mathrm{i}} = \vec{x}_1 + F.\left(\vec{x}_{\mathrm{best\_cell}} - \vec{x}_1\right) + F.(\vec{x}_2 - \vec{x}_3) + F.(\vec{x}_4 - \vec{x}_5) \tag{6}$$

where $\vec{v}_i$ is the donor vector corresponding to ith 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. F is the scaling factor used to control the amplification of difference vector.

In this work, various neighborhood structures are employed to determine the best cell among the neighbors. Different neighborhood structures for the central cell of a 3×3 grid are illustrated in Fig. 2.

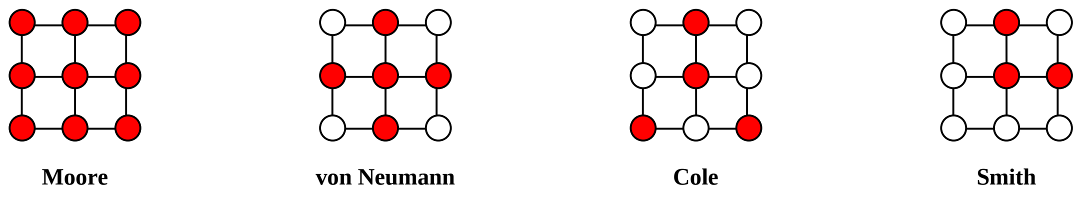


Fig. 2. Different neighborhood topologies for the central cell in a typical grid structure.

It is clear that parameter F plays a key role on the behavior of DE. A large value of F can spread out the mutant vector over a wide area in the search space, and thus increases the diversity of the population. In contrast, a small value of F focuses the search process around the neighborhood of the current solution, and thus it can accelerate the convergence. With these in mind, in this work, a learning automaton is associated with each genome of the population to choose a value for parameter F, $|LA_F|=|popsize|=100$. The actions of LA_F correspond to F=0.4 and F=0.6, and the initial selection probability of both of them is set to 0.5.

5. 3. Repair operator

If the generated mutant vector is out of the search boundary, a repair operator is used to make $\vec{v}_i$ 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:

$$v_{ij} = \begin{cases} \frac{x_{ij} + lb_j}{2} & if\ v_{ij} < lb_j \\ \frac{x_{ij} + ub_j}{2} & if\ v_{ij} > ub_j \end{cases} \tag{7}$$

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

5. 4. 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 article, we use *binomial crossover* which is defined as follows:

$$u_{ij} = \begin{cases} v_{ij} & if\ rand_{ij}[0,1] \le CR\ or\ j = jrand \\ x_{ij} & otherwise \end{cases} \tag{8}$$

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.

Considering Eq. (8), the parameter *CR* has a great influence on the diversity of the offspring population. A large value of *CR* can maintain the diversity among the individuals, which is favorable for multi-modal functions. A small value of *CR*, in turn, can make trial vector little different from the target vector. This feature is desirable for optimizing separable problems. In this work, each genome of the population is equipped with a learning automaton for adjusting the crossover rate *CR*, i.e. $|LA_{CR}|=|popsize|=100$. LA_{CR} is a 2-action learning automaton, which its actions correspond to *CR*=0.9 and *CR*=0.1. At the beginning of the optimization process, the selection probability of both actions is set to 0.5.

5. 5. Selection

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.

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G} & if\ f(\vec{X}_{i,G}) \le f(\vec{U}_{i,G}) \\ \vec{X}_{i,G} & otherwise \end{cases} \tag{9}$$

5. 6. Updating the probability vector of each automaton

After the selection stage was done, the automata update their probability vectors based on the reinforcement signal they receive from the environment. Based on the received reinforcement signals, the automata can determine whether their action were right or wrong, and update their probability vector accordingly. In this work, the reinforcement signal is generated as follows:

$$Reinforcement\ signal = \begin{cases} 0 & if\ f(\vec{X}_{i,G}) < (\vec{X}_{i,G+1}) \\ 1 & otherwise \end{cases} \tag{10}$$

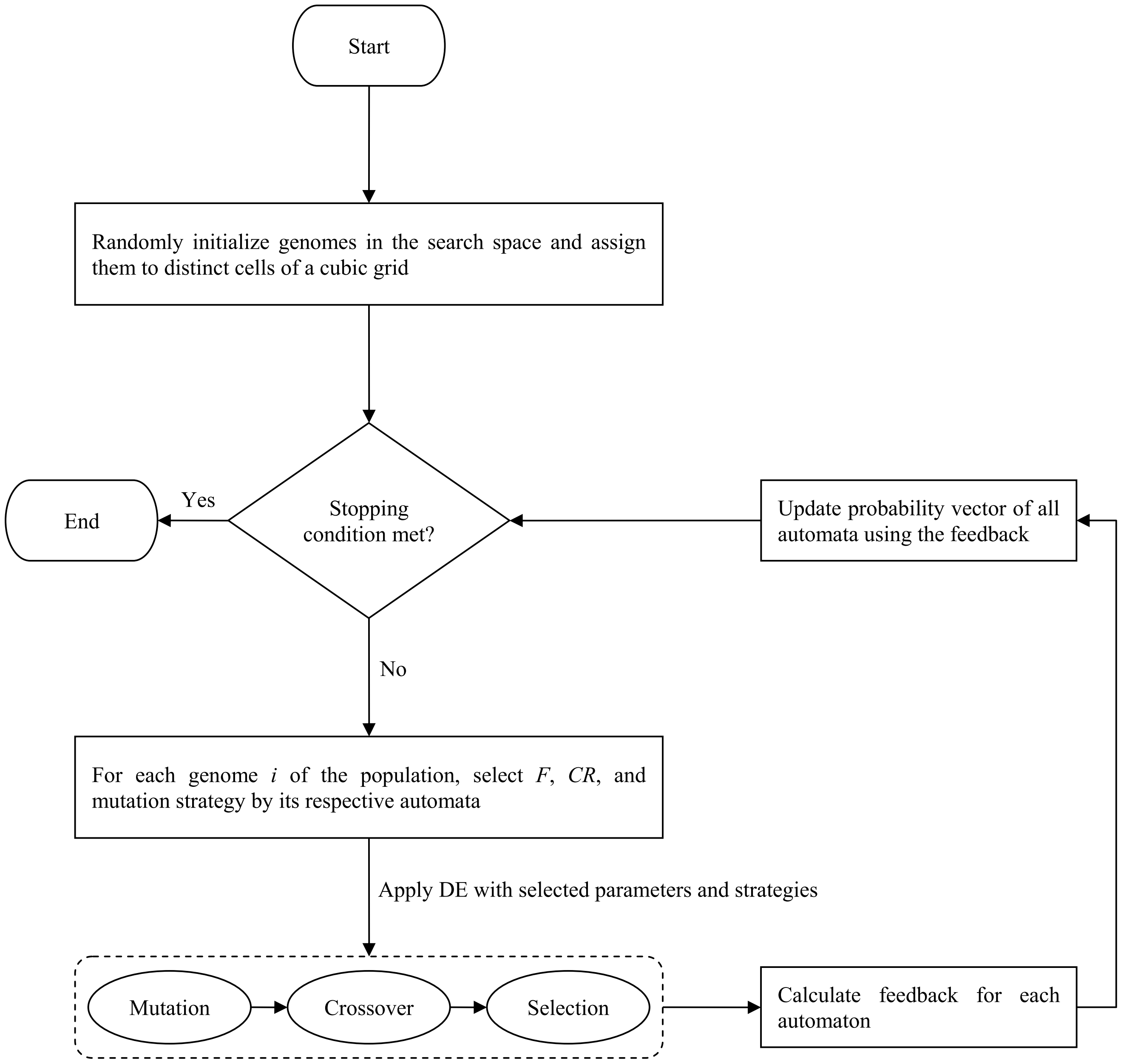
Then, the corresponding probability vectors of the $LA_{mutation}$, LA_{CR} and LA_F for each genome are modified based on the learning algorithm of automata using Algorithm 2.

Algorithm 2. *updateProbability*(automata *LA* , selected action *i* , reinforcement signal *β*)

```
1. for each action $j \epsilon [1,2,\dots,r]$ of $LA$ do
2.     if ($\beta$=0) //favorable response
3.         $p_j(n+1) = \begin{cases} p_j(n) + a.(1-p_j(n)) & if\ i=j \\ p_j(n).(1-a) & if\ i \ne j \end{cases}$    (11)
4.     else if ($\beta$=1) //unfavorable response
5.         $p_j(n+1) = \begin{cases} p_j(n).(1-b) & if\ i=j \\ \frac{b}{r-1} + (1-b).p_j(n) & if\ i \ne j \end{cases}$    (12)
6.     end-if
7. end-for
```

In Algorithm 2, *a* (alpha) and *b* (beta) are *learning parameters* associated with the reward and penalty that can take values in the range of [0.0, 1.0]. Three linear reinforcement schemes can be obtained by selecting different values for the reward and penalty parameters, namely $L_{R\text{-}P}$, $L_{R\text{-}I}$ and $L_{R\varepsilon P}$. In $L_{R\text{-}P}$ (Linear Reward-Penalty), the value of the parameters *a* and *b* are equal (*a*=*b*). In $L_{R\text{-}I}$ (Linear Reward-Inaction) the learning parameter *b* is zero. Finally, in $L_{R\varepsilon P}$ (Linear Reward-epsilon-Penalty) the learning parameter b is much smaller than a (a>>b). In this study, we consider an $L_{R\varepsilon P}$ reinforcement scheme with a=0.1 and b=0.05 for all types of automata.

In short, the general framework of ADE-Grid utilizes three learning automata for each genome of the population, i.e. LA_{mut}, LA_{CR} and LA_F. Each automaton tries to adjust its associated parameter (or strategy) by monitoring the search progress of its corresponding genome. The working procedure of the proposed ADE-Grid is shown in Fig. 3. The pseudo-code for ADE-Grid is also presented in Algorithm 3.



Start
Randomly initialize genomes in the search space and assign them to distinct cells of a cubic grid
Stopping condition met?
Yes
End
No
Update probability vector of all automata using the feedback
For each genome i of the population, select F, CR, and mutation strategy by its respective automata
Apply DE with selected parameters and strategies
Mutation
Crossover
Selection
Calculate feedback for each automaton

Fig. 3. A schematic of the proposed ADE-Grid.

Algorithm 3. ADE-Grid Algorithm

```
1.  define
2.  Initialize the algorithm and problem parameters: grid size $n$, population $P$, population size $NP = n \times n$, neighborhood structure $K$, fitness evaluations counter $fe = 0$, maximum fitness evaluations $FE_{max}$, current generation $G = 0$.
3.  Initialize an $n \times n$ cubic structure.
4.  Initialize the set of $LA_{mutation} = \{LA_{mut}(1), \ldots, LA_{mut}(NP)\}$ with parameters: action set $a = \{$"DE/rand cell/1", "DE/rand-to-best neighbor cell/1", "DE/rand-to-best cell/2"$\}$, action probability vector $p = \{(1/3), (1/3), (1/3)\}$, alpha = 0.1, beta = 0.05.
5.  Initialize the set of $LA_{Crossover\ rate} = \{LA_{CR}(1), \ldots, LA_{CR}(NP)\}$ with parameters: action set $a = \{CR = 0.9, CR = 0.1\}$, action probability vector $p = \{(1/2), (1/2)\}$, alpha = 0.1, beta = 0.05.
6.  Initialize the set of $LA_{scale\ factor} = \{LA_F(1), \ldots, LA_F(NP)\}$ with parameters: action set $a = \{F = 0.4, F = 0.6\}$, action probability vector $p = \{(1/2), (1/2)\}$, alpha = 0.1, beta = 0.05.
7.  Let $\alpha_j(M) \mid j \in \{1,2,3\}, M \in [1, \ldots, NP]$ be the selected action of $LA_{mut}(M)$.
8.  Let $\alpha_k(M) \mid k \in \{1,2\}, M \in [1, \ldots, NP]$ be the selected action of $LA_{CR}(M)$.
9.  Let $\alpha_l(M) \mid l \in \{1,2\}, M \in [1, \ldots, NP]$ be the selected action of $LA_F(M)$.
10. Generate an initial population $P_0 = \{\vec{X}_{1,0}, \ldots, \vec{X}_{NP,0}\}$ in the $D$-dimensional search space according to Eq. (3).
11. Assign each genome of the population to a distinct cell of the grid.
12. Let $f$ be the fitness value.
13. for each genome $i \in [1, \ldots, popsize]$
14.     Place $LA_{mut}$(i), $LA_{CR}$(i), and $LA_F$(i) on $i$th individual.
15. end-for
16. Evaluate the objective function values $f(\vec{X}_1), \ldots, f(\vec{X}_{NP})$.
17. $fe := NP$
18. while ($fe < FE_{max}$)
19.     $P_{G+1} = \emptyset$ // to determine the population of the next generation
20.     for each genome $i \in [1, \ldots, popsize]$ in current population ($P_G$)
21.         $LA_{mut}$(i), $LA_{CR}$(i), and $LA_F$(i) select an action from their action set based on their probability vector $p$, according to a random uniform distribution.
22.         Generate the donor vector $\vec{V}_i$ using selected mutation strategy and scale factor $F$. // mutation step
23.         Repair $\vec{V}_i$ according to Eq. (7). // repair operator
24.         Generate trial vector $\vec{U}_i$ using selected crossover rate $CR$ and Eq. (8). // crossover step
25.         Evaluate the objective function value $f(\vec{U}_i)$.
26.         Add an individual to $P_{G+1}$ using Eq. (9). // selection and replacement step
27.         Calculate the reinforcement signal $\beta$ according to Eq. (10).
28.         updateProbability($LA_{mut}$(i), $\alpha_j(i)$, $\beta$)
29.         updateProbability($LA_{CR}$(i), $\alpha_k(i)$, $\beta$)
30.         updateProbability($LA_F$(i), $\alpha_l(i)$, $\beta$)
31.     end-for
32.     $G := G + 1$
33.     $fe := fe + NP$
34. end-while
```

6. Experimental study

6.1. Experimental setup

6.1.1. Benchmark functions

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 [51]. 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 ADE-Grid, a set of experiments was carried out using 14 classical numerical test instances and 10 benchmark functions from CEC2005 special session on real-parameter optimization [52]. This group of 24 functions is a combination of unimodal, multimodal, separable, non-separable, shifted, rotated and noisy functions. For all test problems, except for f_3, f_{11}, f_{12}, and f_{13}, the dimension of the search space is 30. All of the problems have symmetric search space boundaries except for f_{13} and f_{21}. f_{13} uses an asymmetric search space boundary to investigate the bias of the algorithms toward the center of the search space. Also in f_{21}, the search space is not bounded. This group of functions also let us to study the effect of shift, rotation, or noise on the performance of different algorithms. A detailed description of these test instances can be found in Table 1. Moreover, the performance of the proposed approach is also evaluated on one real-world optimization problem from CEC2011 [53], which we will describe it later in Experiment 4.

6.1.2. Algorithms for comparison

Several algorithms have been chosen to compare with the proposed method. These algorithms include a classical DE, and three recent or well-known DE variants listed below:

- DE/rand/1/bin with a population size *NP*=100, *F*=0.5, and *CR*=0.9 (DE).
- Self-adapting control parameters in differential evolution (jDE) [20].
- Differential evolution algorithm with strategy adaptation for global numerical optimization (SaDE) [14].

- Differential evolution algorithm with ensemble of parameters and mutation strategies (EPSDE) [4].

For all contestant DE variants, except for DE/rand/1/bin, all relevant parameters are set in accordance with their original papers.

Table 1. Test problems used in the experiment of this paper.

Function	Name	Formula	D	Search domain	f_{min}	accuracy
f_1	Sphere	$\sum_{i=1}^{D} x_i^2$	30	$[-100,100]^D$	0	1e-10
f_2	Rastrigin	$10D + \sum_{i=1}^{D}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	30	$[-5.12,5.12]^D$	0	1e-10
f_3	Camelback	$4x_1^2 - 2.1x_1^2 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 - 4x_2^4$	2	$[-5,5]^2$	-1.031628	1e-10
f_4	Rosenbrock	$\sum_{i=1}^{D-1}\left[100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right]$	30	$[-2,2]^D$	0	1e-10
f_5	Ackley	$-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right) + 20 + \mathrm{e}$	30	$[-32,32]^D$	0	1e-10
f_6	Griewank	$\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600,600]^D$	0	1e-10
f_7	Salomon	$-\cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + \frac{1}{10}\sqrt{\sum_{i=1}^{D} x_i^2} + 1$	30	$[-100,100]^D$	0	1e-10
f_8	Schwefel	$\sum_{i=1}^{D} -x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	30	$[-512,512]^D$	-418.9829D	1e-10
f_9	Quartic Function	$\sum_{i=1}^{D} ix_i^4 + rand$	30	$[-1.28,1.28]^D$	0	1e-10
f_{10}	Rotated hyper-ellipsoid	$\sum_{i=1}^{D}\left(\sum_{j=1}^{i} x_j\right)^2$	30	$[-100,100]^D$	0	1e-10
f_{11}	Easom	$-\cos x_1 - \cos x_2 \exp(-(x_1-\pi)^2 - (x_2-\pi)^2)$	2	$[-100,100]^2$	-1	1e-10
f_{12}	Goldstein and Price	$\left(1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right) \times \left(30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right)$	2	$[-2,2]^2$	3	1e-10
f_{13}	Shekel function	$-\sum_{i=1}^{m}[(x_i - a_i)(x_i - a_i)^T + c_i]^{-1} \quad , with\ m = 10, where \quad a = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix},\ c = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{bmatrix}$	4	$[0,10]^4$	-10.5364	1e-10

Table 1. Continued

Function	Name	Formula	D	Search domain	f_{min}	accuracy
f_{14}	Levy	$\frac{\pi}{D}\{10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_D-1)^2\}$ $where \quad y_i=1+\frac{1}{4}(x_i-1)$	30	$[-10,10]^D$	0	1e-10
f_{15}	Shifted Sphere	$\sum_{i=1}^{D} z_i^2+f_{bias1}$ $,f_{bias1}=-450 \ ,\vec{Z}=\vec{X}-\vec{O} \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-100,100]^D$	-450	1e-06
f_{16}	Shifted Schwefel's Problem 1.2	$\sum_{i=1}^{D}\left(\sum_{j=1}^{i} z_j\right)^2+f_{bias2}$ $,f_{bias2}=-450 \ ,\vec{Z}=\vec{X}-\vec{O} \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-100,100]^D$	-450	1e-06
f_{17}	Shifted Rotated High Conditioned Elliptic Function	$\sum_{i=1}^{D}(10^6)^{\frac{i-1}{D-1}}z_i^2+f_{bias3}$ $,f_{bias3}=-450 \ ,\vec{Z}=(\vec{X}-\vec{O})\times M_{orth} \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-100,100]^D$	-450	1e-06
f_{18}	Shifted Schwefel's Problem 1.2 with Noise in Fitness	$\left(\sum_{i=1}^{D}\left(\sum_{j=1}^{i} z_j\right)^2\right)\times(1+0.4\lvert N(0,1)\rvert)+f_{bias4}$ $,f_{bias4}=-450 \ ,\vec{Z}=(\vec{X}-\vec{O}) \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-100,100]^D$	-450	1e-06
f_{19}	Schwefel's Problem 2.6 with Global Optimum on Bounds	$f_{basic}=max\{x_1+2x_1-7,2x_1+x_2-5\}$,extend to D dimension $f_{19}=max\{A_iX-B_i\}+f_{bias5} \ ,f_{bias5}=-310 \ ,B_i=A_i\times o$ o is a $D\times 1$ vector with random number in $[-100,100]$, i is a row	30	$[-100,100]^D$	-310	1e-06
f_{20}	Shifted Rosenbrock's Function	$\sum_{i=1}^{D-1}\left(100\left(z_i^2-z_{i+1}\right)^2+(z_i-1)^2\right)+f_{bias6}$ $,f_{bias6}=390 \ ,\vec{Z}=(\vec{X}-\vec{O})+1$	30	$[-100,100]^D$	390	1e-02
f_{21}	Shifted Rotated Griewank's Function without Bounds	$\sum_{i=1}^{D}\frac{z_i^2}{4000}-\prod_{i=1}^{D}\cos\left(\frac{z_i}{\sqrt{i}}\right)+1+f_{bias7}$ $,f_{bias7}=-180 \ ,\vec{Z}=(\vec{X}-\vec{O})\times M \ ,M=M'(1+0.3\lvert N(0,1)\rvert)$	30	R	-180	1e-02
f_{22}	Shifted Rotated Ackley's Function with Global Optimum on Bounds	$-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}z_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right)+20+e+$ $f_{bias8} \ ,f_{bias8}=-140 \ ,\vec{Z}=(\vec{X}-\vec{O})\times M_{100}$	30	$[-32,32]^D$	-140	1e-02
f_{23}	Shifted Rastrigin's Function	$\sum_{i=1}^{D}\left(z_i^2-10\cos(2\pi z_i)+10\right)+f_{bias9}$ $,f_{bias9}=-330 \ ,\vec{Z}=\vec{X}-\vec{O} \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-5,5]^D$	-330	1e-02
f_{24}	Shifted Rotated Rastrigin's Function	$\sum_{i=1}^{D}\left(z_i^2-10\cos(2\pi z_i)+10\right)+f_{bias10}$ $,f_{bias10}=-330 \ ,\vec{Z}=(\vec{X}-\vec{O})\times M_2 \ ,\vec{O}=[o_1,o_2,\dots,o_D]$	30	$[-5,5]^D$	-330	1e-02

O is the shifted global optimum, M_{orth} is orthogonal matrix, N is a normal random number, A is a $D\times D$ dimensional matrix with random integer elements in [-500,500] where $det(A)\neq 0$, M , is the linear transformation matrix with condition number 3, and M_n is the transformation matrix with condition number n.

6.1.3. Parameters settings for ADE-Grid

The parameter settings for ADE-Grid have been explained explicitly in Section 5. However, for the sake of completeness, we describe it here again. Many experiments were carried out to examine the effect of different parameter settings on the performance of ADE-Grid. The following settings are adopted in the experiments of this paper. A 10×10 (i.e. n = 10) grid is used which corresponds to a population size NP=100. For each learning automaton, $L_{R\varepsilon P}$ is used with learning parameters a=0.1 and b=0.05.

6.1.4. Simulation settings

The following settings are adopted for all experiments of this paper. For a fair comparison among different methods, the maximum number of Function Evaluations (FEs) was set to 300000. All experiments on each function were run 30 times. All the algorithms were executed on an Intel Pentium Dual core with 2.5 GHz CPU and 4 GB of memory.

The average and standard deviation of the function error values $f(\vec{x}_{best}) - f(\vec{x}_{opt})$ among 30 independent runs recorded for each benchmark function, where $f(\vec{x}_{best})$ is the best solution 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 portion of successful runs among 30 trials. A run is successful if its function error value is smaller than the target error accuracy level ε, which is set to 10^{-10} for test functions f_1-f_{14}, 10^{-6} for the test functions f_{15}-f_{19}, and 10^{-2} for the rest of the test functions.

6.2. Experimental investigation of ADE-Grid

6.2.1. Experiment 1: effect of the neighborhood structure on the performance of ADE-Grid

This experiment aims to investigate the effect of the neighborhood structure on the performance of ADE-Grid. The results of ADE-Grid with different neighborhood structure are given in Table 2. In Table 2, the results of the best performing algorithm, which are significantly superior to the others with a t-test at 5% significant level, are printed in boldface. In situations where the outcomes of the *t*-test reveal that the results of the best performing algorithms are not statically significant, all of them are printed in boldface.

Table 2. Results of ADE-Grid with different neighborhood structure on the set of test problems.

Func.	ADE-Grid$_{Moor}$	ADE-Grid$_{von\text{-}Neumann}$	ADE-Grid$_{Cole}$	ADE-Grid$_{Smith}$
f_1	**8.51E-108 ± 1.02E-107**	2.06E-103 ± 2.25E-103	2.94E-101 ± 5.35E-101	1.99E-98 ± 3.38E-98
f_2	**0.00E+00 ± 0.00E+00**	3.78E-15 ± 2.07E-14	1.26E-13 ± 6.74E-13	3.16E-07 ± 1.73E-06
f_3	8.25E-14 ± 4.10E-13	8.83E-14 ± 3.67E-13	**7.11E-15 ± 3.86E-14**	1.73E-13 ± 7.37E-13
f_4	**5.62E-18 ± 1.59E-17**	3.80E-16 ± 1.21E-15	1.16E-14 ± 5.92E-14	1.32E-01 ± 7.27E-01
f_5	**4.44E-15 ± 0.00E+00**	**4.44E-15 ± 0.00E+00**	**4.44E-15 ± 0.00E+00**	**4.44E-15 ± 0.00E+00**
f_6	**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_7	**1.13E-01 ± 3.46E-02**	1.18E-01 ± 3.90E-02	1.29E-01 ± 4.66E-02	1.26E-01 ± 4.50E-02
f_8	**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_9	1.30E-03 ± 2.71E-04	1.30E-03 ± 3.10E-04	1.30E-03 ± 3.29E-04	**1.20E-03 ± 3.01E-04**
f_{10}	**4.23E-16 ± 4.57E-16**	8.23E-15 ± 1.86E-14	1.16E-14 ± 2.19E-14	5.02E-14 ± 8.78E-14
f_{11}	**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_{12}	**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_{13}	1.85E-11 ± 1.62E-11	2.09E-11 ± 1.90E-11	1.63E-11 ± 1.42E-11	**1.55E-11 ± 1.48E-11**
f_{14}	**1.49E-32 ± 1.11E-47**	**1.49E-32 ± 1.11E-47**	**1.49E-32 ± 1.11E-47**	**1.49E-32 ± 1.11E-47**
f_{15}	**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_{16}	**1.00E-15 ± 1.79E-15**	1.53E-14 ± 3.20E-14	2.45E-14 ± 3.20E-14	1.00E-13 ± 1.10E-13
f_{17}	1.58E+05 ± 9.60E+04	**1.49E+05 ± 9.66E+04**	1.83E+05 ± 1.00E+05	2.17E+05 ± 1.19E+05
f_{18}	**4.80E-08 ± 1.32E-07**	1.27E-07 ± 2.00E-07	4.13E-07 ± 9.87E-07	3.54E-07 ± 6.31E-07
f_{19}	1.51E+01 ± 6.16E+01	2.86E+00 ± 6.88E+00	**1.11E+00 ± 3.27E+00**	4.97E+00 ± 1.34E+01
f_{20}	7.97E-01 ± 1.62E+00	**2.65E-01 ± 1.01E+00**	3.98E-01 ± 1.21E+00	3.98E-01 ± 1.21E+00
f_{21}	1.52E-02 ± 8.60E-03	**1.46E-02 ± 9.50E-03**	1.71E-02 ± 1.63E-02	1.82E-02 ± 1.31E-02
f_{22}	**2.09E+01 ± 3.71E-02**	**2.09E+01 ± 5.43E-02**	**2.09E+01 ± 5.80E-02**	**2.09E+01 ± 4.67E-02**
f_{23}	**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_{24}	1.05E+02 ± 9.86E+00	1.06E+02 ± 1.16E+01	**1.02E+02 ± 1.18E+01**	1.09E+02 ± 8.29E+00

From Table 2, we can observe that the results obtained by ADE-Grid$_{moor}$ are better than those achieved by the other ADE-Grid variants. The reason is mainly lies in the fact that the neighborhood structure of the cells in ADE-Grid can affect the movement of genomes toward several promising areas of the search space. This means that the communication structure of genomes can influence both the cooperation among the population and the self-adaption of the individuals, and further affect the effectiveness of the algorithms. Considering neighborhood structure of the cells, we can establish a balance between diversification capability and intensification capability of the ADE-Grid. As it can be seen in Table 2, moor neighborhood structure is the most suitable topology for optimizing the set of test problems.

6.2.2. Experiment 2: comparison with other state-of-the-art DE variants

In this experiment the performance of ADE-Grid in terms of reaching to global optimum, success rate, and speed of convergence is compared with that of other recent successful variants of DE. Table 3 summarizes the optimization results in terms of mean and standard deviation of the function error values of the best solutions found by different approaches over 30 independent trials. To validate the statistical difference between ADE-Grid and other DE variants, we have performed a *t*-test with a 5% significance level on the results of the ADE-Grid and the other contestant DE algorithm, on each test function. From Table 3, it is clear that ADE-Grid is superior to the other DE-based algorithms. Comparing to DE, jDE, and SaDE, the performance of ADE-Grid is significantly better on most of the tested problems. Considering the results of the *t*-test in Table 3, ADE-Grid is collectively performed better than EPSDE in 45% of the test functions. However, EPSDE can only outperform ADE-Grid in 33% of the test functions.

For each algorithm, the success rate along with the average number of function evaluations required to reach to the specified accuracy on the given problems are also shown in Table 4. Considering problems f_9, f_{17}, f_{22} and f_{24}, none of the compared algorithms have reached to the specified accuracy. From Table 4, it is observed that ADE-Grid is the only algorithm that can effectively solve most of the problems. Considering problems f_{18} and f_{19}, it is only ADE-Grid that is able to achieve successful runs on these problems. Considering the success rates in Table 4, it is apparent that ADE-Grid is a robust algorithm. Based on the results of Table 4, we provide the success rate and the average number of function evaluation ranks of each method in Table 5. As can be seen in Table 5, although ADE-Grid is not the best performing algorithm in terms of approaching speed to the global optima, it is the most successful algorithm among the other contestant methods.

Figures 4, 5 and 6, depict the evolution curve (the mean error values versus the number of function evaluations) for different DE algorithms. The results also indicate that ADE-Grid has a weaker convergence rate than approaches like SaDE and EPSDE on some benchmark functions, but the convergence rate of ADE-Grid is acceptable and it produces competitive results in comparison with the best algorithm.

Table 3. Mean ± standard deviation of the function error values of DE, jDE, SaDE, EPSDE and ADE-Grid over 30 independent runs.

Func.	DE	jDE	SaDE	EPSDE	ADE-Grid$_{Moor}$
f_1	8.70E-32 ± 7.27E-32 $^{+}$	9.02E-62 ± 1.32E-61 $^{+}$	2.58E-131 ± 9.35E-135 $^{-}$	8.95E-172 ± 0.00E+00 $^{-}$	8.51E-108 ± 1.02E-107
f_2	1.36E+02 ± 2.31E+01 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	6.63E-02 ± 2.52E-01 $^{+}$	3.31E-02 ± 1.81E-01 $^{+}$	0.00E+00 ± 0.00E+00
f_3	0.00E+00 ± 0.00E+00 $^{-}$	0.00E+00 ± 0.00E+00 $^{-}$	1.79E-12 ± 1.94E-12 $^{+}$	9.17E-16 ± 5.02E-15 $^{-}$	8.25E-14 ± 4.10E-13
f_4	3.73E-01 ± 3.64E-01 $^{+}$	7.84E+00 ± 6.61E-01 $^{+}$	2.19E+01 ± 1.06E+00 $^{+}$	2.65E-01 ± 1.01E+00 $^{+}$	5.62E-18 ± 1.59E-17
f_5	6.09E-15 ± 1.80E-15 $^{+}$	5.38E-15 ± 1.59E-15 $^{+}$	6.20E-02 ± 2.36E-01 $^{+}$	3.10E-02 ± 1.70E-01 $^{+}$	4.44E-15 ± 0.00E+00
f_6	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	4.67E-03 ± 1.04E-02 $^{+}$	6.55E-04 ± 3.59E-03 $^{+}$	0.00E+00 ± 0.00E+00
f_7	1.74E-01 ± 4.21E-02 $^{+}$	1.93E-01 ± 2.54E-02 $^{+}$	2.63E-01 ± 5.56E-02 $^{+}$	1.99E-01 ± 4.54E-02 $^{+}$	1.13E-01 ± 3.46E-02
f_8	8.03E+01 ± 7.82E+01 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	6.35E-04 ± 6.77E-04 $^{+}$	0.00E+00 ± 0.00E+00
f_9	4.24E-03 ± 1.44E-03 $^{+}$	3.50E-03 ± 9.46E-04 $^{+}$	2.78E-03 ± 1.12E-03 $^{+}$	9.43E-04 ± 3.88E-04 $^{-}$	1.30E-03 ± 2.71E-04
f_{10}	2.54E-05 ± 2.57E-05 $^{+}$	1.93E-07 ± 2.76E-07 $^{+}$	5.35E-07 ± 1.60E-06 $^{+}$	8.12E-36 ± 4.45E-35 $^{-}$	4.23E-16 ± 4.57E-16
f_{11}	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	8.28E-06 ± 4.53E-05 $^{+}$	0.00E+00 ± 0.00E+00
f_{12}	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00
f_{13}	4.47E-11 ± 4.57E-11 $^{+}$	1.27E-11 ± 1.01E-11 $^{-}$	2.09E-11 ± 1.81E-11 $^{+}$	2.23E-01 ± 1.22E+00 $^{+}$	1.85E-11 ± 1.62E-11
f_{14}	1.49E-32 ± 1.11E-47 $^{\approx}$	1.49E-32 ± 1.11E-47 $^{\approx}$	4.22E-02 ± 1.27E-01 $^{+}$	1.49E-32 ± 1.11E-47 $^{\approx}$	1.49E-32 ± 1.11E-47
f_{15}	2.86E-29 ± 6.97E-29 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	6.73E-30 ± 3.68E-29 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00
f_{16}	4.13E-05 ± 3.92E-05 $^{+}$	4.35E-07 ± 4.35E-07 $^{+}$	6.11E-06 ± 1.30E-05 $^{+}$	2.66E-17 ± 1.46E-16 $^{-}$	1.00E-15 ± 1.79E-15
f_{17}	4.82E+05 ± 3.35E+05 $^{+}$	1.84E+05 ± 1.15E+04 $^{+}$	4.66E+05 ± 1.96E+05 $^{+}$	9.69E+05 ± 4.15E+06 $^{+}$	1.58E+05 ± 9.60E+04
f_{18}	1.66E-02 ± 1.38E-02 $^{+}$	3.70E-02 ± 8.10E-02 $^{+}$	1.51E+02 ± 1.79E+02 $^{+}$	1.63E+01 ± 6.77E+01 $^{+}$	4.80E-08 ± 1.32E-07
f_{19}	7.23E-01 ± 7.16E-01 $^{-}$	4.53E+02 ± 4.08E+02 $^{+}$	3.27E+03 ± 5.94E+02 $^{+}$	1.41E+03 ± 7.51E+02 $^{+}$	1.51E+01 ± 6.16E+01
f_{20}	2.35E+00 ± 1.24E+00 $^{+}$	3.27E+01 ± 2.85E+01 $^{+}$	4.04E+01 ± 3.16E+01 $^{+}$	6.64E-01 ± 1.51E+00 $^{-}$	7.97E-01 ± 1.62E+00
f_{21}	4.10E-04 ± 2.24E-03 $^{-}$	1.00E-02 ± 8.51E-03 $^{-}$	1.48E-02 ± 1.25E-02 $^{-}$	1.36E-02 ± 1.27E-02 $^{-}$	1.52E-02 ± 8.60E-03
f_{22}	2.09E+01 ± 4.72E-02 $^{\approx}$	2.09E+01 ± 5.23E-02 $^{\approx}$	2.09E+01 ± 3.88E-02 $^{\approx}$	2.09E+01 ± 5.03E-02 $^{\approx}$	2.09E+01 ± 3.71E-02
f_{23}	1.23E+02 ± 2.56E+01 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	6.63E-02 ± 2.52E-01 $^{+}$	0.00E+00 ± 0.00E+00 $^{\approx}$	0.00E+00 ± 0.00E+00
f_{24}	1.82E+02 ± 8.00E+00 $^{+}$	5.57E+01 ± 8.89E+00 $^{-}$	4.74E+01 ± 1.01E+01 $^{-}$	4.63E+01 ± 9.65E+00 $^{-}$	1.05E+02 ± 9.86E+00
+	16	11	17	11	
-	3	4	3	8	
≈	5	9	4	5	

‘‘+’’ and ‘‘-’’ indicate a 0.05 level of significance by *t*-test. ‘‘+’’, ‘‘-’’ and ‘‘≈’’ denote that the performance of ADE-Grid is better than, worse than, and similar to that of counterpart algorithms, respectively.

Table 4. Average number of function evaluations to reach the specified accuracy for successful runs (Mean success rate).

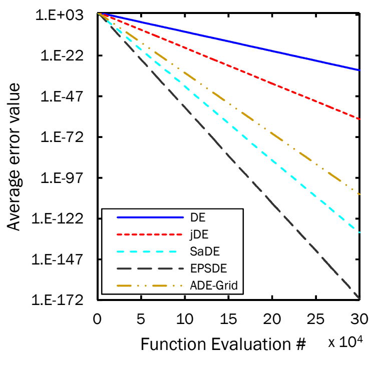
Func.	DE	jDE	SaDE	EPSDE	ADE-Grid$_{Moor}$
f_1	121670 (**1**)	67050 (**1**)	31035 (**1**)	**24843** (**1**)	39170 (**1**)
f_2	– (0)	123370 (**1**)	**68615** (0.93)	69843 (0.96)	263690 (**1**)
f_3	9293 (**1**)	10540 (**1**)	16018 (**1**)	**5786** (**1**)	13990 (**1**)
f_4	– (0)	– (0)	– (0)	**123820** (0.93)	237240 (**1**)
f_5	195200 (**1**)	105450 (**1**)	52698 (0.93)	**38430** (0.96)	63767 (**1**)
f_6	125350 (**1**)	69860 (**1**)	**2321** (0.9)	25415 (0.96)	40783 (**1**)
f_7	– (0)	– (0)	– (0)	– (0)	– (0)
f_8	**25880** (0.06)	88863 (**1**)	51313 (**1**)	– (0)	115600 (**1**)
f_9	– (0)	– (0)	– (0)	– (0)	– (0)
f_{10}	– (0)	– (0)	– (0)	**97930** (**1**)	217500 (**1**)
f_{11}	16990 (**1**)	21553 (**1**)	**10918** (**1**)	20885 (0.96)	28117 (**1**)
f_{12}	5366 (**1**)	7566 (**1**)	4640 (**1**)	**3690** (**1**)	6213 (**1**)
f_{13}	63990 (0.83)	**50327** (**1**)	59345 (**1**)	63302 (0.96)	74093 (**1**)
f_{14}	113410 (**1**)	58300 (**1**)	31983 (0.83)	**27647** (**1**)	36733 (**1**)
f_{15}	87033 (**1**)	49327 (**1**)	24482 (**1**)	**19213** (**1**)	29633 (**1**)
f_{16}	– (0)	243530 (0.86)	138610 (0.43)	**86240** (**1**)	161090 (**1**)
f_{17}	– (0)	– (0)	– (0)	– (0)	– (0)
f_{18}	– (0)	– (0)	– (0)	– (0)	**247610** (**1**)
f_{19}	– (0)	– (0)	– (0)	– (0)	**39286** (**0.06**)
f_{20}	**22824** (0.06)	– (0)	– (0)	92672 (0.83)	149920 (0.8)
f_{21}	92773 (0.96)	94003 (0.83)	107530 (0.73)	**22203** (0.63)	34233 (0.53)
f_{22}	– (0)	– (0)	– (0)	– (0)	– (0)
f_{23}	– (0)	81007 (**1**)	**50305** (0.93)	56058 (**1**)	223490 (**1**)
f_{24}	– (0)	– (0)	– (0)	– (0)	– (0)

It is worth noticing that in functions f_5, f_7, f_{17} and f_{18} ADE-Grid obtained the best results with the highest convergence rate. From obtained results in Tables 3, 4 and 5, it can be observed that ADE-Grid is a significant improvement over the plain DE in terms of both the quality of final solutions and speed of convergence.

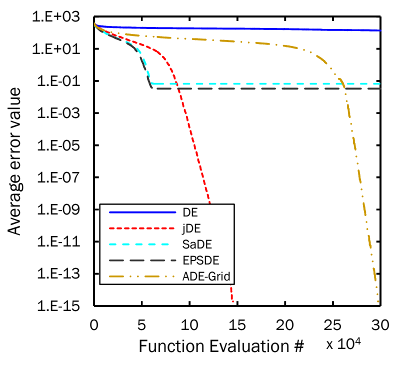
Table 5. Average function evaluation ranks (success rate ranks) for each method.

Func.	DE	jDE	SaDE	EPSDE	ADE-Grid$_{Moor}$
f_1	5 (1)	4 (1)	2 (1)	1 (1)	3 (1)
f_2	5 (4)	3 (1)	1 (3)	2 (2)	4 (1)
f_3	2 (1)	3 (1)	5 (1)	1 (1)	4 (1)
f_4	3 (3)	3 (3)	3 (3)	1 (2)	2 (1)
f_5	5 (1)	4 (1)	2 (3)	1 (2)	3 (1)
f_6	5 (1)	4 (1)	1 (3)	2 (2)	3 (1)
f_7	1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
f_8	1 (2)	3 (1)	2 (1)	5 (3)	4 (1)
f_9	1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
f_{10}	3 (2)	3 (2)	3 (2)	1 (1)	2 (1)
f_{11}	2 (1)	4 (1)	1 (1)	3 (2)	5 (1)
f_{12}	3 (1)	5 (1)	2 (1)	1 (1)	4 (1)
f_{13}	4 (3)	1 (1)	2 (1)	3 (2)	5 (1)
f_{14}	5 (1)	4 (1)	2 (2)	1 (1)	3 (1)
f_{15}	5 (1)	4 (1)	2 (1)	1 (1)	3 (1)
f_{16}	5 (4)	4 (2)	2 (3)	1 (1)	3 (1)
f_{17}	1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
f_{18}	2 (2)	2 (2)	2 (2)	2 (2)	1 (1)
f_{19}	2 (2)	2 (2)	2 (2)	2 (2)	1 (1)
f_{20}	1 (3)	4 (4)	4 (4)	2 (1)	3 (2)
f_{21}	3 (1)	4 (2)	5 (3)	1 (4)	2 (5)
f_{22}	1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
f_{23}	5 (3)	3 (1)	1 (2)	2 (1)	4 (1)
f_{24}	1 (1)	1 (1)	1 (1)	1 (1)	1 (1)
Overall rank	2.95 (1.75)	2.87 (1.41)	2.04 (1.83)	**1.58** (1.54)	2.66 (**1.20**)

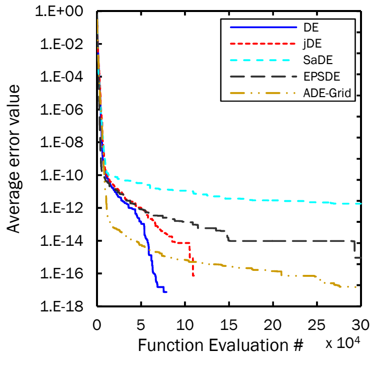
The reason for the success of ADE-Grid is that it can establish some compromise between the exploration and exploitation. The search process of ADE-Grid is based on learning successful trajectories for each individual of the population. Moreover, the adjustment of the parameters *F* and *CR* for each individual, based on its search progress, can ensure a good balance between global and local search throughout the course of optimization. These characteristics are favorable to the performance of ADE-Grid.



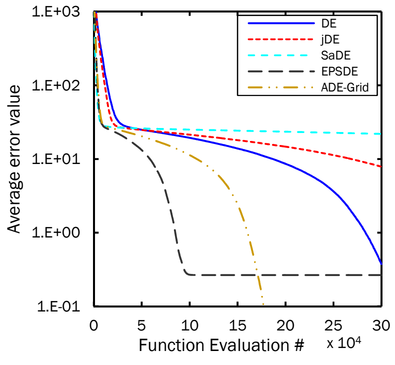
(a)



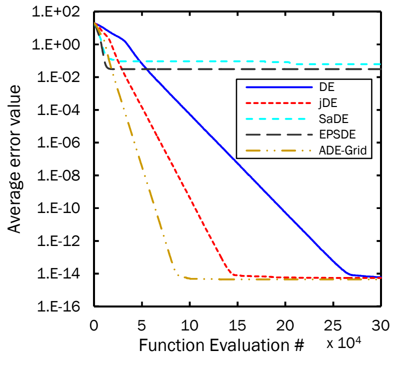
(b)



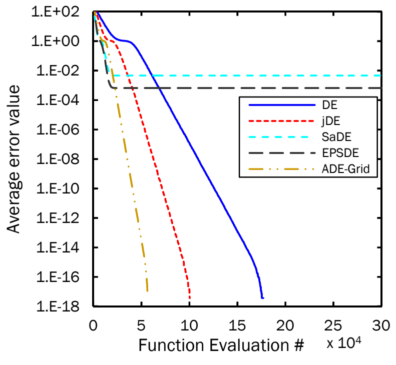
(c)



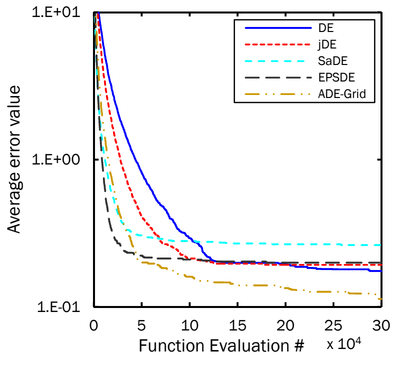
(d)



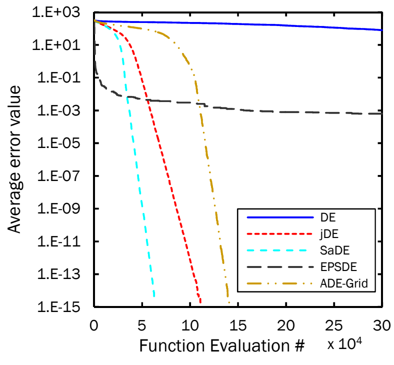
(e)



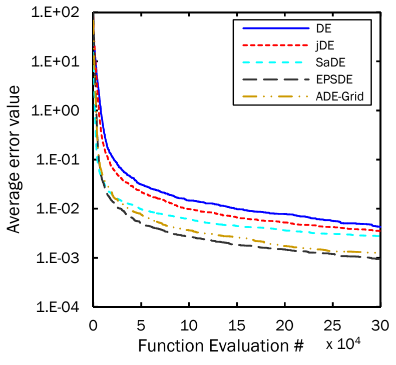
(f)



(g)



(h)



(i)

Fig. 4. Evolution of the mean error values versus the number of function evaluations for different DE variants on test functions (a) f_1, (b) f_2, (c) f_3, (d) f_4, (e) f_5, (f) f_6, (g) f_7, (h) f_8, and (i) f_9.

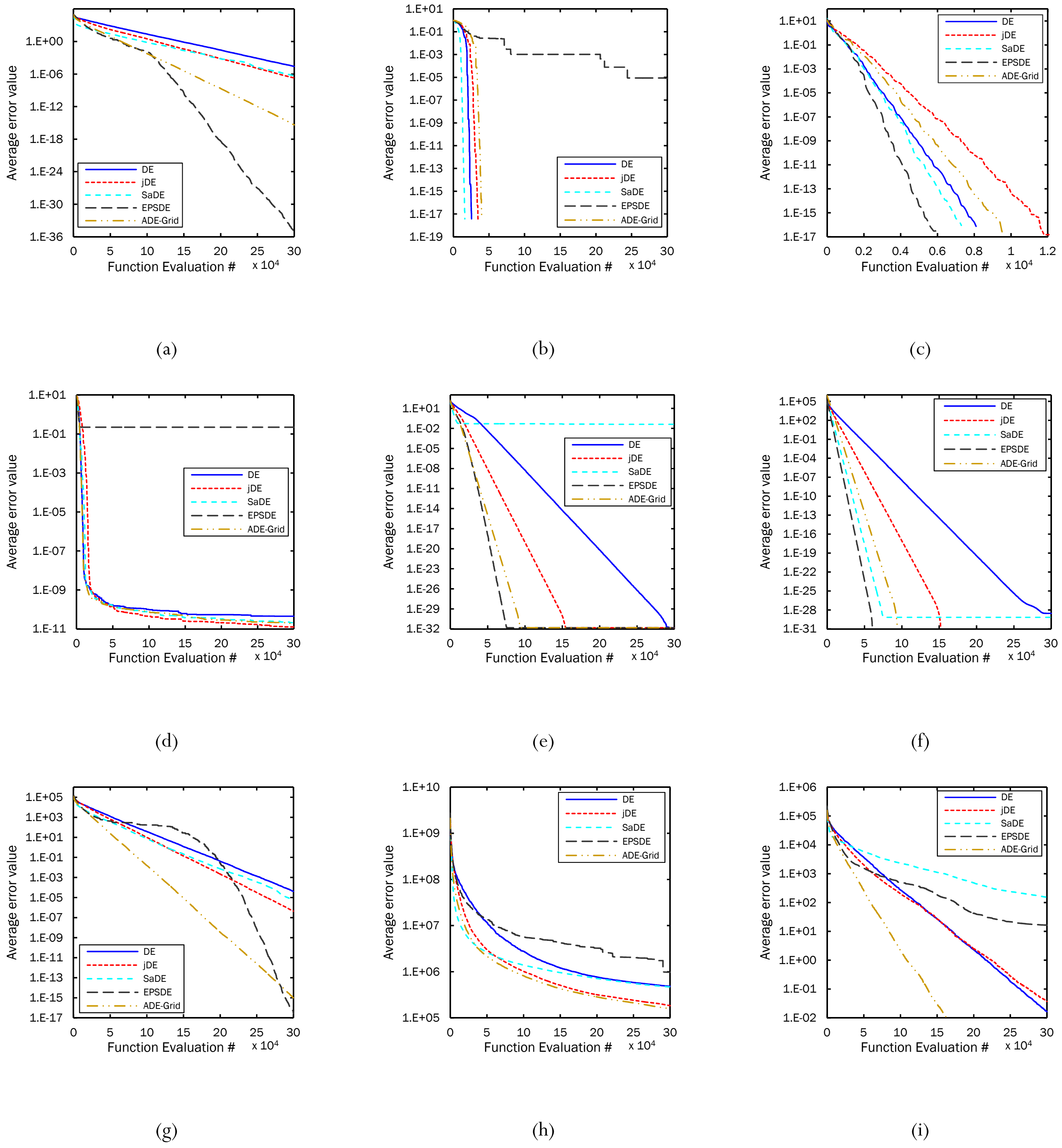


Fig. 5. Evolution of the mean error values versus the number of function evaluations for different DE variants on test functions (a) f_{10}, (b) f_{11}, (c) f_{12}, (d) f_{13}, (e) f_{14}, (f) f_{15}, (g) f_{16} , (h) f_{17} , and (i) f_{18}.

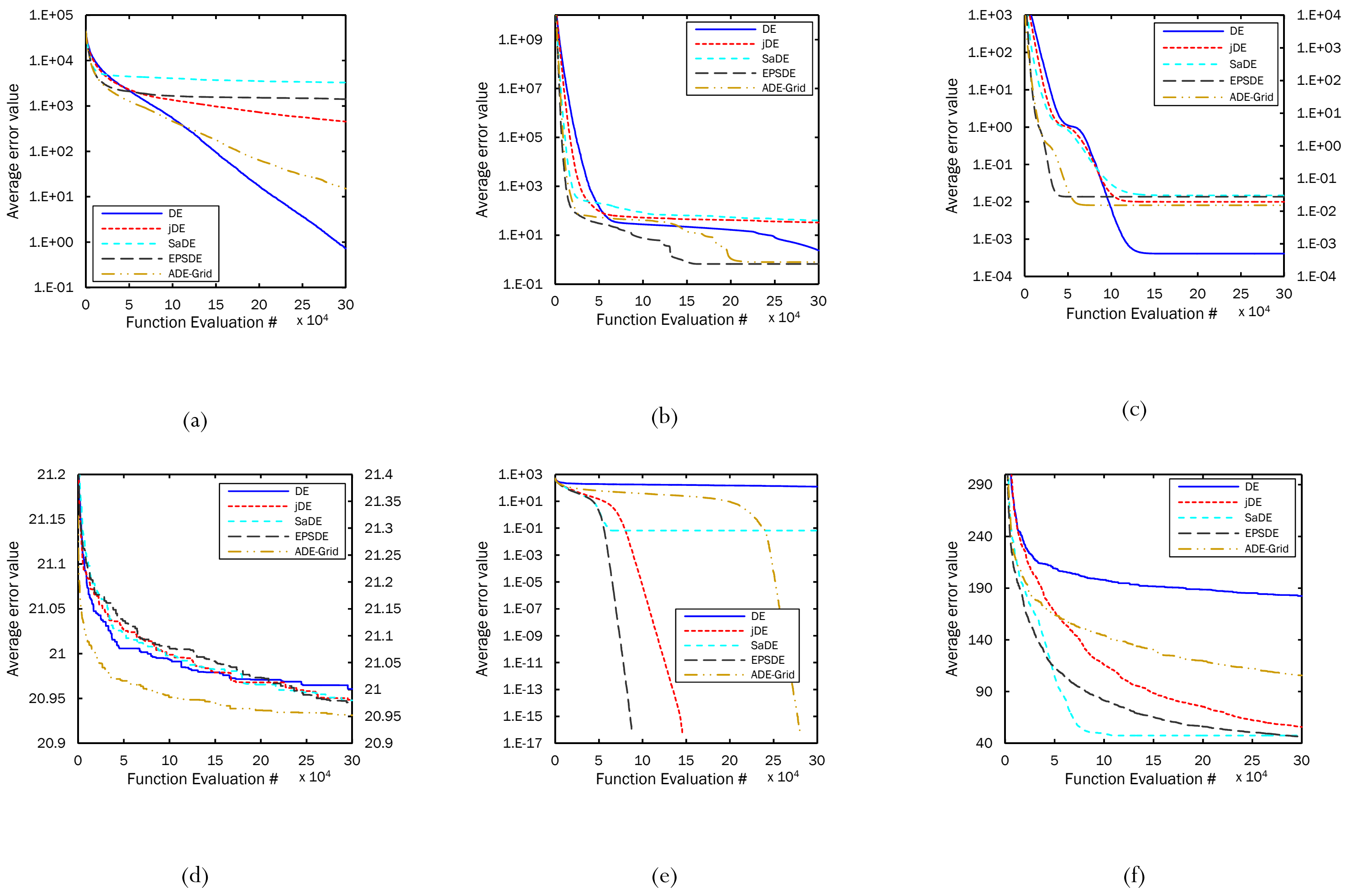


Fig. 6. Evolution of the mean error values versus the number of function evaluations for different DE variants on test functions (a) f_{19}, (b) f_{20}, (c) f_{21}, (d) f_{22}, (e) f_{23}, and (f) f_{24}.

6.2.3. Experiment 3: Runtime complexity of ADE-Grid

In order to evaluate the runtime complexity of our proposed algorithm, we use the method based on [52]. Table 6 reports the computational complexity of ADE-Grid$_{\text{Moor}}$ 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 ADE-Grid$_{\text{Moor}}$. In Table 6, T_0 is the computing time needed for a test program depicted in Fig. 7, T_1 is the computing time of the function f_{17} for 200000 evaluations of a certain dimension D and $\hat{T}_2$ is the average computing time for the 5 runs of the algorithm with 200000 evaluations of the same D dimensional function f_{17}.

Table 6. Computational complexity of ADE-Grid$_{\text{Moor}}$ in comparison with EPSDE

Dim	T_0	T_1	$\hat{T}_2$		$(\hat{T}_2 - T_1) / T_0$	
Method	-	-	EPSDE	ADE-Grid$_{\text{Moor}}$	EPSDE	ADE-Grid$_{\text{Moor}}$
10	0.17	12.09	26.70	16.00	85.94	23
30	0.17	15.00	41.14	21.76	153.76	39.76
50	0.17	17.61	66.23	30.60	286	76.41

From Table 6, it is clear that ADE-Grid$_{\text{Moor}}$ is more computationally efficient than EPSDE.

```
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. 7. The test program for computing T_0

6.2.4. Experiment 4: real-world optimization problem

In this sub-section, the effectiveness of the ADE-Grid is studied on a real-world parameter estimation problem, namely, parameter estimation for frequency modulated sound waves [53]. Here, the aim is to approximate the target sound wave Eq. (14) with the sound wave Eq. (13).

$$y(t) = a_1 \times \sin(\omega_1 . t . \theta + a_2 \times \sin(\omega_2 . t . \theta + a_3 \times \sin(\omega_3 . t . \theta))) \tag{13}$$

$$y_0(t) = (1.0) \times \sin\Big((5.0) . t . \theta + (-1.5) \times \sin\big((4.8) . t . \theta + (2.0) \times \sin((4.9) . t . \theta)\big)\Big) \tag{14}$$

Therefore, the task of optimization in this problem is to estimate a six-dimensional vector $\vec{X} = \{a_1, \omega_1, a_2, \omega_2, a_3, \omega_3\}$ with the aim to minimize the following fitness function:

$$f = \sum_{t=0}^{100} (y(t) - y_0(t))^2 \tag{15}$$

This problem, which has an important role in several modern music systems, is a highly complex multimodal instance with minimum value $f_{\min} = 0$. The results of ADE-Grid, EA-DE-Mimetic [54], GA-SBX [55], SAMODE [56], and DE-Λ_{Cr} [57] on the frequency modulated sound waves are summarized in Table 7. For ADE-Grid, the results are reported based on 25 independent runs. Results of the other methods are directly taken from their respective papers.

Table 7. Optimization results of parameter estimation for frequency modulated sound waves.

FEs	Function value	EA-DE-Mimetic	GA-SBX	SAMODE	DE-Λ_{Cr}	ADE-Grid
5×10^4	Best	1.1674E-11	1.6775E-04	2.5397E-11	1.6147E-10	**5.2316E-27**
	Median	1.0167E+01	–	8.9616E-06	**3.8779E-09**	9.2467E-05
	Worst	1.5440E+01	1.4527E+01	1.2547E+01	1.4813E+01	**1.1612E+01**
	Mean	7.6022E+00	4.2007E+00	3.0769E+00	**2.1870E+00**	3.0327E+00
	Std	6.0205E+00	–	5.0495E+00	4.6326E+00	3.7850E+00
1×10^5	Best	1.1674E-11	6.7922E-05	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	1.4526E+01	1.0942E+01	1.1757E+01	**4.7082E+00**
	Mean	3.8056E+00	4.1978E+00	1.2120E+00	1.0978E+00	**2.4852E-01**
	Std	5.2091E+00	–	3.3762E+00	3.1751E+00	9.7344E-01
1.5×10^5	Best	1.1674E-11	6.7922E-05	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	1.4526E+01	1.0942E+01	1.1757E+01	**0.0000E+00**
	Mean	2.0948E+00	4.1976E+00	1.2120E+00	8.7697E-01	**0.0000E+00**
	Std	4.3063E+00	–	3.3762E+00	3.0439E+00	0.0000E+00

From Table 7, it can be observed that ADE-Grid provides the best results.

8. Conclusion and future works

A new variant of DE is presented in this paper which aims at controlling the diversity and greediness of the genomes by adapting the control parameters and trial vector generation strategy in DE. In this approach, the concept of learning automata is employed to let genomes of the population autonomously determine their crossover rate, scale factor, and mutation strategy. To achieve this goal, the genomes of the population are placed on distinct cells of a mesh grid. Three learning automata are assigned to each individual to adjust its control parameters and mutation strategy. Each automaton selects one of its own actions based on its probability vector and determines the *CR*, *F* and mutation strategy for its corresponding individual. The automata then update their probability vector based on the signal received from the search progress of their corresponding individuals. A modified mutation strategy is also integrated into the trial vector generation cycle which adjusts the trajectory of the genomes toward their best neighborhood cells. Four different neighborhood types, i.e. Moore, von Neumann, Cole and Smith, were examined to determine the topology of communication among the genomes. The experimental studies were carried out on 24 well-known benchmark functions and one real-life engineering problem. Moreover, the proposed approach was compared with other state-of-the-art DE variants. The experimental results show the effectiveness of the proposed method.

Future research may focus on integrating the collaboration mechanisms among the neighbor individuals in the learning process. It would be very interesting to use continuous action-set learning automata for parameters *F* and *CR*. Moreover, it is also possible to consider other mutation strategies as the actions of the $LA_{mutation}$. Finally, considering the application of learning automata for adaptive selection of the type of crossover being used might be also beneficial.

Acknowledgement The authors would like to thank A. Rezvanian for his excellent insights, constructive comments and helpful suggestions on the initial version of the manuscript.

References

1. Fogel DB (1997) The Advantages of Evolutionary Computation. In: Proceedings of the International Conference on Bio-Computing and Emergent Computation. World Scientific, pp 1–11

2. Storn R, Price K (1995) 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

3. Storn R, Price K (1997) Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. J Glob Optim 11:341–359. doi: 10.1023/A:1008202821328

4. Mallipeddi R, Suganthan PN, Pan QK, Tasgetiren MF (2011) Differential evolution algorithm with ensemble of parameters and mutation strategies. Appl Soft Comput 11:1679–1696. doi: 10.1016/j.asoc.2010.04.024

5. Wang Y, Cai Z, Zhang Q (2011) Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans Evol Comput 15:55–66. doi: 10.1109/TEVC.2010.2087271

6. Vafashoar R, Meybodi MR, Momeni Azandaryani AH (2012) CLA-DE: a hybrid model based on cellular learning automata for numerical optimization. Appl Intell 36:735–748. doi: 10.1007/s10489-011-0292-1

7. Brest J, Zamuda A, Boskovic B, et al. (2009) Dynamic optimization using self-adaptive differential evolution. IEEE Congress on Evolutionary Computation. pp 415–422

8. Noroozi V, Hashemi AB, Meybodi MR (2012) Alpinist CellularDE: a cellular based optimization algorithm for dynamic environments. Proceedings of the fourteenth international conference on genetic and evolutionary computation conference companion. ACM, New York, pp 1519–1520

9. Kordestani JK, Rezvanian A, Meybodi M (2014) CDEPSO: a bi-population hybrid approach for dynamic optimization problems. Appl Intell 40:682–694. doi: 10.1007/s10489-013-0483-z

10. Xue F, Sanderson AC, Graves RJ (2003) Pareto-based multi-objective differential evolution. The 2003 Congress on Evolutionary Computation. pp 862–869

11. Santana-Quintero LV, Coello CAC (2005) An algorithm based on differential evolution for multi-objective problems. Int J Comput Intell Res 1:151–169.

12. Huang VL, Qin AK, Suganthan PN, Tasgetiren MF (2007) Multi-objective optimization based on self-adaptive differential evolution algorithm. IEEE Congress on Evolutionary Computation. pp 3601–3608

13. Gämperle R, Müller SD, Koumoutsakos P (2002) A parameter study for differential evolution. In: Grmela A, Mastorakis NE (eds) Advances in intelligent systems, fuzzy systems, evolutionary computation. WSEAS Press, Interlaken, Switzerland, pp 293–298

14. Qin AK, Huang VL, Suganthan PN (2009) Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput 13:398–417. doi: 10.1109/TEVC.2008.927706

15. Das S, Suganthan PN (2011) Differential evolution: A survey of the state-of-the-art. IEEE Trans Evol Comput 15:4–31. doi: 10.1109/TEVC.2010.2059031

16. Ali M, Pant M, Abraham A (2013) Unconventional initialization methods for differential evolution. Appl Math Comput 219:4474–4494. doi: 10.1016/j.amc.2012.10.053

17. Rahnamayan S, Tizhoosh HR, Salama MMA (2007) A novel population initialization method for accelerating evolutionary algorithms. Comp Math Appl 53:1605–1614. doi: 10.1016/j.camwa.2006.07.013

18. Rahnamayan S, Tizhoosh HR, Salama MM (2008) Opposition-based differential evolution. IEEE Trans Evol Comput 12:64–79. doi: 10.1109/TEVC.2007.894200

19. Das S, Konar A, Chakraborty UK (2005) Two improved differential evolution schemes for faster global search. Proceedings of the 2005 conference on genetic and evolutionary computation. ACM, Washington DC, USA, pp 991–998

20. Brest J, Greiner S, Boskovic B, et al. (2006) Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans Evol Comput 10:646–657. doi: 10.1109/TEVC.2006.872133

21. Brest J, Sepesy Maučec M (2008) Population size reduction for the differential evolution algorithm. Appl Intell 29:228–247. doi: 10.1007/s10489-007-0091-x

22. Liu J, Lampinen J (2005) A fuzzy adaptive differential evolution algorithm. Soft Comput 9:448–462. doi: 10.1007/s00500-004-0363-x

23. Zhang J, Sanderson AC (2009) JADE: adaptive differential evolution with optional external archive. IEEE Trans Evol Comput 13:945–958. doi: 10.1109/TEVC.2009.2014613

24. Gong W, Fialho Á, Cai Z (2010) Adaptive strategy selection in differential evolution. Proceedings of the 12th annual conference on genetic and evolutionary computation. ACM, Portland, Oregon, USA, pp 409–416

25. Sun J, Zhang Q, Tsang EPK (2005) DE/EDA: A new evolutionary algorithm for global optimization. Inf Sci 169:249–262. doi: 10.1016/j.ins.2004.06.009

26. Zhang W-J, Xie X-F (2003) DEPSO: hybrid particle swarm with differential evolution operator. IEEE International Conference on Systems, Man and Cybernetics. pp 3816–3821

27. Kennedy J, Eberhart R (1995) Particle swarm optimization. IEEE International Conference on Neural Networks. pp 1942–1948

28. Halder U, Das S, Maity D (2013) A Cluster-Based Differential Evolution Algorithm With External Archive for Optimization in Dynamic Environments. IEEE Trans Cybern 43:881 – 897. doi: 10.1109/TSMCB.2012.2217491

29. Storn R, Price KV, Lampinen J (2005) Differential Evolution-a practical approach to global optimization. Springer, Berlin, Germany

30. Price KV (1999) An introduction to differential evolution. In: Corne D, Dorigo M, Glover F, et al. (eds) New ideas in optimization. McGraw-Hill Ltd., UK, London, pp 79–108

31. Wang Y, Cai Z, Zhang Q (2012) Enhancing the search ability of differential evolution through orthogonal crossover. Inf Sci 185:153–177. doi: 10.1016/j.ins.2011.09.001

32. Das S, Konar A, Chakraborty UK (2007) Annealed differential evolution. IEEE Congress on Evolutionary Computation. pp 1926–1933

33. Kennedy J (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In Proceedings of the 1999 Congress on Evolutionary Computation. pp 1931–1938

34. Kennedy J, Mendes R (2002) Population structure and particle swarm performance. Proceedings of Congress on Evolutionary Computation. pp 1671–1676

35. Kennedy J, Mendes R (2006) Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. IEEE Trans Syst Man Cybern Part C Appl Rev 36:515–519. doi: 10.1109/TSMCC.2006.875410

36. Mendes R, Kennedy J, Neves J (2003) Watch thy neighbor or how the swarm can learn from its environment. pp 88–94

37. Omran MH, Engelbrecht A, Salman A (2006) Using the Ring Neighborhood Topology with Self-adaptive Differential Evolution. In: Jiao L, Wang L, Gao X, et al. (eds) Advances in Natural Computation. Springer Berlin Heidelberg, pp 976–979

38. Das S, Abraham A, Chakraborty UK, Konar A (2009) Differential Evolution Using a Neighborhood-Based Mutation Operator. IEEE Trans Evol Comput 13:526–553. doi: 10.1109/TEVC.2008.2009457

39. Han M-F, Liao S-H, Chang J-Y, Lin C-T (2013) Dynamic group-based differential evolution using a self-adaptive strategy for global optimization problems. Appl Intell 39:41–56. doi: 10.1007/s10489-012-0393-5

40. Narendra KS, Thathachar MA (1974) Learning automata-a survey. IEEE Trans Syst Man Cybern 323–334. doi: 0.1109/TSMC.1974.5408453

41. Thathachar MA, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Trans Syst Man Cybern Part B Cybern 32:711–722. doi: 10.1109/TSMCB.2002.1049606

42. Rezvanian A, Meybodi M (2010) LACAIS: Learning Automata Based Cooperative Artificial Immune System for Function Optimization. In: Ranka S, Banerjee A, Biswas K, et al. (eds) Contemporary Computing. Springer Berlin Heidelberg, pp 64–75

43. Hashemi AB, Meybodi MR (2011) A note on the learning automata based algorithms for adaptive parameter selection in PSO. Appl Soft Comput 11:689–705. doi: 10.1016/j.asoc.2009.12.030

44. Moghiss V, Meybodi MR, Esnaashari M (2010) An intelligent protocol to channel assignment in wireless sensor networks: Learning automata approach. International Conference on Information Networking and Automation. pp V1–338–V1–343

45. Esnaashari M, Meybodi MR (2011) A cellular learning automata-based deployment strategy for mobile wireless sensor networks. J Parallel Distrib Comput 71:988–1001. doi: 10.1016/j.jpdc.2010.10.015

46. Akbari Torkestani J, Meybodi MR (2011) A cellular learning automata-based algorithm for solving the vertex coloring problem. Expert Syst Appl 38:9237–9247. doi: 10.1016/j.eswa.2011.01.098

47. Barzegar S, Davoudpour M, Meybodi MR, et al. (2011) Formalized learning automata with adaptive fuzzy coloured Petri net; an application specific to managing traffic signals. Sci Iran 18:554–565. doi: 10.1016/j.scient.2011.04.007

48. Akbari Torkestani J (2012) An adaptive learning automata-based ranking function discovery algorithm. J Intell Inf Syst 39:441–459. doi: 10.1007/s10844-012-0197-4

49. Jahanshahi M, Dehghan M, Meybodi M (2013) LAMR: learning automata based multicast routing protocol for multi-channel multi-radio wireless mesh networks. Appl Intell 38:58–77. doi: 10.1007/s10489-012-0357-9

50. Pant M, Thangaraj R, Singh VP (2009) A new differential evolution algorithm for solving global optimization problems. International Conference on Advanced Computer Control. pp 388–392

51. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1:67–82.

52. Suganthan PN, Hansen N, Liang JJ, et al. (2005) Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Nanyang Technol. Univ., Singapore, IIT Kanpur, Kanpur, India, #2005005

53. Das S, Suganthan PN (2010) Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems. Jadavpur University, Nanyang Technological University, Kolkata, India

54. Singh HK, Ray T (2011) Performance of a hybrid EA-DE-memetic algorithm on CEC 2011 real world optimization problems. IEEE Congress on Evolutionary Computation. pp 1322–1326

55. Bandaru S, Tulshyan R, Deb K (2011) Modified SBX and adaptive mutation for real world single objective optimization. IEEE Congress on Evolutionary Computation. pp 1335–1342

56. Elsayed SM, Sarker RA, Essam DL (2011) Differential evolution with multiple strategies for solving CEC2011 real-world numerical optimization problems. IEEE Congress on Evolutionary Computation. pp 1041–1048

57. Reynoso-Meza G, Sanchis J, Blasco X, Herrero JM (2011) Hybrid DE algorithm with adaptive crossover operator for solving real-world numerical optimization problems. IEEE Congress on Evolutionary Computation. pp 1551–1556

Javidan Kazemi Kordestani received the B.Sc. in Computer Engineering (Software Engineering) from Islamic Azad University of Karaj, Iran in 2008 and his M.Sc. in Computer Engineering (Artificial Intelligence) from Islamic Azad University of Qazvin, Iran in 2012. He is

currently working toward the Ph.D. degree from the Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran. His current research interests include evolutionary computation, dynamic optimization problems, real-world applications, and learning systems.

Ali Ahmadi received his B.Sc. in Electrical Engineering from Amirkabir University, Tehran, Iran in 1991 and M.Sc. and Ph.D. in Artificial Intelligence and Soft Computing from Osaka Prefecture University, Japan in 2001 and 2004, respectively. He worked as a researcher in Research Center for Nanodevices and Systems in Hiroshima University, Japan during 2004–2007. He has been with Khajeh-Nasir University of Technology, Tehran, Iran as assistant professor from 2007. His research interests include intelligent models, machine vision, hardware-software co-design, and learning algorithms.

Mohammad Reza Meybodi received the B.Sc. and M.Sc. degrees in Economics from the Shahid Beheshti University, Tehran, Iran, in 1973 and 1977, respectively. He also received the M.Sc. and Ph.D. degrees from the Oklahoma University, USA, in 1980 and 1983, respectively, in Computer Science. Currently he is a Full Professor in Computer Engineering Department, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an Assistant Professor at the Western Michigan University, and from 1985 to 1991 as an Associate Professor at the Ohio University, USA. His research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing and software development.