Appl Intell
DOI 10.1007/s10489-011-0292-1

ORIGINAL PAPER

CLA-DE: a hybrid model based on cellular learning automata for numerical optimization

R. Vafashoar · M.R. Meybodi · A.H. Momeni Azandaryani

Abstract This paper presents a hybrid model named: CLA-DE for global numerical optimization. This model is based on cellular learning automata (CLA) and differential evolution algorithm. The main idea is to learn the most promising regions of the search space using cellular learning automata. Learning automata in the CLA iteratively partition the search dimensions of a problem and learn the most admissible partitions. In order to facilitate incorporation among the CLA cells and improve their impact on each other, differential evolution algorithm is incorporated, by which communication and information exchange among neighboring cells are speeded up. The proposed model is compared with some evolutionary algorithms to demonstrate its effectiveness. Experiments are conducted on a group of benchmark functions which are commonly used in the literature. The results show that the proposed algorithm can achieve near optimal solutions in all cases which are highly competitive with the ones from the compared algorithms.

Keywords Cellular learning automata · Learning automata · Optimization · Differential evolution algorithm

R. Vafashoar (✉) · M.R. Meybodi · A.H. Momeni Azandaryani
Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran
e-mail: vafashoar@aut.ac.ir

M.R. Meybodi
e-mail: mmeybodi@aut.ac.ir

A.H. Momeni Azandaryani
e-mail: a_h_momeni@aut.ac.ir

Published online: 14 April 2011

1 Introduction

Global optimization problems are one of the major challenging tasks in almost every field of science. Analytical methods are not applicable in most cases; therefore, various numerical methods such as evolutionary algorithms [1] and learning automata based methods [2, 3] have been proposed to solve these problems by estimating global optima. Most of these methods suffer from convergence to local optima and slow convergence rate.

Learning automata (LA) follows the general schemes of reinforcement learning (RL) algorithms. RL algorithms are used to enable agents to learn from the experiences gained by their interaction with an environment. For more theoretical information and applications of reinforcement learning see [4–6]. Learning automata are used to model learning systems. They operate in a random unknown environment by selecting and applying actions via a stochastic process. They can learn the optimal action by iteratively acting and receiving stochastic reinforcement signals from the environment. These stochastic signals or responses from the environment exhibit the favorability of the selected actions, and according to them the learning automata modify their action selecting mechanism in favor of the most promising actions [7–10]. Learning automata have been applied to a vast variety of science and engineering applications [10, 11], including various optimization problems [12, 13]. A learning automata approach has been used for a special optimization problem, which can be formulated as a special knapsack problem [13]. In this work a new on-line Learning Automata System is presented for solving the problem. Sastry et al. used a stochastic optimization method based on continuous-action-set learning automata for learning a hyperplane classifier which is robust to noise [12]. In spite of its effectiveness in various domains, learning automata have been criticized for having a slow convergence rate.

Meybodi et al. [14] introduced a new model obtained from the combination of learning automata and cellular automata (CA), which they called cellular learning automata (CLA). It is a cellular automaton that one or more learning automata reside in each of its cells. Therefore, the cells of CLA have learning abilities, and also can interact with each other in a cellular manner. Because of its learning capabilities and cooperation among its elements, this model is considered superior to both CA and LA. Because of local interactions, cellular models can be implemented on distributed and parallel systems. Up to now, CLA has been applied to many problems. Esnaashari and Meybodi proposed a fully distributed clustering algorithm based on cellular learning automata for sensor networks [15]. They also used cellular learning automata for scheduling the active times of the nodes in Wireless Sensor Networks [16]. A dynamic web recommender system based on cellular learning automata is introduced in [17]. More information about the theory and some applications of CLA can be found in [14, 18, 19].

These two paradigms (LA and CLA) have also been used to tackle global numerical optimization problems. Zeng and Liu [3] have used learning automata for continuously dividing and sampling the search space. Their algorithm gradually concentrates on favorable regions and leaves out others. Beigy and Meybodi proposed a new learning algorithm for continuous action learning automata [20] and proved its convergence in stationary environments. They also showed the applicability of the introduced algorithm in solving noise corrupted optimization problems. An evolutionary inspired learning method called genetic learning automata was introduced by Howell et al. [2]. In this hybrid method, genetic algorithm and learning automata are combined to compensate the drawbacks of both methods. Genetic learning automata algorithm evolves a population of probability strings for reaching the global optima. At each generation, each probability string gives rise to one bit string child. These children are evaluated using a fitness function, and based on this evaluation, probability strings are adjusted. Then these probability strings are further maturated using standard evolutionary operators. A somehow closer approach termed CLA-EC [21] is developed on the basis of cellular learning automata and evolutionary computing paradigm. Learning automata technique is also used for improving the performance of the PSO algorithm by adaptively tuning its parameters [22].

Differential evolution algorithm (DE) is one of the recent evolutionary methods that has attracted the attention of many researchers. Up to now, after its initial proposal by Storn and Price [23, 24], a great deal of work has been done for its improvement [25–28]. DE is simple, yet very powerful, making it an interesting tool for solving optimization problems in various domains. Some applications of DE can be found in [29]. It is shown [30] that DE has better performance than genetic algorithm (GA) and particle swarm optimization (PSO) over some benchmark functions; still, it suffers from premature convergence and stagnation. In the premature convergence, population converges to some local optima and loses its diversity. In stagnation, population ceases approaching global optima whether or not it has been converged to some local ones.

The work presented in this paper is primarily based on cellular learning automata, and uses differential evolution algorithm to improve its convergence rate and the accuracy of achieved results. CLA constitutes the population of the algorithm and is evolved for reaching the optima. Each cell of the CLA is composed of two parts that are termed: Candidate Solution and Solution Model. The Solution Model represents a model for the desirability of different regions of the problem search space. The Candidate Solution of a cell is the temporary fittest solution obtained by the cell. The Solution Model is a learning system including a set of learning automata in which each automaton is prescribed to one dimension of the problem search space, and learns the promising regions of that dimension. The Solution Model governs the evolution of its related Candidate Solution. On the other hand, Candidate Solutions also guide the learning process of their Solution Models through reinforcement signals provided for Solution Models. Unlike DE or GA approaches in which each entity represents a single point in the search space, each Solution Model represents the entire search space, and, so trapping on local optima is avoided. The Solution Models interact with each other through a set of local rules. In this way, they can impact on the learning process of each other. DE algorithm is used to transfer information between neighboring Candidate Solutions which increases the impact of the Solution Models on each other's learning process.

This paper is organized as follows. In Sect. 2, the principles on which cellular learning automata is based are discussed. Section 3 includes a brief overview of differential evolution algorithm. The CLA-DE algorithm is then described in more detail in Sect. 4. Section 5 contains implementation considerations, simulation results, and comparison with other algorithms to highlight the contributions of the proposed approach. Finally, conclusions and future works are discussed in Sect. 6.

2 Cellular learning automata

Learning automata are adaptive decision making systems that operate in unknown random environments, and adaptively improve their performance via a learning process. The learning process is as follows: at each stage the learning automaton selects one of its actions according to its probability vector and performs it on the environment; the environment evaluates the performance of the selected action to determine its effectiveness and, then, provides a response for

the learning automaton. The learning automaton uses this response as an input to update its internal action probabilities. By repeating this procedure the learning automaton learns to select the optimal action which leads to desirable responses from the environment [7, 8].

The type of the learning automata which is used in this paper belongs to a family of learning automata called variable structure learning automata. This kind of automata can be represented with a six-tuple like $\{\Phi, \alpha, \beta, A, G, P\}$, where Φ is a set of internal states, α a set of outputs, β a set of input actions, A a learning algorithm, $G(\cdot)$: $\Phi \rightarrow \alpha$ a function that maps the current state into the current output, and P a probability vector that determines the selection probability of a state at each stage. In this definition the current output depends only on the current state. In many cases it is enough to use an identity function as G; in this case the terms action and state may be used interchangeably. The core factor in the performance of the learning automata approach is the choice of the learning algorithm. The learning algorithm modifies the action probability distribution vector according to the received environmental responses. One of the commonly used updating methods is the linear reward-penalty algorithm (L_{R-P}). Let a_i be the action selected at cycle n according to the probability vector at cycle n, $p(n)$; in an environment with two outputs, $\beta \in \{0, 1\}$, if the environmental response to this action is favorable ($\beta = 0$), the probability vector is updated according to (1), otherwise (when $\beta = 1$) it is updated using (2).

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

$$p_j(k+1) = \begin{cases} p_j(k)(1-b) & \text{if } i = j \\ \frac{b}{r-1} + p_j(k)(1-b) & \text{if } i \neq j \end{cases} \tag{2}$$

where a and b are called reward and penalty parameters respectively.

Cellular automata are abstract models for systems consisting of a large number of identical simple components [31]. These components which are called cells are arranged in some regular forms like grid and ring and have local interactions with each other. Cellular automaton operates in discrete times called steps. At each time step, each cell will have one state out of a limited number of states; the states of all cells define the state of the whole cellular automaton. The state of a cell at the next time step is determined according to a set of rules and the states of its neighboring cells. Through these rules, cellular automaton evolves from an internal state during the time to produce complicated patterns of behavior. The ability to produce sophisticated patterns from an internal state with only local interactions and a set of simple rules has caused cellular automaton to be a popular tool for modeling complicated systems and natural phenomena. There are some neighborhood models defined in the literature for cellular automaton, the one which is used in this paper is called Von Neumann.

Cellular Learning Automata (CLA) is an abstract model for systems consisting a large number of simple entities with learning capabilities [14]. It is a hybrid model composed of learning automata and cellular automata. It is a cellular automaton in which each cell contains one or more learning automata. The learning automaton (or group of learning automata) residing in a cell determines the cell state. Each cell is evolved during the time based on its experience and the behavior of the cells in its neighborhood. For each cell the neighboring cells constitute the environment of the cell.

Cellular learning automata starts from some initial state (it is the internal state of every cell); at each stage, each automaton selects an action according to its probability vector and performs it. Next, the rule of the cellular automaton determines the reinforcement signals (the responses) to the learning automata residing in its cells, and based on the received signals each automaton updates its internal probability vector. This procedure continues until final results are obtained.

3 Differential evolution algorithm

A differential evolution algorithm, like other evolutionary algorithms, starts with an initial population and evolves it through some generations. At each generation, it uses evolutionary operators: mutation, crossover, and selection to generate a new and probably better population. The population consists of a set of N-dimensional parameter vectors called individuals: Population $= \{X_{1,G}, X_{2,G}, \ldots, X_{NP,G}\}$ and $X_{i,G} = \{X^1_{i,G}, X^2_{i,G}, \ldots, X^N_{i,G}\}$, where NP is the population size and G is the generation number. Each individual is an encoding corresponding to a temporary solution of the problem in hand. At first stage, the population is created randomly according to the uniform distribution from the problem search space for better covering of the entire solution space [23, 24].

In each generation G, a mutation operator creates a mutant vector $V_{i,G}$, for each individual $X_{i,G}$. There are some mutation strategies reported in the literature, three common ones are listed in (3), (4) and (5).

*DE/rand/*1:

$$V_{i,G} = X_{r1,G} + F(X_{r2,G} - X_{r3,G}) \tag{3}$$

*DE/best/*1:

$$V_{i,G} = X_{\text{best},G} + F(X_{r1,G} - X_{r2,G}) \tag{4}$$

*DE/rand-to-best/*1:

$$V_{i,G} = X_{i,G} + F(X_{\text{best},G} - X_{i,G}) + F(X_{r1,G} - X_{r2,G}) \tag{5}$$

where r_1, r_2, and r_3 are three mutually different random integer numbers uniformly taken from the interval $[1, NP]$, F is a constant control parameter which lies in the range $[0, 2]$, and $X_{best,G}$ is the individual with the maximum fitness value at generation G.

Each mutant vector $V_{i,G}$ recombines with its respective parent $X_{i,G}$ through crossover operation to produce its final offspring vector $U_{i,G}$. DE algorithms mostly use binomial or exponential crossover, of which the binomial one is given in (6). In this type of crossover each generated child inherits each of its parameter values from either of the parents.

$$U_{i,G}^{j} = \begin{cases} V_{i,G}^{j} & \text{if } Rand(0,1) < CR \text{ or } j = irand \\ X_{i,G}^{j} & \text{otherwise} \end{cases} \tag{6}$$

CR is a constant parameter controlling the portion of an offspring inherited from its respective mutant vector. *irand* is a random integer generated uniformly from the interval $[1, N]$ for each child to ensure that at least one of its parameter values are taken from the mutant vector. $Rand(0, 1)$ is a uniform random number generator from the interval $[0, 1]$.

Each generated child is evaluated with the fitness function, and based on this evaluation either the child vector $U_{i,G}$ or its parent vector $X_{i,G}$ is selected for the next generation as in (7).

$$X_{i,G+1} = \begin{cases} U_{i,G} & \text{if } f(U_{i,G}) < f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \tag{7}$$

4 The proposed model

This section describes in detail the model proposed to solve global optimization problems defined as:

$$\begin{aligned} &\text{Minimize:} \quad y = f(x), \\ &\text{Subject to:} \quad x_d \in [s_d, e_d], \end{aligned} \tag{8}$$

where $s = [s_1, s_2, \ldots, s_d, \ldots, s_N]$ and $e = [e_1, e_2, \ldots, e_d, \ldots, e_N]$ together define the feasible solution space, $x = [x_1, x_2, \ldots, x_d, \ldots, x_N]^T$ is a variable vector from this space, and $[s_d, e_d]$ denotes the domain of the variable x_d.

Like evolutionary algorithms, in order to reach the final answer, a population of individuals will be evolved through some generations. Each individual is composed of two parts. The first part of an individual is an N dimensional real vector from the search space, and will be referred to as Candidate Solution. A Candidate Solution represents a probable solution to the problem. The second part which will be called Solution Model is a set of N learning automata each one corresponding to one dimension of the Candidate Solution. The Solution Model of an individual governs the evolutionary process of its Candidate Solution. Its objective is to determine the most promising regions of the problem

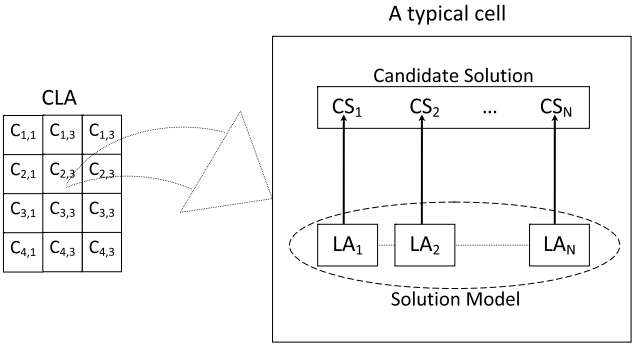

Fig. 1 A scheme of the CLA-DE model entities

search space by continuously dividing the search space and learning the effectiveness of each division. The population of individuals is spatially distributed on a cellular learning automata. Each cell is occupied by only one individual and each individual is exactly assigned to one cell; therefore, the terms individual and cell can be used interchangeably.

As mentioned earlier, each learning automaton pertains to one dimension of the problem search space and can modify only its associated parameter value in the Candidate Solution. For each individual, LA_d will be used to denote the learning automaton associated to its dth dimension. During the evolution process, LA_d will learn an admissible interval where the dth element of the solution will fall. This interval should be as small as possible in order to achieve a high precision final result. Figure 1 shows the relationship between the described entities.

For a typical learning automaton like LA_d, each action corresponds to an interval in its associated domain ($[s_d, e_d]$). The actions of LA_d divide the domain d into some disjoint intervals in such a way that the union of the intervals equals to the whole domain. These intervals are adapted during the execution of the algorithm. At the beginning, the domain is divided into K intervals with equal length that have no intersections. So at the beginning, each learning automaton has K actions, and these actions have identical probabilities. During subsequent stages, each learning automaton learns which one of its actions is more appropriate. After that, this action is substituted by two actions whose related intervals are bisections of the former interval and each one's probability is half of the former one's. By continuing this process, the search in the promising regions is refined and the accuracy of the solution is improved. On the other hand, to avoid an unlimited increase in the number of actions, adjacent intervals with small probabilities are merged together, so are their related actions.

CLA cells (individuals) are adapted during a number of iterations. This adaption comprises both Solution Model and Candidate Solution parts of an individual. In each cell, some copies of the Candidate Solution of the cell are generated. Then, each learning automaton of the cell selects one of its

Fig. 2 A general pseudo code for CLA-DE

CLA-DE algorithm:

(1) Initialize the population.
(2) While stopping condition is not met do:
  (2.1) Synchronously update each cell based on Model Based Evolution.
  (2.2) If generation number is a multiple of 5 do:
    (2.2.1) Synchronously update each cell based on DE Based Evolution.
  (2.3) End.
(3) End.

actions and applies it to one of these copies (this process will be explained later). The altered copies generated by a Solution Model are compared to the neighboring Candidate Solutions, and based on this comparison, a reinforcement signal is generated for each learning automaton of the Solution Model. Learning automata use these responses to update their action probabilities, and based on these new action probabilities, they may adapt their internal structure using action refinement, which will be discussed later. The best generated copy in each cell is used to update the Candidate Solution part of the cell.

Each Solution Model modifies its Candidate Solution, and neighboring Candidate Solutions affect the Solution Model by producing reinforcement signals. In this way, information is communicated among Solution Models. Though, this method of communication is rather slow, so in complicated problems, Solution Models tend to learn by themselves and have little impact on the learning behavior of each other. It is desirable to spread good Solution Models into the population for improving convergence rate and search speed. One way to attain this is to exchange information among neighboring Candidate Solutions since the Solution Model and the Candidate Solution parts of each cell are correlated. A method based on differential evolutionary algorithm will be used for this purpose, and it will be applied at some generations. The general algorithm is shown in Fig. 2, and each of its steps is described in detail in the subsequent sections.

4.1 Initialization

At first stage, the population is initiated. The Candidate Solution part of each individual is randomly generated from the search space using uniform random distribution. Each learning automaton like LA_d will have K actions with equal probabilities $1/K$ at the beginning. These K actions partition the domain of the dimension d into K nonintersecting intervals. For each action $a_{d,j}$, its corresponding interval is defined as in (9).

$$[s_{d,j}, e_{d,j}] = \left[s_d + (j-1)\frac{e_d - s_d}{K}, s_d + j\frac{e_d - s_d}{K} \right] \quad 1 \le j \le K \tag{9}$$

4.2 Model based evolution

At each iteration of the algorithm, each cell evolves based on its Solution Model and the received signals from its neighboring cells. Consider a cell like C, first its learning automata are partitioned into L disjoint groups (some of these groups may be void); each group creates one replica of the cell's Candidate Solution. Then, each learning automaton like LA_d in the group selects one of its actions based on its probability vector. Let the selected action of LA_d be its jth action which is associated with an interval like $[s_{d,j}, e_{d,j}]$. A random number r is uniformly selected from this interval, and the parameter value in the dth dimension of the replica is changed to r. This way, a group of children is generated for each cell. Figure 3(a) schematically shows these ideas.

The Best generated child in each cell is picked out and compared with the Candidate Solution of the cell, and, then, the superior one is selected as the Candidate Solution for the next generation. After updating the Candidate Solution part of a cell, its Solution Model is also updated. Each selected action is evaluated in the environment and its effectiveness is measured. Consider a typical learning automaton like LA_d, its selected action will be rewarded if this action matches more than half of the neighboring Candidate Solutions, or if its corresponding generated child is at least better than half of the neighboring Candidate Solutions. An action like $a_{d,j}$ that defines an interval like $[s_{d,j}, e_{d,j}]$ matches a Candidate Solution like $CS = \langle CS_1, CS_2, \ldots, CS_N \rangle$ if $CS_d \in [s_{d,j}, e_{d,j}]$. If an action is favorable, then its corresponding learning automaton is updated based on (1), and, otherwise, it is updated as in (2). Figure 3(b) depicts these steps.

A pseudo code for this phase of algorithm is shown in Fig. 4. Consider that, after updating the probability vector of each learning automaton, its actions are also refined. The action refinement step is described in the subsequent section.

Action refinement The proposed model slightly differs from other learning automata based algorithms that have fixed actions. In our proposed model, the actions of learning automata will change during the execution of the algorithm. After updating action probabilities of a typical learning automaton like LA_d, if one of its actions like $a_{d,j}$ corresponding to an interval like $[s_{d,j}, e_{d,j}]$ seems to be more appropriate, then it will be replaced by two actions representing the

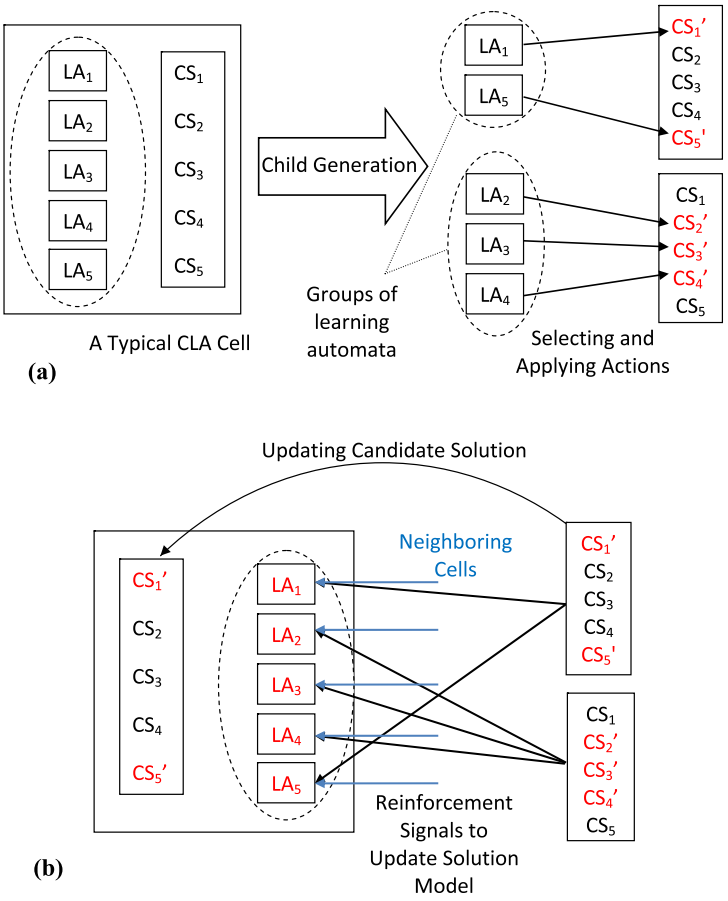

Fig. 3 (**a**) Learning automata of each cell are randomly partitioned into exclusive groups and each group creates one child for the cell. (**b**) Reinforcement signals from generated children and the neighboring cells are used for updating the Solution Model and the best generated child is used for updating the Candidate Solution

Model Based Evolution Phase:

(1) For each cell C with Candidate Solution $CS = \{CS_1, \ldots, CS_N\}$ and Solution Model $M = \{LA_1, \ldots, LA_N\}$, where N is the number of dimensions, do:
  (1.1) Randomly partition M into L mutually disjoint groups: $G = \{G^1, \ldots, G_L\}$.
  (1.2) For Each nonempty Group G^i do:
    (1.2.1) Create a copy $CSC^i = \{CSC^i_1, \ldots, CSC^i_N\}$ of CS ($CSC^i = CS$) for G^i.
    (1.2.2) For each $LA_d \in G^i$ associated with the dth dimension of CS do:
      (1.2.2.1) Select an action from the action set of LA_d according to its probability.
      (1.2.2.2) Let this action correspond to an interval like $[s_{d,j}, e_{d,j}]$.
      (1.2.2.3) Create a uniform random number r from the interval $[s_{d,j}, e_{d,j}]$, and alter the value of CSC^i_d to r.
    (1.2.3) End.
    (1.2.4) Evaluate CSC^i with fitness function.
  (1.3) End.
(2) End.
(3) For each cell C do:
  (3.1) Create a reinforcement signal for each one of its learning automata.
  (3.2) Update the action probabilities of each learning automaton based on its received reinforcement signal.
  (3.3) Refine the actions of each learning automaton in C.
(4) End.

Fig. 4 A pseudo code for the model based evolution phase

two halves $[s_{d,j}, (s_{d,j} + e_{d,j})/2]$ and $[(s_{d,j} + e_{d,j})/2, e_{d,j}]$ of the interval. An action is considered to be appropriate if its probability exceeds a threshold like χ. Consequently, this will cause a finer probing of promising regions. Dividing an action of a learning automaton has no effect on what has been learned by the learning automaton since the probability of selecting a subinterval from any desired interval in the pertaining dimension is the same as before.

Action Refinement Step:

(1) If the probability $p_{d,j}$ of an action $a_{d,j}$ corresponding to an interval $[s_{d,j}, e_{d,j}]$ is greater than χ, split the action into two actions $a^1_{d,j}$ and $a^2_{d,j}$ with the corresponding intervals $[s_{d,j}, (s_{d,j} + e_{d,j})/2]$ and $[(s_{d,j} + e_{d,j})/2, e_{d,j}]$ and equal probabilities $p_{d,j}/2$.
(2) If the probabilities of two adjacent actions $a_{d,j}$ and $a_{d,j+1}$ with corresponding intervals $[s_{d,j}, e_{i,j}]$ and $[e_{d,j}, e_{i,j+1}]$ are both less than ε, merge them into one action $a'_{d,j}$ with corresponding interval $[s_{d,j}, e_{d,j+1}]$ and probability $p_{d,j} + p_{d,j+1}$.

Fig. 5 Pseudo code for action refinement step

Replacing one action with two ones causes an increase in the number of actions, and continuing this process would cause the number of actions to grow tremendously. Considering that, in a learning automaton, the sum of probabilities of all actions always equals one; therefore, an increase in the probability of one action induces a decrease on the probabilities of other actions of the learning automaton. So, there would be some actions with low probabilities that we can merge together making the number of actions almost constant. To attain this, any adjacent actions (whose related intervals are adjacent) with probabilities lower than some threshold like ε are merged together.

As an example, consider a learning automaton with five actions $\{a_{d,1}, a_{d,2}, a_{d,3}, a_{d,4}, a_{d,5}\}$, with respective probabilities $\{0.05, 0.05, 0.51, 0.35, 0.04\}$ and corresponding intervals $\{[1, 2), [2, 4), [4, 4.3), [4.3, 5), [5, 6)\}$. If χ and ε be respectively 0.5 and 0.07, then after the action refinement step the learning automaton will be changed as follows. It will have 5 actions $\{a'_{d,1}, a'_{d,2}, a'_{d,3}, a_{d,4}, a_{d,5}\}$ with corresponding probabilities $\{0.1, 0.255, 0.255, 0.35, 0.04\}$ and respective intervals $\{[1, 4), [4, 4.15), [4.15, 4.3), [4.3, 5), [5, 6)\}$. The pseudo code in Fig. 5 describes the action refinement step.

4.3 DE based evolution

In this phase, each Candidate Solution is evolved based on (3), (4), and (6), though the process slightly differs from that of the standard DE algorithm. In our model, instead of producing just one new child for each Candidate Solution, several children are generated, and then the best one is selected for updating the corresponding Candidate Solution. In order to do this, four Candidate Solutions are selected without replacement from the neighborhood of each cell. The best one is considered as P_{best} and others as P_1, P_2 and P_3. Using different combinations of P_1, P_2, and P_3, a set of mutant vectors is generated based on (3). Based on (4), another set of mutant vectors is also generated with different combinations of P_1, P_2, and P_3 in the deferential term and P_{best} in the base term. The set of generated mutant vectors is then recombined with the Candidate Solution of the cell, based on (6), to generate the children set. A pseudo code describing this phase is shown in Fig. 6.

5 Implementation and numerical results

In order to evaluate the performance of the proposed model, it is applied to a set of 11 benchmark functions that are defined in Table 1. These functions, including unimodal and multimodal functions with correlated and uncorrelated variables are vastly used in the literature for examining the reliability and effectiveness of optimization algorithms [32, 33]. For unimodal functions, the convergence rate is the most interesting factor of a study, and multimodal ones are incorporated to study the capability of algorithms in escaping from local optima and achieving global optima.

5.1 Algorithms for comparison and simulation settings

Algorithms for comparison and their settings To evaluate the effectiveness of the proposed model, it is compared to four other evolutionary algorithms. These algorithms include: DE/rand/bin/1,[1] DERL [25], ODE [27] and PSO with inertia weight (PSO-W) [34].

The population size (NP) is set to 100 for DE/rand/bin/1, DERL, and ODE. This population size is used and recommended in many works for 30 dimensional problems [26, 27]. $NP = 10 \times N$ is used in the original paper for DERL [25], and is recommended in some other works. But almost on all of the used benchmark functions DE algorithms, including DERL, have much better performance with $NP = 100$, so the population size of 100 is adopted for all DE algorithms. There are two 100-dimensional functions in Table 1, larger population sizes are also tested for these functions, but better results achieved when we used $NP = 100$. For ODE and DE/rand/bin/1, F and CR are set to the values 0.5 and 0.9 respectively which are used in many works [26, 27, 30, 36]. RLDE uses $CR = 0.5$ along with a random value for F to generate its trial vectors. The random values for the parameter F are taken uniformly from one of the intervals $[-1, -0.4]$ or $[0.4, 1]$ [25].

The PSO algorithm is implemented as described in [34]. In this implementation, an inertia weight (w) is considered for velocity which is linearly decreased from 0.9 to 0.4 during the execution of the algorithm. The weight parameters $C1$ and $C2$ are kept 2 as used in [34].

[1] Source code for DE is publicly available at: http://www.icsi.berkeley.edu/~storn/code.html.

DE Based Evolution Phase:

(1) For each cell C with Candidate Solution $CS = \{CS_1, \ldots, CS_N\}$ do (where N is the number of dimensions):
  (1.1) Let P_{best}, P_1, P_2, and P_3 be four mutually different Candidate Solutions in the neighborhood of C where P_{best} is the best Candidate Solution neighboring C.
  (1.2) Create 12 children $U = \{U^1, U^2, \ldots, U^{12}\}$ for CS, $U^i = \{U_1^i, U_2^i, \ldots, U_N^i\}$:

$$V_{1:6} = \{P_i + F \times (P_j - P_k) | i, j, k = 1, 2, 3\},$$

$$V_{6:12} = \{P_{best} + F \times (P_i - P_j) | i, j = 1, 2, 3\},$$

Recombine each $V_{1:12}$ with CS based on (6) to form H:

$$U_{i,j} = \begin{cases} V_{i,j} & \text{if } Rand(0,1) < CR \text{ or } j = irand \\ X_{i,j} & \text{otherwise} \end{cases}$$

  (1.3) Select the best generated child U_{best} according to fitness function.
  (1.4) If U_{best} is better than CS replace CS with U_{best}.
(2) End.

Fig. 6 pseudo code for DE based evolution

Table 1 benchmark functions and their properties

Equation	Search domain	Problem dimension	Known global value	Max precision
$f_1 = \sum_{i=1}^{N} (-x_i \sin(\sqrt{\lvert x_i \rvert}))$	$[-500, 500]^N$	30	-12569.49	–
$f_2 = \sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5.12, 5.12]^N$	30	0	1e–14
$f_3 = -20\exp(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2})$ $-\exp(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)) + 20 + \exp(1)$	$[-32, 32]^N$	30	0	1e–14
$f_4 = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N}\cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]^N$	30	0	1e–15
$f_5 = \frac{\pi}{N}\{\sum_{i=1}^{N-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})]$ $+10\sin^2(\pi y_1) + (y_N - 1)^2\} + \sum_{i=1}^{N} u(x_i, 10, 100, 4)$ where: $y_i = 1 + \frac{1}{4}(x_i + 1)$ and: $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & a < x_i \end{cases}$	$[-50, 50]^N$	30	0	1e–30
$f_6 = \sum_{i=1}^{N} u(x_i, 5, 100, 4) + \frac{1}{10}\{\sin^2(3\pi x_1)$ $+\sum_{i=1}^{N-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]$ $+(x_N - 1)^2[1 + \sin^2(2\pi x_N)]\}$ with u defined as in f_5	$[-50, 50]^N$	30	0	1e–30
$f_7 = -\sum_{i=1}^{N}\sin(x_i)\sin^{20}(\frac{i \times x_i^2}{\pi})$	$[-0, \pi]^N$	100	-99.2784	
$f_8 = \frac{1}{N}\sum_{i=1}^{N}(x_i^4 - 16x_i^2 + 5x_i)$	$[-5, 5]^N$	100	-78.33236	
$f_9 = \sum_{j=1}^{N-1}[100(x_j - x_{j+1})^2 + (x_j - 1)^2]$	$[-30, 30]^N$	30	0	
$f_{10} = \sum_{i=1}^{N} x_i^2$	$[-100, 100]^N$	30	0	
$f_{11} = \sum_{i=1}^{N} \lvert x_i \rvert + \prod_{i=1}^{N} \lvert x_i \rvert$	$[-10, 10]^N$	30	0	

Table 2 Mean success rate and Mean number of function evaluations to reach the specified accuracy for successful runs. Success rates are shown inside parentheses

	CLA-DE	DE	DERL	ODE
f_1	**71600** (**1**)	– (0)	184242.50 (1)	– (0)
f_2	**102256.6** (**1**)	– (0)	– (0)	115147 (0.05)
f_3	116616.6 (**1**)	124688.8 (**1**)	143642.5 (**1**)	**69087.5** (**1**)
f_4	90455.5 (**1**)	93902.3 (0.95)	111670 (**1**)	**53958.9** (0.97)
f_5	50496.6 (**1**)	80140.4 (**1**)	104097.5 (**1**)	**41220** (**1**)
f_6	60430 (**1**)	88048.3 (**1**)	109680 (**1**)	**40052.5** (**1**)
f_7	**3570** (**1**)	– (0)	– (0)	233263.6 (0.55)
f_8	**179595** (**1**)	267199 (0.1)	298807 (0.02)	– (0)
f_9	246218 (0.64)	**238117** (**0.9**)	266219 (0.05)	247251 (0.05)
f_{10}	80595 (**1**)	89956.4 (**1**)	101243 (**1**)	**46147.5** (**1**)
f_{11}	113265 (**1**)	135922.3 (**1**)	135055 (**1**)	**106115** (**1**)

Table 3 Success rate and Mean number of function evaluation ranks of the compared methods. Success rate ranks are shown inside parentheses

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	Average rank
CLA-DE	1(1)	1(1)	2(1)	2(1)	2(1)	2(1)	1(1)	1(1)	2(2)	2(1)	2(1)	1.63(1.09)
DE	3(2)	3(3)	3(1)	3(3)	3(1)	3(1)	3(3)	2(2)	1(1)	3(1)	3(1)	2.72(1.72)
DERL	2(1)	3(3)	4(1)	4(1)	4(1)	4(1)	3(3)	3(3)	4(3)	4(1)	4(1)	3.54(1.72)
ODE	3(2)	2(2)	1(1)	1(2)	1(1)	1(1)	2(2)	4(4)	3(3)	1(1)	1(1)	1.81(1.81)

Parameter settings for CLA-DE Many experiments are conducted to study the impact of different parameter settings on the performance of the proposed model, and, accordingly, the following settings are adopted. A 3×4 CLA with $L_{R-\varepsilon P}$ learning algorithm for its learning automata is used; population size is kept small in order to let the CLA learn for more generations and to demonstrate its learning capability. K, the number of initial actions, is set to be 5. At each iteration, M is partitioned into $N/2$ groups ($L = N/2$), so in the average case each group contains two Learning automata. Large values for L would cause learning automata to learn individually, which may cause failure in achieving a desirable result for correlated problems, though small values for L are also inappropriate because with large groups of cooperating learning automata it is difficult to evaluate the performance of each learning automaton. Probability thresholds for dividing and merging actions, χ and ε, are reasonably set to the values: two times the initial action probability and some less than half of the initial action probability respectively e.g.: 0.4 and 0.07. DE based evolution part parameters, i.e. F and CR, are set to the values: 0.8 and 0.9 as recommended in [35]. DE based evolution is executed every 5 generations.

5.2 Experiment 1: comparison of finding the global optimum

In this section the effectiveness of the tested algorithms in reaching global optima is studied, and the success rate of each algorithm on each benchmark is investigated. A run of an algorithm on a benchmark is considered to be successful if the difference between the reached result and the global optima is less than a threshold like Δ. A run of an algorithm is terminated after the success occurrence or after 300000 function evaluations. The average number of function evaluations (FE) for the successful runs is also determined for each algorithm. Success rate and the average number of function evaluations are used as criteria for comparison. For all benchmark functions, except f_7 and f_9, the accuracy of achieved results or Δ is considered 1e–5. None of the compared algorithms have reached this accuracy on the functions f_7 and f_9 (Table 4), so 60 and 0.1 are used as Δ for f_7 and f_9, respectively. The success rate along with the average number of function evaluations for each algorithm on the given benchmarks is shown in Table 2. Based on the results of Table 2, Table 3 shows the success rate and the average number of function evaluation ranks of each method. All of the results reported in this and next sections are obtained based on 40 independent trials.

The first point which is obvious form the achieved results is the robustness of the proposed approach. This issue is apparent from the success rates of CLA-DE which, except in one case, are the highest of all compared ones. Considering problems f_2, f_7 and f_8, CLA-DE is the only algorithm that effectively solved them. Although CLA-DE didn't achieve the best results in terms of the average number of function evaluations in all of the cases, but it has an acceptable one on all of the benchmarks. CLA-DE has a better performance than DE and DERL in terms of approaching speed to the global optima in almost all of the cases. Although it has a

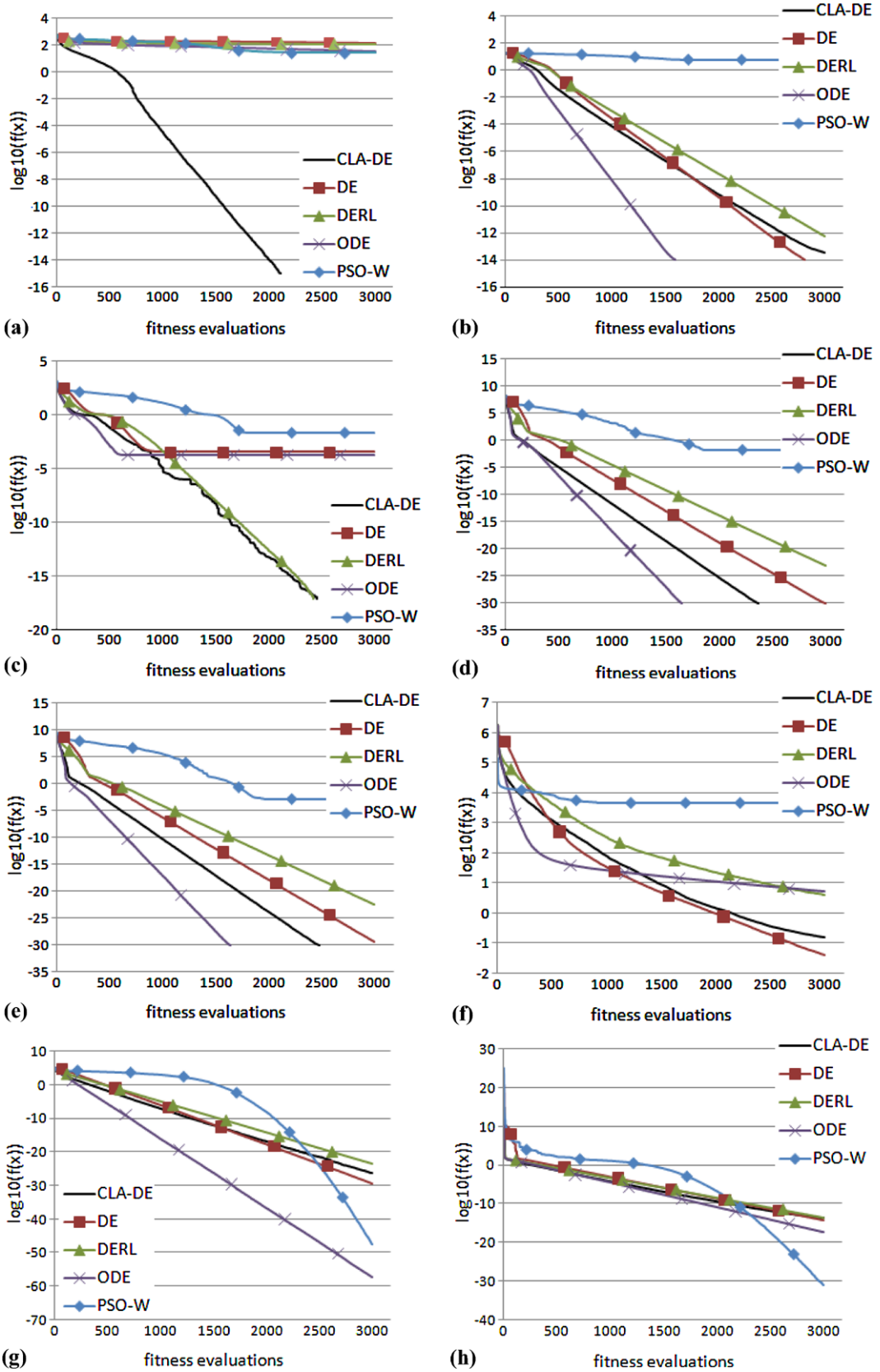

Fig. 7 Fitness curves for functions (**a**) f_2, (**b**) f_3, (**c**) f_4, (**d**) f_5, (**e**) f_6, (**f**) f_9, (**g**) f_{10} and (**h**) f_{11}. The *horizontal axis* is the number of fitness evaluations and the *vertical axis* is the logarithm in base 10 of the mean best function value over all trials

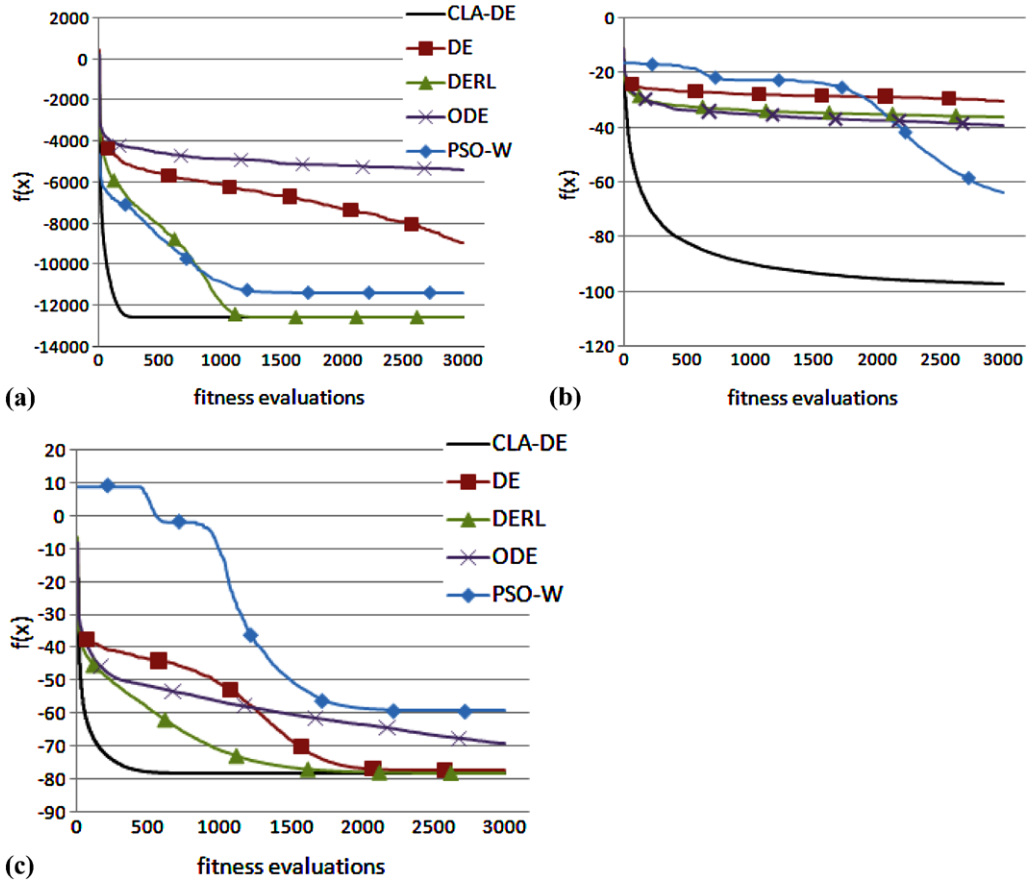

Fig. 8 Fitness curves for functions (**a**) f1, (**b**) f7 and (**c**) f8. The *horizontal axis* is the number of fitness evaluations and the *vertical axis* is the mean best function value over all trials

weaker performance than ODE on some benchmarks, according to this criterion, but CLA-DE has a high success rate in comparison to ODE. Considering Table 3, the success rate rank of ODE is a bit lower than DE and DERL which is the result of its fast convergence rate.

5.3 Experiment 2: comparison of mean best fitness

This experiment is conducted for investigating the mean best fitness that the compared algorithms can achieve after a fixed number of fitness function evaluations. For a fair comparison, all runs of an algorithm are terminated after 300000 function evaluations, or when the error of the achieved result falls below some specified precision (Table 1) for some particular functions. These threshold values are used because variable truncations in implementations limit the precision of results.

Evolution curves (the mean best versus the number of fitness evaluations) for all of the algorithms described in Sect. 5.1 are depicted in Figs. 7 and 8; in addition, Table 4 shows the average final result along with its standard deviation for the different algorithms.

From the obtained results, it is clear that no algorithm performs superiorly better than others, but CLA-DE can achieve a near optimal solution in all cases. From this perspective, it is superior to all the other compared algorithms because any one of them fails to achieve a reasonable result on some of the benchmark functions. For f_1, f_2, f_7, and f_8 CLA-DE obtained the least mean best fitness values. Considering Figs. 7(a) and 8, it also achieved these results with the highest convergence rate. For f_3, f_5, and f_6 it also achieved near optimal results, but with a lower convergence rate than ODE. However, its convergence rate is acceptable and is not far worse than that of ODE for these functions. It also has a good convergence rate for these functions in comparison to algorithms other than ODE (Fig. 7).

CLA-DE presents some compromise between local and global search. Because it considers the whole search space, and each parameter can take its values from the entire space, it is immune to entrapment in local optima. CLA-DE is primarily based on meta-intelligent mutations, which diversify its population, so it does not have the problem of stagnation and premature convergence even in the case of small populations (3×4 CLA in our experiments). Due to this charac-

Table 4 average final result (along with its standard deviation)

	CLA-DE	PSO-W	DE	DERL	ODE
f_1	**−12569.49**	−11367.10	−8940.19	**−12569.49**	−5415.23
	(3.71e–12)	(8.43e+2)	(1.55e+3)	(3.68e–12)	(5.35e+2)
f_2	**8.70e–15**	26.41	133.55	117.11	32.27
	(7.15e–16)	(8.07)	(25.63)	(6.72)	(22.67)
f_3	3.08e–14	6.057	**7.99e–15**	5.84e–13	**7.99e–15**
	(1.72e–14)	(9.49)	(0)	(2.10e–13)	(0)
f_4	**9.53e–16**	2.17e–2	3.69e–4	**8.96e–16**	1.84e–4
	(7.70e–17)	(2.05e–2)	(1.65e–3)	(1.18e–16)	(1.16e–3)
f_5	**9.36e–31**	2.07e–2	1.31e–30	8.69e–24	**9.20e–31**
	(6.04e–32)	(7.21e–2)	(7.83e–31)	(8.46e–24)	(6.98e–32)
f_6	**7.16e–31**	1.64e–3	4.41e–30	3.15e–23	**8.97e–31**
	(7.75e–32)	(4.02e–3)	(4.03e–30)	(2.44e–23)	(8.32e–32)
f_7	**−97.46**	−63.67	−30.21	−35.92	−39.34
	(4.04e–2)	(4.45)	(1.97)	(0.73)	(1.76)
f_8	**−78.33**	−59.24	−77.14	−78.29	−69.13
	(3.97e–10)	(1.54)	(6.39e–1)	(7.67e–2)	(5.75)
f_9	1.56e–1	4.68e+3	**4.00e–2**	4.26	5.14
	(2.09e–1)	(4.47e+3)	(6.65e–2)	(6.84)	(6.02)
f_{10}	5.54e–27	4.78e–48	4.86e–30	4.34e–24	**8.21e–58**
	(1.38e–26)	(1.44e–47)	(4.53e–30)	(2.28e–24)	(1.70e–57)
f_{11}	2.02e–14	**8.88e–32**	7.11e–15	3.36e–14	4.83e–18
	(5.77e–14)	(1.94e–31)	(4.64e–15)	(8.63e–15)	(6.20e–18)

teristic, it shows a slower convergence rate on functions f_{10} and f_{11}.

6 Conclusion

In this paper, a hybrid method based on cellular learning automata and deferential evolution algorithm is presented for solving global optimization problems. The methodology relies on evolving some models of the search space which are named Solution Model. Each Solution Model is composed of a group of learning automata, and each such a group is placed in one cell of a CLA. Using these Solution Models, the search space is dynamically divided into some stochastic areas corresponding to the actions of the learning automata. These learning automata interact with each other through some CLA rules. Also, for better commutation and finer results DE algorithm is incorporated.

CLA-DE was compared to some other global optimization approaches tested on some benchmark functions. Results exhibited the effectiveness of the proposed model in achieving finer results. It was able to find the global optima on almost all of the test functions with acceptable accuracy. In some cases, it showed slower convergence rate than the best algorithms, but the difference between its results and the best ones is admissible. The proposed method totally exhibits good global search capability with an acceptable convergence rate.

As future works, working on fuzzy actions for learning automata to improve searching at boundary areas of action intervals will be interesting. Also, considering other learning algorithms for the learning automata might be beneficial.

References

1. Eiben AE, Smith JE (2003) Introduction to evolutionary computing. Springer, New York
2. Howell MN, Gordon TJ, Brandao FV (2002) Genetic learning automata for function optimization. IEEE Trans Syst Man Cybern 32:804–815
3. Zeng X, Liu Z (2005) A learning automata based algorithm for optimization of continuous complex functions. Inf Sci 174:165–175
4. Hong J, Prabhu VV (2004) Distributed reinforcement learning control for batch sequencing and sizing in just-in-time manufacturing systems. Int J Appl Intell 20:71–87

5. Iglesias A, Martinez P, Aler R, Fernandez F (2009) Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. Int J Appl Intell 31:89–106
6. Wiering MA, Hasselt HP (2008) Ensemble algorithms in reinforcement learning. IEEE Trans Syst Man Cybern, Part B Cybern 38:930–936
7. Narendra KS, Thathachar MAL (1989) Learning automata: an introduction. Prentice Hall, Englewood Cliffs
8. Tathachar MAL, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Trans Syst Man Cybern, Part B Cybern 32:711–722
9. Najim K, Poznyak AS (1994) Learning automata: theory and applications. Pergamon, New York
10. Thathachar MAL, Sastry PS (2003) Networks of learning automata: techniques for online stochastic optimization. Springer, New York
11. Haleem MA, Chandramouli R (2005) Adaptive downlink scheduling and rate selection: a cross-layer design. IEEE J Sel Areas Commun 23:1287–1297
12. Sastry PS, Nagendra GD, Manwani N (2010) A team of continuous-action learning automata for noise-tolerant learning of half-spaces. IEEE Trans Syst Man Cybern, Part B Cybern 40:19–28
13. Granmo OC, Oommen BJ (2010) Optimal sampling for estimation with constrained resources using a learning automaton-based solution for the nonlinear fractional knapsack problem. Int J Appl Intell 33:3–20
14. Meybodi MR, Beigy H, Taherkhani M (2003) Cellular learning automata and its applications. Sharif J Sci Technol 19:54–77
15. Esnaashari M, Meybodi MR (2008) A cellular learning automata based clustering algorithm for wireless sensor networks. Sens Lett 6:723–735
16. Esnaashari M, Meybodi MR (2010) Dynamic point coverage problem in wireless sensor networks: a cellular learning automata approach. J Ad Hoc Sens Wirel Netw 10:193–234
17. Talabeigi M, Forsati R, Meybodi MR (2010) A hybrid web recommender system based on cellular learning automata. In: Proceedings of IEEE international conference on granular computing, San Jose, California, pp 453–458
18. Beigy H, Meybodi MR (2008) Asynchronous cellular learning automata. J Autom 44:1350–1357
19. Beigy H, Meybodi MR (2010) Cellular learning automata with multiple learning automata in each cell and its applications. IEEE Trans Syst Man Cybern, Part B Cybern 40:54–66
20. Beigy H, Meybodi MR (2006) A new continuous action-set learning automata for function optimization. J Frankl Inst 343:27–47
21. Rastegar R, Meybodi MR, Hariri A (2006) A new fine-grained evolutionary algorithm based on cellular learning automata. Int J Hybrid Intell Syst 3:83–98
22. 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
23. Storn R, Price K (1995) Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, Berkeley
24. Storn R, Price KV (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous. Spaces J Glob Optim 11:341–359
25. Kaelo P, Ali MM (2006) A numerical study of some modified differential evolution algorithms. Eur J Oper Res 169:1176–1184
26. Brest J, Greiner S, Boskovic B, Mernik M, Zumer V (2006) Self adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. IEEE Trans Evol Comput 10:646–657
27. Rahnamayan S, Tizhoosh HR, Salama MMA (2008) Opposition-based differential evolution. IEEE Trans Evol Comput 12:64–79
28. Brest J, Maucec MS (2008) Population size reduction for the differential evolution algorithm. Int J Appl Intell 29:228–247
29. Chakraborty UK (2008) Advances in differential evolution. Springer, Heidelberg
30. Vesterstrom J, Thomsen R (2004) A comparative study of differential evolution, particle swarm optimization and evolutionary algorithms on numerical benchmark problems. In: Proceedings of IEEE international congress on evolutionary computation, Piscataway, New Jersey, vol 3, pp 1980–1987
31. Wolfram S (1994) Cellular automata and complexity. Perseus Books Group, Jackson
32. Leung YW, Wang YP (2001) An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Trans Evol Comput 5:41–53
33. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3:82–102
34. Shi Y, Eberhart RC (1999) Empirical study of particle swarm optimization. In: Proceedings of IEEE international congress evolutionary computation, Piscataway, NJ, vol 3, pp 101–106
35. Srorn R (2010) Differential Evolution Homepage. http://www.icsi.berkeley.edu/~storn/code.html. Accessed January 2010
36. Liu J, Lampinen J (2005) A fuzzy adaptive differential evolution algorithm. Soft Comput Fusion Found Methodol Appl 9:448–462

R. Vafashoar received the B.S. degree in Computer Engineering from Urmia University, Urmia, Iran, in 2007, and the M.S. degree in Artificial Intelligence from Amirkabir University of Technology, Tehran, Iran, in 2010. His research interests include learning systems, evolutionary computing and other computational intelligence techniques.

M.R. Meybodi received the B.S. and M.S. degrees in Economics from the Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. 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, 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.

A.H. Momeni Azandaryani was born in Tehran in 1981. He received the B.Sc. degree in Computer Engineering from Ferdowsi University of Mashad, Iran in 2004 and the M.Sc. degree in Artificial Intelligence from Sharif University of Technology, Iran in 2007. He is now a Ph.D. Candidate at AmirKabir University of Technology, Iran and he is affiliated with the Soft Computing Laboratory. His research interests include Learning Automata, Cellular Learning Automata, Evolutionary Algorithms, Fuzzy Logic and Grid Computing. His current research addresses Self-Management of Computational Grids.