A Cooperative Learning Method Based on Cellular Learning Automata and Its Application in Optimization Problems

Milad Mozafari

Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran

Mohammad Ebrahim Shiri[1]

Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran

Hamid Beigy

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

---

Abstract

In this paper, a novel reinforcement learning method inspired by the way humans learn from others is presented. This method is developed based on cellular learning automata featuring a modular design and cooperation techniques. The modular design brings flexibility, reusability and applicability in a wide range of problems to the method. This paper focuses on analyzing sensitivity of the method's parameters and the applicability in optimization problems. Results of the experiments justify that the new method outperforms similar ones because of employing knowledge sharing technique, reasonable exploration logic, and learning rules based on the action trajectory.

Keywords: Cellular automata, Cellular learning automata, Knowledge sharing, Optimization

---

[1]Corresponding author

1. Introduction

Machine learning is about designing techniques and algorithms by which intelligent agents automatically learn the optimal acting for a problem using available data. These techniques are very useful in complex problems whose space are not completely available or predictable. Machine learning techniques fall into three main categories, that are supervised learning [1], unsupervised learning [2], and reinforcement learning [3]. In the case of supervised learning techniques, there is a set of training data for which the solutions are available and the learner tries to infer a function from the training data to map unseen data with high accuracy. In unsupervised learning techniques, there is no training data set and the learner tries to find hidden structures by analyzing different features of the data. Reinforcement learning techniques, that are involved in our method, are inspired by behaviorist psychology, in which agents try to figure out how to act efficiently in an environment by interaction only. With each action an agent performs, the environment sends a reinforcement signal to the agent. Reinforcement signals are the only information the agent receives about its acting, and reinforcement techniques are about learning optimal acting in an environment using the reinforcement signals. Reinforcement learning is applicable to problems in various fields of study such as prediction [4], scheduling [5], wireless networks [6, 7], robotics [8], ensemble learning [9] to mention a few. The variety of applications and the concept of learning from experience, give rise to the study and design of reinforcement learning techniques for complex and large-scale problems.

Progress of technology and emergence of powerful processors provide the ability of parallelism and multi-agent computing which effectively improves the performance and the speed of a learning method. We use cellular learning automata (CLA) [10, 11] which are cellular automata (CA) [12] in which cells are equipped with learning automata (LA) [13, 14], to design a parallel multi-agent method. Our method benefits from advantages of both LAs that use reinforcement learning techniques, and CA as a tool for parallelism and a basis for modeling the fact that human decisions are influenced by the behavior of others he refers to [15]. The magnitude of the influence depends on various factors such as trust, relationship, expertness, and etc. Cellular learning automata have applications in complex systems and simulations [16], optimization [17, 18], classification [19], pattern recognition [20], image processing [21], wireless sensor networks [22], dynamic channel assignment [23, 24] recommender systems [25] and many other types of problems.

In multi-agent scenarios it is very useful to let the agents cooperate with each other and benefit from the knowledge gained by others. Cooperative learning techniques [26] are a class of techniques that are designed to address cooperation between agents in multi-agent environments with the purpose of improving the quality, the accuracy, and the speed of the learning process. Knowledge sharing [27] is one of the cooperative learning techniques in which agents try to learn from each other and share what they learn. We use this technique in our method by letting agents use neighboring agents' knowledge gathered through their individual learning processes.

In this paper, we focus on solving optimization problems and show how the proposed method can be applied to these problems. We also compare our method with Cellular Learning Automata based Evolutionary Computing (CLA-EC) [28] and Recombinative CLA-EC (RCLA-EC) [29], which are the most similar methods to ours. CLA-EC is a combination of CLA and evolutionary algorithms and can be applied to optimization problems. For example, CLA-EC is used for clustering and intrinsic hardware evolution, [30, 31]. Our experiments show that CLA-EC suffers from the lack of exploration and it is susceptible to local optima. The authors in [29] addressed this problem and proposed RCLA-EC, which is a CLA-EC enhanced with a recombination operator that helps the method to be more exploratory in searching the problem space. The results of our experiments indicate that our method has a better performance than both CLA-EC and RCLA-EC, and does not suffer from their disadvantages. Besides, we examine how our method performs on two real-world problems and compare the obtained results with methods other than CLAs.

The rest of this paper is organized as follows: In Section 2, we briefly describe CLAs that are the core of our method. Next, in Section 3 we introduce our method and describe it in full details. In Section 4, we perform different experiments to show the sensitivity of different parameters and compare our method with both CLA-EC and RCLA-EC. Conclusion and future works are highlighted in Section 5.

2. Preliminaries

In this section, we give a brief demonstration of CLAs, which are combinations of LAs and CAs. We begin by introducing LAs followed by description of CAs and CLAs.

2.1. Learning Automata

An LA is a decision-making machine consisting of a set of inputs, a set of states, a set of actions, action probabilities, and a learning algorithm in its primary form. An LA chooses an action in each state based on its action probability vector, and performs it on a responsive random environment. The environment receives the action as an input and produces a reinforcement signal with a fixed unknown probability distribution (as a response). This response shows to what extent the performed action is good or bad and the LA uses this response to update its action probabilities. Action probabilities are crucial to an LA's decisions and form the basis of its learning process. Interactions between the learning automaton and the random environment are depicted in Figure 1.

The random environment is an entity consisting of a set of inputs, a set of outputs (reinforcement signals) and an unknown probability distribution for producing different reinforcement signals. There are three well-known models based on the environment's outputs, which are also the LA's inputs; if the LA's input set is binary (e.g. $\{0, 1\}$), the model is known as the P-model, if the input consists of a set of distinct symbols, it is known as a Q-model, and finally it is an S-model, if the input set is the interval $[0, 1]$ [13]. Depending on the

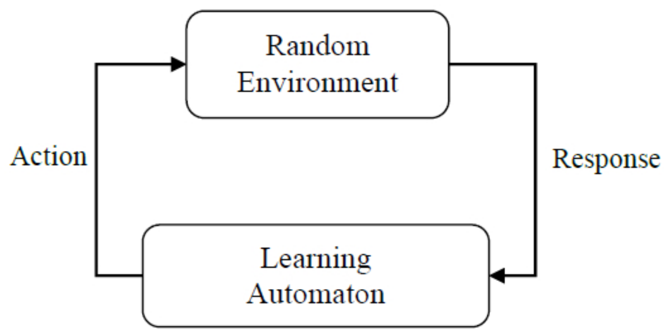

Figure 1: Interaction between an environment and a learning automaton [13]

problem to be solved, a proper model must be designed in order to achieve the best performance.

2.2. Cellular Automata

Cellular automata are dynamic systems consisting of a set of identical automata in which space and time are discrete. These automata are arranged in a cellular grid and interact locally with their neighboring automata which are defined by popular neighborhood, such as the Von Neumann (Figure 2a), or any other kind of neighborhoods [32, 33]. Local interactions are defined by a set of local transition rules indicating the next state of each automaton given the states of its neighboring automata. Transition rules describe the behavior of a CA and designing suitable transition rules for a CA is an important task for which scientists try to develop useful methods. For instance, the authors in [34] employ principal component analysis (PCA) to evaluate transition rules of CAs and retrieve a set of rules which are suitable for different kinds of situations.

Acknowledging that the grid of CAs cannot be infinite in practice, boundary conditions are needed for the cells placed on the edges of the grid. Figure 2b depicts how periodic boundary conditions can be imposed in the case of a two-dimensional grid.

Despite the simple structure of CA, they are able to model non-linear dynamics in natural systems, which are very hard to model by other mathematical

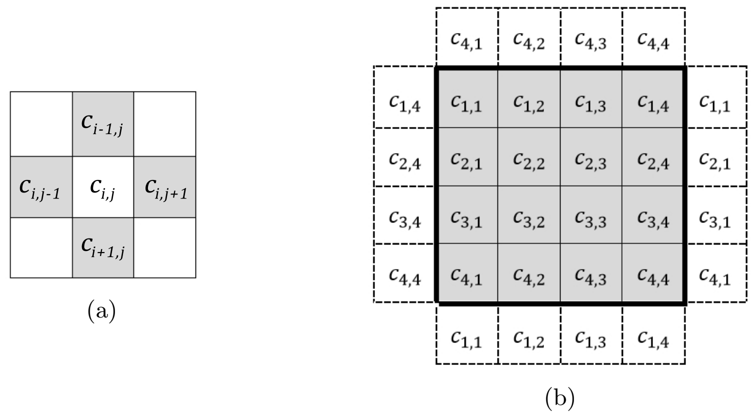

Figure 2: (a) Von Neumann neighborhood for cell $c_{i,j}$ and (b) periodic boundary condition for a grid of size 4×4.

tools like differential equations [35, 36, 37, 38]. Here, we employ CAs to facilitate the modeling of knowledge sharing between agents. For further readings please see [12].

2.3. Cellular Learning Automata

A CLA is a CA in which cells of the grid are equipped with LAs. In these automata, there are local learning rules in addition to the evolution rules and neighbors affect the learning process of the individuals. Simply put, CLAs can learn based on the learning capabilities of LAs and evolve based on local rules of CAs. A d-dimensional CLA is a quintuple $CLA = (Z^d, \varphi, A, N, F)$ where

- Z^d is a d-dimensional grid of cells;
- φ is a finite set of states that each cell can possess;
- A is a set of LAs, each of which is assigned to a specific cell;
- $N = X_1, , X_m$ is a finite subset of Z^d that is called the neighborhood vector;
- $F : \varphi^m \rightarrow \beta$ is the local rule of the CLA where β is a set of valid reinforcement signals that can be applied to LA.

CLAs can be updated either synchronously or asynchronously [39]. In synchronous scheme, all cells are synchronized with a global clock and executed at the same time, while in asynchronous scheme each cell is evolved based on its own clock. Each of these updating schemes can change the behavior of the underlying CA completely [40, 41]. In [10], a mathematical methodology to study the behavior of the synchronous CLA is given and its convergence properties have been investigated.

2.4. Cellular Learning Automata with Multiple Learning Automata in each Cell

In some problems, there is a need to make a decision about multiple (decision) variables, for example, channel assignment in cellular mobile networks for which several decision variables must be tuned. In these situations it is very helpful to extend each cell of the CLA to include a set of LAs (where the cardinality equals to the number of decision variables) instead of one [42], and assign different decision variables to different LAs in each cell.

3. Proposed Method

In this section, we describe our proposed learning method in detail. One of the main goals of our method is to create a modular, flexible and reusable method that fits in many machine learning scenarios by appropriately altering its modules. We begin by introducing the individual modules, then we give a detailed demonstration of their interactions, which reveals how agents learn acting optimally through interaction with their environments.

3.1. Agent and Environment

Agent and environment are the only two distinct modules in our method that have no dependencies but inputs and outputs (the way they interact with each other). In our method, we employ a synchronous CLA whose cells are conceived as agents that are arranged in a 2-dimensional grid. An action of an agent is in the form of a finite-length bit string. To generate such an action, we apply the idea of using multiple LAs in each cell of the CLA [42]. Here, agents include a finite set of LAs that have only two actions; 0 and 1, and the action of each agent is generated by concatenating the actions of the associated LAs. Therefore, the action set for each agent, is the set of all combinations of zeros and ones in a finite-length bit string. Agents store two kinds of actions and their corresponding environment's response throughout their life-time: (1) the latest action they have applied to the environment and (2) the best action they have generated from the beginning of their evolution. They use these actions through their learning process which is discussed in the following subsection.

The goal of these agents is to learn the optimal acting in a random environment by using their individual experience and the knowledge gathered from their neighboring agents. We use a set of random environments which are responsible for evaluating the agents' actions and producing a reinforcement signal. These random environments are identical and each of them is associated to an individual agent.

An environment is the module specifying the problem that must be solved by the agent. Agents know nothing about the problem encapsulated in the environment, but the result of the action they perform on it. Iteratively, agents generate an action and apply it to the environments, then the environments respond with a reinforcement signal and agents use these signals as a guidance to learn how to act better in their future trials.

The only dependency between an agent and its environment, is the length of the action bit string needed to encode solutions of the problem encapsulated in the environment. This low degree of dependency makes our method reusable for solving different kinds of problems without changing the way agents find a solution. To solve a new problem with our method, it is sufficient to define the length of the agent's action and design a proper environment for the new problem. In Section 4, we illustrate how our method can be used to solve optimization problems.

Table 1: List of notations used in this paper

Notation	Definition
d	Size of each dimension of the CLA grid.
$A_{i,j}$	The agent located in the i^{th} row and the j^{th} column of the grid.
$N_{i,j}$	The set of neighboring agents of $A_{i,j}$.
$\mathcal{M}_{i,j}$	The set of LAs associated to $A_{i,j}$.
n	The cardinality of $M_{i,j}$ for all $i, j \in \{1, 2, ..., d\}$.
$m_t^{i,j}$	The t^{th} LA associated to $A_{i,j}$.
$M_{i,j}(k)$	The action probability matrix of $A_{i,j}$ at the k^{th} iteration.
$P_t^{i,j}(k)$	The t^{th} row of $M_{i,j}$ which is the action probability vector of $m_t^{i,j}$.
$p_{t,v}^{i,j}(k)$	The probability that $m_t^{i,j}$ chooses action $v \in \{0, 1\}$ at the k^{th} iteration.
$B_{i,j}(k)$	The action (bit string) generated by $A_{i,j}$ at the k^{th} iteration.
$b_t^{i,j}(k)$	The t^{th} bit of $B_{i,j}(k)$.
$w_{i,j}(k)$	The reinforcement signal produced by the random environment associated to $A_{i,j}$ at the k^{th} iteration as a response to the performed action.
$ec_{i,j}$	The ϵ-escape bit-counter for the agent $A_{i,j}$.

3.2. Acting and Learning

Here, we give a precise and detailed description of the agents' acting and learning procedures in which several notations are employed. Definitions of the notations are summarized in Table 1.

We design an iterative step-wise procedure which involves three phases: decide, act and learn. Each agent generates a new action in the first phase, then applies it to its own environment in the second phase and finally updates its action probability matrix using a learning algorithm provided with the environment's reinforcement signal as an input, in the third phase. Agents perform this procedure iteratively until a stop criterion is met. The remainder of this subsection describes each phase in details.

When an agent enters the decision phase, each of its LAs chooses an action for its corresponding bit in the agent's action bit string, using its associated action probability vector in the action probability matrix. More precisely, when $A_{i,j}$ enters the decision phase at the k^{th} iteration, each LA $m_t^{i,j}$ decides on choosing

1 or 0 for $b_t^{i,j}(k)$, based on its action probability vector $P_t^{i,j}(k)$. Therefore, the automaton $m_t^{i,j}$ assigns the value of 0 with probability $p_{t,0}^{i,j}(k)$ and the value of 1 with the probability $p_{t,1}^{i,j}(k)$ to $b_t^{i,j}(k)$.

In the acting phase, agents perform their generated action (from the previous phase) on their own environment. Then, environments generate reinforcement signals and pass them back to their corresponding agent, which inform the agent about the consequences of its action. As the value of the reinforcement signal gets lower (higher), the positive (negative) consequences of the performed action gets more. In the next phase, agents use these signals to update their probability matrix in order to find the action with the lowest possible reinforcement signal.

As mentioned before, agents generate their actions based on their own action probability matrix. In the learning phase, LAs of each agent update the elements of their corresponding action probability vectors using a learning procedure. In our method, there are two types of learning procedures; individual learning and cooperative learning procedures. In the case of the former, each agent tries to learn based on its own previous actions and the reinforcement signals from the environment (learning from experience), while agents try to learn from the knowledge shared by their neighboring agents in the case of the latter. We control the impact of cooperative learning procedure by the parameter λ, which denotes the probability that the cooperative learning is invoked after each individual learning procedure. So high λ-values result in a more cooperative method.

In the individual learning procedure, the action probability matrix of each agent is updated based on the learning rules. For each action $B_{i,j}(k)$ performed by agent $A_{i,j}$ at the k^{th} iteration and its corresponding reinforcement signal $w_{i,j}(k)$, the action probability matrix $M_{i,j}(k)$ is updated using the following rules:

- If $w_{i,j}(k) < w_{i,j}(k-1)$ then for all $t \in \{1, 2, ..., n\}$ and $v \in \{0, 1\}$:

$$\begin{aligned} &p_{t,v}^{i,j}(k+1) = \\ &p_{t,v}^{i,j}(k) + (2v-1)\left(b_t^{i,j}(k) - b_t^{i,j}(k-1)\right)\left(\eta Q\left(1 - p_{t,b_t^{i,j}(k)}^{i,j}(k)\right)\right) \end{aligned} \tag{1}$$

where

$$Q = \frac{w_{i,j}(k-1) - w_{i,j}(k)}{w_{i,j}(k-1)} = 1 - \frac{w_{i,j}(k)}{w_{i,j}(k-1)}. \tag{2}$$

- If $w_{i,j}(k) > w_{i,j}(k-1)$ then for all $t \in \{1, 2, ..., n\}$ and $v \in \{0, 1\}$:

$$\begin{aligned} &p_{t,v}^{i,j}(k+1) = \\ &p_{t,v}^{i,j}(k) + (1-2v)\left(b_t^{i,j}(k) - b_t^{i,j}(k-1)\right)\left(\eta Q\left(1 - p_{t,b_t^{i,j}(k)}^{i,j}(k)\right)\right) \end{aligned} \tag{3}$$

where

$$Q' = \frac{w_{i,j}(k) - w_{i,j}(k-1)}{w_{i,j}(k)} = 1 - \frac{w_{i,j}(k-1)}{w_{i,j}(k)}. \tag{4}$$

These learning rules are designed in a way that whenever an agent receives a response for the new action, it compares it with the previous signal and sends the signal to a subset of its LAs who choose a different action as compared to the previously selected one, after which, each LA updates its probability vector toward the action that results in the lower reinforcement signal (better action).

Let $\mathcal{C}$ denote the set of indices of the LAs of an agent whose corresponding bits in the newly generated action bit string take different values in comparison with the values in the previous action bit string. When new changes in action bit string results in an improvement, i.e. $w_{i,j}(k) < w_{i,j}(k-1)$, the probability vectors of the LAs in $\mathcal{C}$ must be updated in a way that the generation of an action similar to the new one becomes more probable. Hence, for each bit $b_t^{i,j}(k)$ where $t \in \mathcal{C}$, if its previous value is 0 and its current value is 1, the automaton $m_t^{i,j}$ decreases the probability of choosing action 0 and increases the probability of choosing action 1, and conversely if $b_t^{i,j}(k)$ is 1 (Equation (1)). The coefficient Q controls the magnitude by which the probabilities change.

According to Equation (2), the more the new action improves the reinforcement signal (smaller $w_{i,j}(k)$), the more Q approaches 1, which causes bigger changes in probabilities.

The learning rules for the case of $w_{i,j}(k) > w_{i,j}(k-1)$ (i.e., changes in the newly generated action are bad) are similar, but LAs now change the probabilities so that the generation of an action similar to the previous one becomes more probable (Equation (3)).

In some problems where the difference between two reinforcement signals can be high, the value of Q or Q' may become large and cause rapid convergence toward local optima. To address this problem, we introduced a smoothening parameter $\eta \in [0, 1]$ in the learning equations controlling the rate of convergence.

In a cooperative learning procedure, all the agents share their best action with their neighbors. Each agent compares its latest action value, i.e. $w_{i,j}(k)$, to the best action values of its neighbors and tries to learn the best action among neighbors using the same learning rules in the individual learning procedure by choosing the best neighbor's action as its own new action. More precisely, each agent imitates the best action among its neighbors instead of generating a new action based on its own knowledge (probability vectors).

In our method, there is a chance of being trapped in local optima in the case of complex optimization problems involving many local optima. Therefore, we designed a procedure called ϵ-escape to increase the chance of escaping from local optima when solving complex optimization problems. This procedure is executed with probability ϵ after each action generation. In this procedure, agent $A_{i,j}$ chooses $ec_{i,j}$ bits of its generated action bit string randomly and inverts not only the bits themselves but also the probabilities in their corresponding probability vectors. More precisely, assume that the probability of ϵ holds at the iteration k of the procedure. If we denote the set of indices of the selected bits of the action bit string $B_{ij}(k)$ by

$$\mathcal{S}_{i,j}(k) = \{s_1, s_2, ..., s_{ec_{i,j}}\}, \tag{5}$$

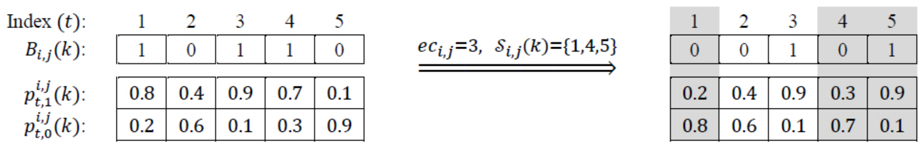

Figure 3: An example of running the escape procedure when $ec_{i,j} = 3$ and selected bit indices are 1,4,5

then, for each bit $b^{i,j}_r(k)$ where $r \in \mathcal{S}_{i,j}(k)$, the agent replaces the $b^{i,j}_r(k)$ with $1-b^{i,j}_r(k)$ and swaps the values of $p^{i,j}_{r,1}(k)$ and $p^{i,j}_{r,0}(k)$ (Figure 3). For each agent $A_{i,j}$, $ec_{i,j}$ is initialized with 1 and is increased each time the ϵ-escape procedure is called. This increment is cyclic, which means that whenever the value of $ec_{i,j}$ reaches n, the increment operator resets it to 1.

Algorithm 1 illustrates the pseudo code of the acting and learning procedures in our method.

4. Experimental Results

In this section, we are going to answer the following questions through performing different experiments:

1. How do the parameters λ (cooperation), ϵ (exploration), and η (convergence speed) influence the agents' learning behavior?
2. How well does the proposed method overcome local optima in comparison with CLA-EC and RCLA-EC?
3. How does our method succeed in solving hard optimization problems in comparison with CLA-EC and RCLA-EC?
4. How does our method's performance compare with other methods that are not based on CLAs?

4.1. Parameter Sensitivity

To address the first question, it is of vital importance to investigate the sensitivity of parameters individually. Therefore, we need a problem with simple space, in which any changes in the agents' learning behavior can be clearly

Algorithm 1: Acting and learning in the proposed method

inputs: $\mathcal{A}$ (set of all agents), $\mathcal{E}$ (set of all environments), d (grid size), n (length of agent's action), λ (cooperation probability), ϵ (ϵ-escape probability), η (smoothening constant)

Set k to 1.
foreach *agent* $A_{i,j}$ *in* $\mathcal{A}$ **do**
    Assign neighboring agents.
    Set all action probabilities to 0.5.
    Generate a random action bit string.
    Perform the action on the environment.
    Store the generated action as the best action.
    Set $ec_{i,j}$ to 1.
while *the stop criteria is not met* **do**
    foreach *agent* $A_{i,j}$ *in* $\mathcal{A}$ **do**
        Initialize $B_{i,j}(k)$ with an empty bit string.
        foreach *automaton* $m_t^{i,j}$ *in* $\mathcal{M}_{i,j}$ **do**
            Choose an action.
            Concatenate $B_{i,j}(k)$ with the chosen action.
        Perform ϵ-escape with probability of ϵ on $B_{i,j}(k)$.
        Perform $B_{i,j}(k)$ on the environment.
        Receive the environment's response $w_{i,j}(k)$.
        Update action probability matrix $M_{i,j}(k)$ by the learning rules.
        if $w_{i,j}(k) < w_{i,j}(k-1)$ **then**
            Update the best action with $B_{i,j}(k)$.
    Increase k by 1.
    if *the probability* λ *holds* **then**
        foreach *agent* $A_{i,j}$ *in* $\mathcal{A}$ **do**
            Find the best action among the neighbors.
            Imitate the best action as a new action.
            Receive the environment's response $w_{i,j}(k)$.
            Update action probability matrix $M_{i,j}(k)$ by the learning rules.
            Replace the best action with the current action.
        Increase k by 1.

illustrated. To show how changes in the design parameters affect the learning behavior of agents, we used the OneMax problem, which involves maximizing the number of ones in a bit string of length n. To solve this problem with our method, we assigned n to the action's length and designed the environment in such a way that it produces reinforcement signals using the following function:

$$w_{i,j}(k) = n - \sum_{t=1}^{n} b_t^{i,j}(k). \tag{6}$$

Despite the simplicity of the OneMax problem and the fact that the optimum solution is known a priori, it is a well-suited problem to illustrate the impact of different parameters on the agents' learning behavior.

4.1.1. Impact of λ

To show how cooperation affects the learning performance, we ran our method to solve 10 different instances of the OneMax problem for $n = 10, 20, ..., 100$ with a 10×10 grid (100 agents). For each of the problem instances, we examined the performance of our method with six different λ-values. Besides, we set $\epsilon = 0$ and $\eta = 1$ to avoid the effects of the ϵ-escape procedure and the smoothening coefficient, respectively. Each of the experiments was repeated 100 times and the average number of decisions (action generations) needed to reach the optimum bit string was considered as the performance measure. Results of the experiments show that higher values for λ result in higher learning performance (Figure 4).

As depicted in figure 4, performance of the non-cooperative method ($\lambda = 0$) dramatically decreases for large instances, while the performance of the cooperative methods ($\lambda > 0$) decreases only slightly as the number of bits increases.

4.1.2. Impact of ϵ

Here, we performed experiments for different values of ϵ from 0 (no exploration) to 0.1. We also used only one agent to omit the impact of cooperation and set $\eta = 1$ to avoid effects of smoothening on these experiments. All of these experiments were performed 100 times on different instances of the OneMax

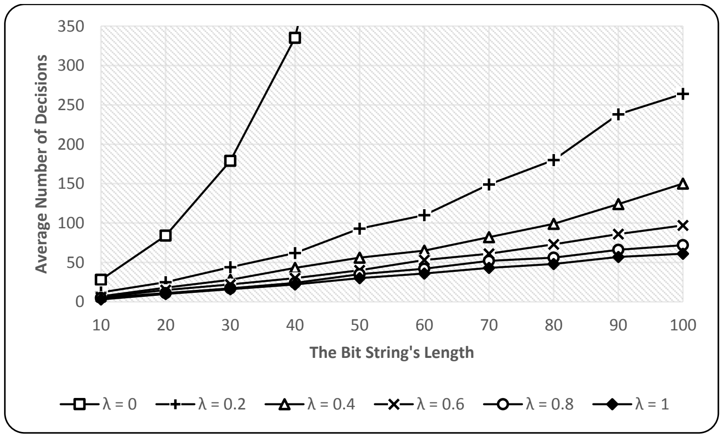

Figure 4: Impact of cooperation (λ) on agents' learning performance in solving the OneMax problem.

problem with $n \in \{10, 20, ..., 70\}$ and the agent was allowed to perform at most 5×10^6 actions. The logarithm of the average number of distinct actions that were examined by the agent to reach the goal, was considered as a measure to assess the effect of the parameter ϵ.

As the value of ϵ increases, the agent explores more distinct actions from its action set to find the optimal action (Figure 5). Exploration helps the agent to explore more actions and escape from local optima, but it is of great importance to assign a tailored value to ϵ according to the problem at stake. An unreasonable high value for ϵ, distracts the agent and does not allow it to exploit the problem space properly.

4.1.3. Impact of η

To show the effects of the smoothening parameter on agents' behavior, we performed experiments for different values of η from 1 (no smoothening) decreasing to 0.2. We also used only one agent to remove the impact of cooperation and set $\epsilon = 0$ to avoid effects of exploration on these experiments. All of these experiments were performed 100 times on different instances of the OneMax problem with $n \in \{10, 20, ..., 70, 80\}$ and the agent was allowed to perform at most 5×10^6 actions. The logarithm of the average number of actions the agent

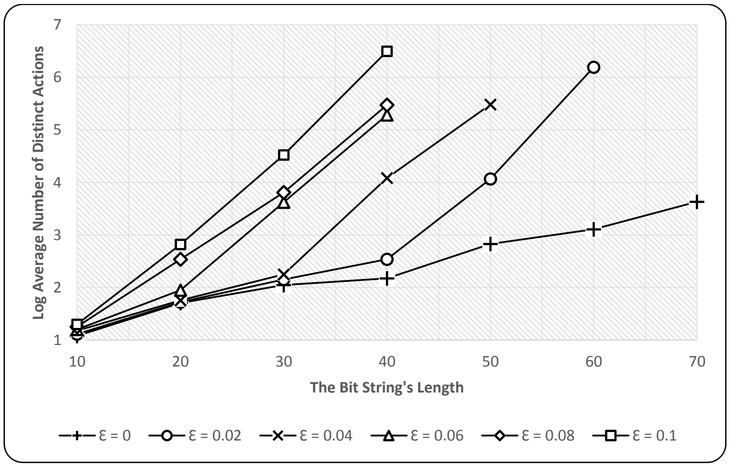

Figure 5: Impact of the ϵ-escape procedure on agents' learning behavior in solving the OneMax problem.

tries to reach the goal was used to assess the effect of this parameter on agent's learning behavior.

As shown in Figure 6, the speed of convergence decreases and the agent needs more tries to learn the optimal action as η decreases. Although it is not desirable to decrease the learning speed in all problems, it is essential in some problems to avoid rapid convergence toward non-optimal actions.

4.1.4. Choosing suitable values for parameters

The OneMax experiments illustrate the influence of each design parameter on the learning behavior of agents. Assigning different values for these parameters directly affects the performance of the method. Therefore, it is necessary to tune them according to the problem at hand and the desired behavior of agents. This can be done by relying on the following guidelines:

(λ) Results show that higher values for λ improve the learning experience of the agents. Yet in real scenarios, where the communication between hardware brings along costs, it is necessary to reduce this value.

(η) Choosing the optimal value for η depends on the maximum and minimum value of the reinforcement signal. Assigning higher (lower) values to

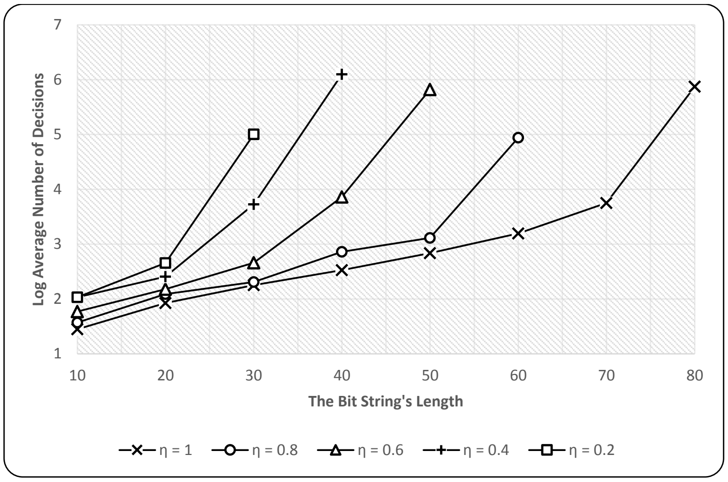

Figure 6: Impact of the smoothening parameter (η) on agent's learning behavior in solving the OneMax problem.

η in case of environments with small (large) range of responses, is better in practice.

(ϵ) Choosing an appropriate value for this parameter is a bit hard. Since ϵ affects the rate of exploration, there is a need for prior knowledge about the problem space to assign a suitable value. Unfortunately, for most of the hard problems, it is not possible to predict the problem space a priori. According to our findings, it is a good practice to begin with lower values and gradually increase it until the desired behavior or performance is reached.

In the following experiments, we chose parameters values in agreement with the above guidelines.

4.2. Escaping from Local Optima

As mentioned before, the methods that are the closest related to ours, namely CLA-EC and RCLA-EC, have problems in exploration and they are very susceptible to become trapped in local optima. We chose the Trap problem, which

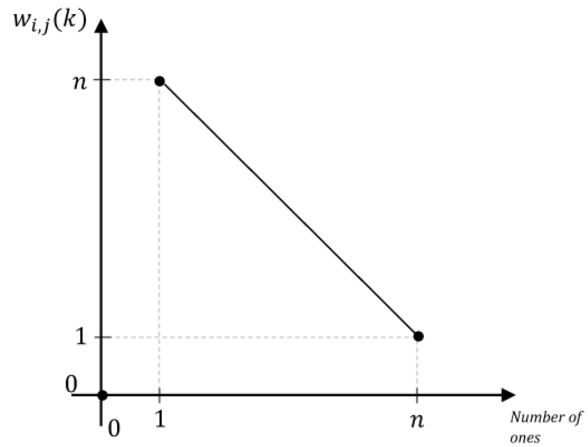

Figure 7: The environment's responses for the Trap problem instance of size n. Best (lowest possible) response is 0 for the string that has zero number of ones.

is similar to the OneMax problem, but we now aim at a binary string in which all bits are zero. In other words, we want to minimize the following function:

$$w_{i,j}(k) = \left(1 - \left(\prod_{t=1}^{n}\left(1 - b_t^{i,j}(k)\right)\right)\right)\left(1 + n - \sum_{t=1}^{n} b_t^{i,j}(k)\right). \tag{7}$$

Equation (7) behaves like a trap function, which makes optimization algorithms end up in the deceptive local optimum (string of all ones) as the best solution. Figure 7 illustrates the environment's response for the Trap problem instance of size n.

In our experiments, CLA-EC always failed to find the global optimum of the Trap problem. Learning rules of a CLA-EC make it act in a greedy manner and it suffers from the lack of exploration. Consequently, CLA-EC does not explore other potential solutions in a complex problem space involving many local optima. The authors of [29] proposed RCLA-EC in which they added a recombination procedure to CLA-ECs, as such designing a method that is both explorative and exploitative. RCLA-EC uses shuffle crossover [43] to recombine solution genome of a cell with its adjacent cells' genomes. This approach helps RCLA-EC to explore the problem space, but it failed in instances with length

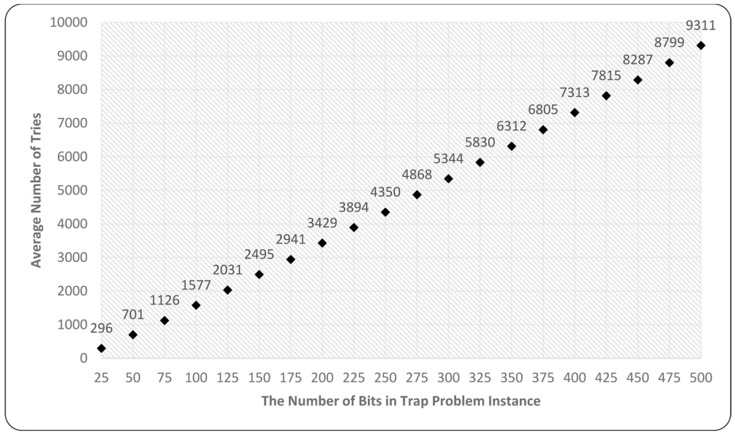

Figure 8: Performance of the proposed method in solving the Trap problem.

larger than 10 in our experiments. We ran our method with lengths from 25 to 500 and averaged the number of tries over agents as a performance measure. We used 100 agents and a Von Neumann neighborhood. We also set $\lambda = 1$ to have a highly cooperative method, $\epsilon = 0.05$ to escape from the trap, and $\eta = 1$ since there is no need to lower the convergence rate in this problem. Results show that our method is able to solve the Trap problem and the average number of decisions made by the agents is almost linear with respect to the length of the bit string (Figure 8).

Table 2 illustrates the experimental results that we collected by running CLA-EC, RCLA-EC and our method for the Trap problem of length 25. As shown in the results, neither CLA-EC nor RCLA-EC succeeds in finding the global optimum in their 100 trials, while our method always finds the global optimum.

We also examined our method to see whether it finds the global optimum or not, using η-values only. As shown in Table 3, relying on η-values solely cannot help the agents in finding the global optimum in this problem. However, η-values do help agents search the problem space locally, while higher ϵ-values increase the chance of escaping local optima.

4.3. Solving Classical Optimization Problems

We designed two experiments in which solving two NP-Hard optimization problems is the goal. These two experiments not only show the flexibility of our method but also demonstrate its superiority over CLA-EC and RCLA-EC. We examined our method for the following NP-Complete optimization problems; bin packing [44] and maximum cut [45]. Because of their complexity class, any other NP problem can be reduced to them in polynomial time. Consequently, by covering these problems we actually cover any other NP problem.

4.3.1. Bin packing problem

We compared our method and both CLA-EC and RCLA-EC in solving the one-dimensional bin packing problem [46]. In each instance of this problem, there are n items $\{1, 2, ..., n\}$ with weights of $\{c_1, ..., c_n\}$. The goal is to pack these items in a minimum number of bins with the same maximum capacity of M.

Since a problem instance with n items can be packed into n bins at the worst case, we assigned each $\lceil \log_2 n \rceil$-bit block of the action bit string to each of the items. Therefore, the bin number in which the item i is packed, is encoded as a binary number located between positions $((i-1)\lceil \log_2 n \rceil) + 1$ to $i\lceil \log_2 n \rceil$ of the action bit string, which results in a action bit string of length $n\lceil \log_2 n \rceil$.

Because of the constraints of the problem instances, it is possible that an agent performs an infeasible action (solution). If the performed action is decoded as a feasible solution, the environment generates a reinforcement signal equal to the number of distinct bins encoded in the performed action bit string. More

Table 2: Performance of the CLA-EC, RCLA-EC, and the proposed method in optimizing the Trap problem of length 25. Success Rate denotes the percentage of trials in which the global optimum found.

Method	Best Fitness	Average Fitness	Success Rate
CLA-EC	1	1	0%
RCLA-EC	1	1	0%
Proposed Method	**0**	**0**	**100%**

precisely if,

$$\mathcal{T} = \{\text{number}\left(\left(\left(i-1\right)\lceil \log_2 n \rceil\right) + 1, i\lceil \log_2 n \rceil\right) \mid i \in \{1, 2, ..., n\}\}, \tag{8}$$

then the reinforcement signal is

$$w_{i,j}(k) = |\mathcal{T}|, \tag{9}$$

where number(x,y) is a function that converts the binary number located between positions x and y of the action bit string to the corresponding decimal number. In the case of an infeasible action, the environment adds $n+1$ units to the signal computed with Equations (8) and (9), which guarantees that a reinforcement signal invoked by an infeasible action is always larger than the ones for feasible actions.

We performed nine different experiments from 10 items to 50 items. The maximum capacity of bins was set to 1 and each item had a weight in the range of $[0, 1]$. In each instance of the problem, the items' weight were assigned in a way that the optimum solution were known in advance so that we could check whether a method finds the optimum or not. Our method found the optimum solution in all settings, whereas CLA-EC and RCLA-EC found only near optimum solutions (Table 4).

4.3.2. Maximum cut problem

We tested our method and both CLA-EC and RCLA-EC on several benchmarks of the Weighted Maximum Cut (max-cut) problem described in [47].

Table 3: Performance of various configurations of the proposed method in optimizing the Trap problem of length 25.

Method	Best Fitness	Average Fitness	Success Rate
Proposed Method ($\epsilon = 0.05, \eta = 1$)	**0**	**0**	**100%**
Proposed Method ($\epsilon = 0, \eta = 0.6$)	1	1	0%
Proposed Method ($\epsilon = 0, \eta = 0.4$)	1	1	0%
Proposed Method ($\epsilon = 0, \eta = 0.2$)	1	1	0%

Table 4: Comparison between proposed method, CLA-EC and RCLA-EC in solving 1-Dimensional Bin Packing problem based on the best and the average of the number of bins they found over all of the trials (best values are boldfaced).

# Items	Best Known	Proposed Method		CLA-EC		RCLA-EC	
		Average	Best	Average	Best	Average	Best
10	3	**3**	**3**	3.71	**3**	3.7	**3**
15	5	**5**	**5**	6.42	**5**	5.3	**5**
20	6	**6**	**6**	8.72	7	8.14	7
25	8	**8**	**8**	11.05	11	10.34	9
30	9	**9**	**9**	13.3	12	12.5	11
35	11	**11**	**11**	18.4	17	17.07	15
40	12	**12.73**	**12**	21.25	20	19.5	18
45	14	**14**	**14**	23.9	23	22.3	21
50	15	**15.64**	**15**	26.3	25	24.2	22

Consider a graph $G = (V, E)$ with vertex set $V = \{v_1, ..., v_n\}$ and edge set $E \subseteq \{\{x, y\} \mid \forall x, y \in V, x \neq y\}$. Let $f_{x,y}$ be the weight associated with edge (x, y) in E. A cut (S, S') is a partitioning of V into two sets being S and $S' = V \setminus S$. The value of a cut (S, S') is given by the following expression:

$$cut(S, S') = \sum_{x \in S, y \in S', y > x} f_{x,y}, \tag{10}$$

where the goal in the max-cut problem is to find the cut in G that yields the maximum cut value.

To solve this problem, each bit of the action is associated with a node of the graph G. If the bit located at the position x has the value 1 (0), its corresponding node v_x in G belongs to the partition set S (S').

Since agents try to minimize the environment's response, it should be computed in such a way that the minimization of the environment's response corresponds to maximization of the value of the cut. For this reason, the environment computes the reinforcement signal of an action using the following equation:

$$w_{i,j}(k) = \sum_{t_1=1}^{n} \sum_{t_2=t_1+1}^{n} f_{t_1,t_2} \left(b_{t_1}^{i,j}(k) + b_{t_2}^{i,j}(k) - 2 b_{t_1}^{i,j}(k) b_{t_2}^{i,j}(k) \right) \tag{11}$$

In all of the chosen benchmarks, there is a weighted graph with 125 vertices

Table 5: Comparison between proposed method, CLA-EC and RCLA-EC in solving max-cut problem based on the best and the average of the cut values they found over all of the trials (best values are boldfaced).

Instance	Best Known	Proposed Method		CLA-EC		RCLA-EC	
		Average	Best	Average	Best	Average	Best
sg3dl051000	110	**107.45**	**110**	85.64	92	82	86
sg3dl052000	112	**106.2**	**112**	84.4	88	84.4	88
sg3dl053000	106	**103.4**	**106**	81.8	90	80.6	86
sg3dl054000	114	**106.2**	**114**	83.8	88	82.8	86
sg3dl055000	112	**106.6**	**112**	84.2	88	83	92
sg3dl056000	110	**105.6**	**110**	86	94	80.8	86
sg3dl057000	112	**106.2**	**112**	82.8	86	83	92
sg3dl058000	108	**103.4**	**108**	83	88	80.8	86
sg3dl059000	110	**105.2**	**110**	84	88	81	86
sg3dl0510000	112	**106.6**	**112**	84.6	90	82.2	86

and 375 edges with weights belonging to the set $\{-1, 0, 1\}$. Table 5 summarizes the performance of the three methods in solving ten different weighted max-cut instances. The proposed method not only found the best-known solution in all of the instances, but also the averages of the answers in different trials are very close to the best-known answers, while CLA-EC and RCLA-EC did not perform well in tackling this problem.

4.4. Solving Real-World Optimization Problems

We implicitly illustrated the performance of our method in comparison with many other methods that are examined on the selected max-cut benchmarks in our previous experiment. Here, we are going to solve two real-world optimization problems and explicitly compare the performance proposed method to other optimization algorithms which participated in the competition of the 2011 Congress on Evolutionary Computation (CEC2011) [48]. We ran and evaluated our method on the problems T01 and T07 under the same evaluation criteria as employed by the other methods considering the rules stated in the problem set.

As shown in Table 6, our method found the best possible solution for the problem T01. Besides, it approached the best solution for the problem T07 and

Table 6: Comparison between proposed method and other methods published in CEC2011 in solving problems T01 and T07 based on the best and the average of the fitness values they found over 25 trials (best values are boldfaced).

	T01		T07	
Method	Best	Average	Best	Average
Proposed Method	**0**	1.6376E-07	6.5822E-01	8.8081E-01
$Adap.DE$ [49]	**0**	3.85	**0.5**	**0.5**
$CDASA$ [50]	3.28E-18	1.01E+01	6.76E-01	9.39E-01
$DE - \Lambda_{Cr}$ [51]	7.21E-15	8.77E-01	6.66E-01	8.85E-01
$DE - RHC$ [52]	5.02E-20	8.91	9.51E-01	1.15
$EA - DE - MA$ [53]	1.17E-11	7.60227	**0.5**	5.8474E-01
$ED - DE$ [54]	**0**	**0**	5.19E-01	1.19
$ENSML\_DE$ [55]	**0**	1.78	1.28	1.42
$GA - MPC$ [56]	**0**	**0**	**0.5**	7.48E-01
Mod_DE_LS [57]	3E-06	2.6E-05	7.2305E-01	8.3277E-01
$mSBX - GA$ [58]	6.7922E-05	4.1976	6.7903E-01	9.8388E-01
RGA [59]	1E-04	9.2907	6.765E-01	9.65E-01
$SAMODE$ [60]	**0**	1.2120	**0.5**	8.17E-01
WI_DE [61]	**0**	3.28	**0.5**	6.56E-01

outperformed seven of the other methods.

It is worth mentioning that our method uses binary strings for decision making. Therefore, to solve the problems with real-valued variables, it needs to encode them as binary strings. As the variables accept a wider range of values with higher precision, much more bits are required for encoding them. For instance, suppose that there are 20 real-valued variables in the interval [0,1]. To have 4-digit precision, we need to assign 10 bits ($2^{10} = 1024$) for each of the variables, which results in bit strings of length 200. This means that agents should decide on 200 variables instead of 20, which leads to a larger search space and a harder task. Still, our method shows good performance on both real-world optimization problems and obtained competitive results.

5. Conclusion and Future Works

In this paper, we have presented a novel learning method based on CLA and cooperative learning techniques. The proposed method has been configured and applied to optimization problems. We have performed several experiments to investigate the impact of design parameters on our method and the agents' learning behavior. Besides, We have performed several experiments to compare our method with CLA-EC and RCLA-EC in solving optimization problems. Our method gives promising results, which is a consequence of three main differences between our method and CLA-EC and RCLA-EC:

(1) **Learning rules based on agents' action trajectory**

Previous methods consider agents' best action as the main parameter in their learning process which makes them behave in a greedy manner and does not allow them to sense the changes in the problem space. Here, we designed the learning rules based on agents' action trajectory.

(2) **Local optima escape procedure**

CLA-EC does not consider exploration and does not perform well in problems containing local optima. Although RCLA-EC uses shuffle crossover as a recombination operator for exploration, it does not have a good performance in complex problems since it blindly shuffles and recombines solutions. We designed an escape procedure by which agents change their action bit string (solution) gradually, and results have shown that this procedure increases the chance of escaping from local optima.

(3) **Cooperation**

Cooperation in previous works is based on a voting method by which agents count their neighbors' votes about the potential solution and decide based on higher votes. Our method uses a knowledge sharing technique by which agents use their best neighbor's action to improve their learning process. We have shown that this way of cooperation increases the method's performance.

We have also evaluated the performance of our method in comparison with a bunch of optimization algorithms in solving two real-world optimization problems. These corroborated the excellence of the proposed method.

Employing trust management systems and expertness in agents' cooperation and applying the method to multi-agent reinforcement learning scenarios are other directions for future researches.

References

[1] S. Kotsiantis, Supervised machine learning: A review of classification techniques, Informatica 31 (2007) 249–268.

[2] S. Marsland, Unsupervised learning, in: J. Fagerberg, D. Mowery, R. Nelson (Eds.), Machine Learning: An Algorithmic Perspective, CRC Press, 2011.

[3] A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

[4] T. Shimokawa, K. Suzuki, T. Misawa, Y. Okano, Predicting investment behavior: An augmented reinforcement learning model, Neurocomputing 72 (2009) 3447–3461.

[5] C. Chen, B. Xia, B.-h. Zhou, L. Xi, A reinforcement learning based approach for a multiple-load carrier scheduling problem, Journal of Intelligent Manufacturing (2013) 1–13.

[6] M. Boushaba, A. Hafid, A. Belbekkouche, M. Gendreau, Reinforcement learning based routing in wireless mesh networks, Wireless networks 19 (2013) 2079–2091.

[7] K.-L. A. Yau, K. H. Kwong, C. Shen, Reinforcement learning models for scheduling in wireless networks, Frontiers of Computer Science 7 (2013) 754–766.

[8] T. Hester, P. Stone, Texplore: real-time sample-efficient reinforcement learning for robots, Machine Learning 90 (2013) 385–429.

[9] I. Partalas, G. Tsoumakas, I. Vlahavas, Pruning an ensemble of classifiers via reinforcement learning, Neurocomputing 72 (2009) 1900–1909.

[10] H. Beigy, M. R. Meybodi, A mathematical framework for cellular learning automata, Advances in Complex Systems 7 (2004) 295–319.

[11] M. Meybodi, H. Beigy, M. Taherkhani, Cellular learning automata, in: Proceedings of 6th Annual International Computer Society of Iran Computer Conference (CSICC2001), Isfahan, Iran, 2001, pp. 153–163.

[12] J. L. Schiff, Cellular automata: a discrete view of the world, volume 45, John Wiley & Sons, 2011.

[13] K. S. Narendra, M. Thathachar, Learning automata-a survey, IEEE Transactions on Systems, Man and Cybernetics (1974) 323–334.

[14] M. Thathachar, P. S. Sastry, Varieties of learning automata: an overview, IEEE Transactions on Systems, Man, and Cybernetics, Part B 32 (2002) 711–722.

[15] P. Mavrodiev, C. J. Tessone, F. Schweitzer, Quantifying the effects of social influence, Nature Publishing Group, Scientific reports 3 (2013).

[16] M. Mozafari, R. Alizadeh, A cellular learning automata model of investment behavior in the stock market, Neurocomputing 122 (2013) 470–479.

[17] J. A. Torkestani, M. R. Meybodi, A cellular learning automata-based algorithm for solving the vertex coloring problem, Expert Systems with Applications 38 (2011) 9237–9247.

[18] R. Vafashoar, M. Meybodi, A. M. Azandaryani, CLA-DE: a hybrid model based on cellular learning automata for numerical optimization, Applied Intelligence 36 (2012) 735–748.

[19] M. Esmaeilpour, V. Naderifar, Z. Shukur, Cellular learning automata approach for data classification, International Journal of Innovative Computing, Information and Control 8 (2012) 8063–8076.

[20] M. Ahangaran, H. Beigy, Cellular learning automata with external input and its applications in pattern recognition, in: Fifth International Conference on Soft Computing, Computing with Words and Perceptions in

System Analysis, Decision and Control (ICSCCW2009), IEEE, 2009, pp. 1–4.

[21] A. A. Abin, M. Fotouhi, S. Kasaei, A new dynamic cellular learning automata-based skin detector, Multimedia Systems 15 (2009) 309–323.

[22] M. Esnaashari, M. R. Meybodi, A cellular learning automata-based deployment strategy for mobile wireless sensor networks, Journal of Parallel and Distributed Computing 71 (2011) 988–1001.

[23] H. Beigy, M. R. Meybodi, A self-organizing channel assignment algorithm: A cellular learning automata approach, in: Intelligent Data Engineering and Automated Learning, Springer, 2003, pp. 119–126.

[24] H. Beigy, M. R. Meybodi, Cellular learning automata based dynamic channel assignment algorithms, International Journal of Computational Intelligence and Applications 8 (2009) 287–314.

[25] M. Talabeigi, R. Forsati, M. R. Meybodi, A hybrid web recommender system based on cellular learning automata, in: 2010 IEEE International Conference on Granular Computing (GrC), IEEE, 2010, pp. 453–458.

[26] L. Panait, S. Luke, Cooperative multi-agent learning: The state of the art, Autonomous Agents and Multi-Agent Systems 11 (2005) 387–434.

[27] R. García-Martínez, D. Borrajo, P. Maceri, P. Britos, Learning by knowledge sharing in autonomous intelligent systems, in: Advances in Artificial Intelligence-IBERAMIA-SBIA 2006, Springer, 2006, pp. 128–137.

[28] R. Rastegar, M. Meybodi, A new evolutionary computing model based on cellular learning automata, in: 2004 IEEE Conference on Cybernetics and Intelligent Systems, volume 1, IEEE, 2004, pp. 433–438.

[29] B. Jafarpour, M. R. Meybodi, Recombinative CLA-EC, in: 19th IEEE International Conference on Tools with Artificial Intelligence, (ICTAI2007), volume 1, IEEE, 2007, pp. 415–422.

[30] R. Rastegar, M. Rahmati, M. Meybodi, A clustering algorithm using cellular learning automata based evolutionary algorithm, in: Adaptive and Natural Computing Algorithms, Springer, 2005, pp. 144–150.

[31] A. Hariri, R. Rastegar, K. Navi, M. S. Zamani, M. R. Meybodi, Cellular learning automata based evolutionary computing (CLA-EC) for intrinsic hardware evolution, in: NASA / DoD Conference on Evolvable Hardware (EH2005), Washington, DC, USA, IEEE, 2005, pp. 294–297.

[32] J. Baran, P. Petrovic, M. Schoenauer, Cellular automata with irregular structure: a compact representation, in: Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW2010), IEEE, 2010, pp. 85–90.

[33] J. M. Baetens, B. De Baets, Cellular automata on irregular tessellations, Dynamical Systems 27 (2012) 411–430.

[34] M. Najafi, H. Beigy, Using PCA to improve evolutionary cellular automata algorithms, in: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, ACM, 2008, pp. 1129–1130.

[35] D. Iudin, Y. D. Sergeyev, M. Hayakawa, Infinity computations in cellular automaton forest-fire model, Communications in Nonlinear Science and Numerical Simulation 20 (2015) 861–870.

[36] A. A. Patel, S. K. Lemieux, R. A. Gatenby, Cellular automaton model of tumor growth, Academic Radiology 5 (1998) 751.

[37] H. Hiyoshi, Y. Tanioka, T. Hamamoto, K. Matsumoto, K. Chiba, Pedestrian movement model based on voronoi cellular automata, Transportation Research Procedia 2 (2014) 336–343.

[38] M. Bezbradica, H. J. Ruskin, M. Crane, Comparative analysis of asynchronous cellular automata in stochastic pharmaceutical modelling, Journal of Computational Science 5 (2014) 834–840.

[39] H. Beigy, M. R. Meybodi, Asynchronous cellular learning automata, Automatica 44 (2008) 1350–1357.

[40] J. Baetens, P. V. der Ween, B. D. Baets, Effect of asynchronous updating on the stability of cellular automata, Chaos, Solitons & Fractals 45 (2012) 383–394.

[41] B. Schnfisch, A. de Roos, Synchronous and asynchronous updating in cellular automata, Biosystems 51 (1999) 123–143.

[42] H. Beigy, M. R. Meybodi, Cellular learning automata with multiple learning automata in each cell and its applications, IEEE Transactions on Systems, Man, and Cybernetics, Part B 40 (2010) 54–65.

[43] D. Simon, Evolutionary optimization algorithms, John Wiley & Sons, 2013.

[44] B. Korte, J. Vygen, Bin-packing, in: Combinatorial Optimization, volume 21 of *Algorithms and Combinatorics 21*, Springer Berlin Heidelberg, 2006, pp. 426–441.

[45] R. M. Karp, Reducibility among combinatorial problems, Springer, 1972.

[46] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, SIAM Journal on Computing 3 (1974) 299–325.

[47] P. Festa, P. M. Pardalos, M. G. Resende, C. C. Ribeiro, Randomized heuristics for the max-cut problem, Optimization methods and software 17 (2002) 1033–1058.

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

[49] M. Asafuddoula, T. Ray, R. Sarker, An adaptive differential evolution algorithm and its performance on real world optimization problems, in:

2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1057–1062.

[50] P. Korosec, J. Silc, The continuous differential ant-stigmergy algorithm applied to real-world optimization problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1327–1334.

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

[52] A. LaTorre, S. Muelas, J.-M. Pena, Benchmarking a hybrid DE-RHC algorithm on real world problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1027–1033.

[53] H. Singh, T. Ray, Performance of a hybrid EA-DE-memetic algorithm on CEC 2011 real world optimization problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1322–1326.

[54] Y. Wang, B. Li, K. Zhang, Estimation of distribution and differential evolution cooperation for real-world numerical optimization problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1315–1321.

[55] R. Mallipeddi, P. Suganthan, Ensemble differential evolution algorithm for CEC2011 problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1557–1564.

[56] S. Elsayed, R. Sarker, D. Essam, GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1034–1040.

[57] A. Mandal, A. Das, P. Mukherjee, S. Das, P. Suganthan, Modified differential evolution with local search algorithm for real world optimization,

in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1565–1572.

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

[59] A. Saha, T. Ray, How does the good old genetic algorithm fare at real world optimization?, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1049–1056.

[60] S. Elsayed, R. Sarker, D. Essam, Differential evolution with multiple strategies for solving CEC2011 real-world numerical optimization problems, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 1041–1048.

[61] U. Haider, S. Das, D. Maity, A. Abraham, P. Dasgupta, Self adaptive cluster based and weed inspired differential evolution algorithm for real world optimization, in: 2011 IEEE Congress on Evolutionary Computation (CEC), 2011, pp. 750–756.