# A Hybrid Method for Optimization (Discrete PSO + CLA)

**B. Jafarpour M. R. Meybodi S. Shiry**

Computer Engineering and Information Technology Department
Amirkabir University of Technology
Tehran Iran
Email: (jafarpour@cic.aut.ac.ir, mmeybodi@aut.ac.ir, shiry@aut.ac.ir)

## Abstract

Particle Swarm Optimization (PSO) is an evolutionary algorithm that is inspired from collective behavior of animals such as fish schooling or bird flocking. A location and a velocity are assigned to every particle in swarm. Velocities of particles are adjusted according to best solution that itself and other members of swarm have found so far. One of the drawbacks of this model is premature convergence and trapping in local optima. In this paper we propose a solution to this problem in discrete PSO using Learning Automata and introduce a Cellular Learning Automata (CLA) based discrete PSO. Experimental results on five optimization problems show the superiority of the proposed algorithm.

**Keywords**: Particle Swarm Optimization, Learning Automata, Cellular Learning Automata, Optimization

## 1. Introduction

Particle Swarm Optimization (PSO) method was first proposed by Kennedy and Eberhart [1] in 1995. According to PSO, the behavior of each particle is affected by the best solution that is found by that particle and the best global particle to help it fly through a search space. Moreover, a particle can learn from its past experience to adjust its flying speed and direction. Therefore, by observing the behavior of the flock and memorizing their flying histories all particles in swarm can quickly converge to near optimal geographical with a well preserved population density distribution [20]. PSO is considered as an evolutionary computation approach in that it possesses many characteristics that are used by evolutionary algorithms such as, initializing with a population of random solutions, searching for optima by updating generations, the adjustment of individuals and evaluating them by a fitness function. However unlike evolutionary algorithms, the updates of particles are not accomplished by crossover or mutation[8]. The particle swarm algorithms reported in the literatures are classified into two groups: discrete PSO and continuous PSO [1][2][3]. In continuous PSO the particles operate in continuous search space, where the trajectories are defined as changes in position on some number of dimensions. But in discrete PSO the particles operates on discrete search space, and the trajectories are defined as changes in the probability that a coordinate will take on a value from feasible discrete values. One of the drawbacks of standard PSO model is premature convergence and trapping in local optima. Recently three solutions based on learning automata for solving this problem have been proposed [3][11][12].

In [3] a discrete version of PSO based on learning automata is proposed. In the proposed algorithm, learning automata are used by the particles to model the dynamics of the group to which the particles belong. The set of leaning automata associated to a particle, by observing the behavior of the group help the particle in searching for optimal geographical with a well preserved population density distribution. In the proposed algorithm the set of learning automata assigned to a particle may be viewed as the brain of the particle determining its position from its own and other particles past experience. To show the effectiveness of the proposed algorithm the authors have tested the algorithm on several function optimization problems. The numerical results have shown that the performance of the proposed algorithm is better than Kennedy's discrete approach for most of the test problems.

In [11] a continuous version of PSO based on learning automata has been proposed. In this model a learning automaton is used to balance exploration and exploitation made by the PSO algorithm. In this paper a new PSO model called PSO-LA is proposed in which a learning automaton takes the role of configuring the behavior of particles and creating a balance between the process of global and local search. The results of experiments conducted by the authors on some standard problems show that the proposed algorithm produces better results than the standard PSO.

In [12] another version of PSO which is a combination of Cellular Learning Automata (CLA)[14] and continuous PSO is proposed. In this model, each cell of CLA contains a population of particles (sub-swarm) and a learning automaton. Learning automaton of each cell is responsible for configuring the behavior of the cell's particles. That is, in each iteration, the learning automaton in each cell of CLA, determines that if the particles should continue their current way or should follow the best experiences. In fact, a sub-swarm in a cell according to the behavior of the sub-swarms residing in the neighboring cells decide to do exploration or exploitation .Indeed, the act of establishing the balance between global and local search which is done by parameter inertia weight (IW) in [5], will be done by learning automata in CLA-PSO. Using learning automata for this purpose makes it possible to use existing knowledge and the feedbacks in IW changes process. Parameter IW is a part of particle's current velocity which is used in calculating its velocity in the next iteration. Using this parameter makes it possible to achieve a better balance between global and local search. In some applications IW has a constant value but in most cases it is decreased linearly by time [19]. Decreasing the IW linearly is a simple method that is unable to use the existing knowledge about the problem domain to guide the process of adjustment of IW.

In [4][5] neighborhood topology between particles of swarm is considered. In this model the behavior of each particle is affected by the best solution that is found by that particle and the best solution found by the neighboring particles. In this model the best local particle that is selected among topologic neighbors of each particle is replaced by best global particle in standard PSO. In this fashion each particle has its own global best particle. This consideration slows down information flow between particles. This means that a good solution found by a particle needs several generations to be transmitted to other particles while in standard PSO it takes only one generation to happen. This modification prevents crowding of all particles around local optima as it is found by a particle and thus trapping in local optima.

In this paper the Learning Automata based PSO reported in [3] is combined with the neighborhood topology concept and a new discrete PSO based on Cellular Learning Automata is introduced. In the proposed algorithm each cell of CLA contains a particle. Like [3] the number of learning automata assigned to each particle is the dimension of problem in binary representation. Rewards and punishments of learning automata for each particle are determined based on its best location and the best locations of its topologic neighbors. To show the effectiveness of the proposed algorithm we test the algorithm on several function optimization problems. The results of computer simulations show that the proposed algorithm attains better solutions in a faster way for most of the problems.

The rest of the paper is organized as follows. Particle swarm optimization is briefly given in section 2. Section 3 briefly presents Cellular Automata, Learning Automata and Cellular Learning Automata. Section 4 presents the proposed algorithm. Simulation results are given in section 5. Paper is concluded in section 6.

## 2. Particle Swarm Optimization

The particle swarm optimization simulates the behaviors of bird flocking. Suppose the following scenario; a group of birds are randomly searching food in an area. There is only one piece of food in the area being searched. All the birds do not know where the food is. The effective solution to find food is to follow the bird, which is nearest to the food. PSO learn from this scenario and use it to solve the optimization problems [1]. In PSO, each single solution is a bird in the search space that is called particle. All of the particles have fitness values, which are evaluated by a fitness function to be optimized. Particles have velocities, which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles (solutions) and then searches for optimal solutions by updating generations. In each iteration, the velocity and the position of each particle $i$ is updated using two quantities: the Personal Best solution obtained by particle $i$ ($PB_i$) and the Global Best solution ($GB$) obtained by the group of particles. After finding these two quantities, particle $i$ updates its velocity and its position according to the following equations.

$$v_{ij} = v_{ij} + rnd \times c_1 (PB_{ij} - x_{ij}) + rnd \times c_2 (GB_j - x_{ij}) \quad (1)$$
$$x_{ij} = x_{ij} + v_{ij} \quad (2)$$

where $v_i=(v_{i1}...v_{in})$ is the velocity of particle $i$, $x_i=(x_{i1},..., x_{in})$ is the current position (solution). $rnd$ is a random number in the range (0,1). $c_1$ and $c_2$ are learning factors. Usually $c_1$ is equal to $c_2$. Particles velocities on each dimension are limited to a maximum velocity of $V_{\max}$. If the sum of the accelerations causes the velocity on that dimension to exceed $V_{max}$, a parameter specified by the user, then the velocity on that dimension is set to $V_{max}$.

In past several years, PSO has been successfully applied to many research and application areas such as minimax problems [21], binary constraint satisfaction problem [22], constrained nonlinear optimization problems [23], improvised music [24], optimization in electromagnetic [25], and multimachine power system stabilizer design [26] to mention a few. It is demonstrated that PSO for optimization gets better results in a faster, cheaper way compared to other methods such as genetic algorithm. Another reason that PSO is attractive is that it has fewer parameters to adjust comparing to other methods of optimization [27][28].

The first version of particle swarm algorithm reported by Kennedy in [1] operates in continuous search space. But many optimization problems are set in discrete space. For this reason in [2], Kennedy proposed a discrete binary version of the particle swarm (DPSO). DPSO is obtained by modify equations in continuous PSO to be adapted to discrete binary space. In discrete binary space the search space can be viewed as a hypercube. Viewing the search space as a hypercube, the meanings of the concepts such as trajectory, velocity between and beyond used in continuous version of PSO will be changed. A particle may be seen to move nearer or farther from corners of hypercube by flipping various numbers of bits; in this way, velocity of the particles can be described by the number of bits changed per iteration. A particle with zeros bits changed does not move. A particle moves the farthest if all of its binary coordinates are changed. In this DPSO a dilemma occurs. What is the velocity or rate of change of a single bit or coordinate. Kennedy and Eberhart solved this dilemma by defining velocities and trajectories in term of changes in the probabilities that a bit will be 0 or 1.

For binary discrete search spaces, DPSO updates the velocity according to equation (1) but computes the new position component to be 1 with a probability which is obtained by applying a sigmoid transformation *(1/(1+exp (-v))* to the velocity component. The pseudo code for DPSO is given in figure 1.

```
Repeat
  For each particle i
    Calculate the fitness value
    If the fitness value is better than the best fitness value LB_i
        Set current value as the new LB_i
    End if
    Choose the particle with the best fitness value and call it GB
  End for
  For each particle i
    Calculate the particle velocity according to equation (1)
    Update the particle position as follows,
        If rnd<(1/(1+exp (-v_ij))
            x_ij=1
        Else
            x_ij=0
        End if
  End for
Until maximum iterations or minimum error criteria is attained
```

**Figure 1: Psudo code of CLA-PSO and VCLA-PSO**

## 3. Cellular Automata, Learning Automata and Cellular Learning Automata

This section briefly describes Cellular Automata, Learning Automata and Cellular Learning Automata.

*Learning Automata*: Learning Automata are adaptive decision-making devices operating on unknown random environments. The Learning Automaton has a finite set of actions and each action has a certain probability (unknown for the automaton) of getting rewarded by the environment of the automaton. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to result in selection of the optimal action. **Error! Reference source not found.** illustrates how a stochastic automaton works in feedback connection with a random environment. Learning Automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA). In the following, the variable structure learning automata is described.

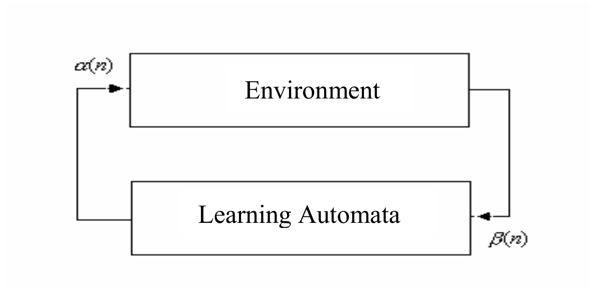



**Figure 2: The interaction between learning automata and environment**

A VSLA is a quintuple $<\alpha,\beta,p,T(\alpha,\beta,p)>$, where $\alpha$, $\beta$, $p$ are an action set with $s$ actions, an environment response set and the probability set $p$ containing $s$ probabilities, each being the probability of performing every action in the current internal automaton state, respectively. The function of $T$ is the reinforcement algorithm, which modifies the action probability vector $p$ with respect to the performed action and received response. Let a VSLA operate in an environment with $\beta=\{0,1\}$. A general linear schema for updating action probabilities can be represented as follows. Let action $i$ be performed.

$$\begin{array}{l} \text{If } \beta(t)=0, \\ \quad p_i(t+1) = p_i(t) + a[1 - p_i(t)] \\ \quad p_{j\neq i}(t+1) = (1-a)p_j(t) \\ \text{If } \beta(t)=1, \\ \quad p_i(t+1) = (1-b)p_i(t) \\ \quad p_{j\neq i}(t+1) = (b/s-1) + (1-b)p_j(t) \end{array} \tag{3}$$

Where $a$ and $b$ are reward and penalty parameters. When $a=b$, automaton is called $L_{RP}$. If $b=0$ and $0<b<<a<1$, the automaton is called $L_{RI}$ and $L_{R\varepsilon P}$, respectively. The overall operation of Learning Automaton is summarized in the algorithm of figure 3. For more information about learning automata the reader may refer to [9].

```
Initialize p to [1/s,1/s,…,1/s] where s is the number of actions
While not done
   Select an action i based on the probability vector p
   Evaluate the action and return a reinforcement signal β
   Update the probability vector using the learning algorithm
End While
```

**Figure 3: Psudocode of variable-structure learning automaton**

*Cellular Automaton*: Cellular Automata consists of a regular array of cells, each in one of finite number of states. The array can be in any finite number of dimensions. The state of a cell at time $t+1$ in CA is a function of the states of a finite number of cells (called its neighborhood) at time $t$. The simplest CA would be one-dimensional, with two possible states per cell, and a cell's neighbors defined to be the adjacent cells on either side of it. For more information about cellular learning automata the reader may refer to [13].

*Cellular Learning Automata* (CLA)**:** CLA which is obtained by combining cellular automata and learning automata is a mathematical model for dynamical complex systems that consists of a large number of simple learning components. Any number of LA can reside in a specific cell. Reinforcement signal for every automaton is computed according to CLA rule and actions of other LA residing in neighbor cells. This model has learning capability of LA and collective behavior and locality of CA. it is formally described in [6] as follows: $d$-dimensional CLA is a quintuple $CLA = (Z^d, \phi, A, N, F)$ that

- $Z^d$ is a $d$-dimensional grid of cells. This grid can be finite, semi-finite or infinite.
- $\phi$ is a finite set of states that each cell can have.

- A is set of learning automata that each of them are assigned to a specific cell.
- $N = \{x_1, ..., x_m\}$ is finite subset of $Z^d$ that is called neighborhood vector.
- $F : \phi^m \rightarrow \underline{\beta}$ is local rule of CLA. $\underline{\beta}$ is set of valid reinforcement signals that can be applied to LA.

The operation of cellular learning automata could be described as follows: At the first step, the internal state of every cell specified. The state of every cell is determined on the basis of action probability vectors of learning automata residing in that cell. The initial value may be chosen on the basis of experience or at random. In the second step, the rule of cellular automata determines the reinforcement signal to each learning automaton residing in that cell. Finally, each learning automaton updates its action probability vector on the basis of supplied reinforcement signal and the chosen action. This process continues until the desired result is obtained. For more information about cellular learning automata the reader may refer to [6] [14] [15] [16] [17].

## 4. A New Discrete PSO based on Cellular Learning Automata

This section we first briefly describe LA-PSO and then present the proposed algorithm.

**Discrete PSO based on learning automata:** In Discrete PSO based on learning automata (LA-PSO) reported in [3] every particle has a set of learning automata that determine the location of particle in the search space. In LA-PSO, learning automata are used by the particles to model the dynamics of the group of which the particle is a member. The set of leaning automata associated to a particle by observing the behavior of the group leads the particle to search the optimal geographical with a well preserved population density distribution. Similar to the DPSO, the velocities and trajectories concepts are defined in terms of changes of probabilities that a bit will become 0 or 1. Instead of using equation (1) and the sigmoid transformation, learning automata are used to determine the position of the particles. In other words, the set of learning automata associated to a particle can be viewed as the components of the brain of the particle. These components collectively lead the particle to search the place where the food can be found with high probability. The number of learning automata assigned to each particle is equal to the dimension of the search space. Each automaton has two actions 0 and 1. The LA based DPSO algorithm works as follows:

While terminal condition is not reached, iterate the following steps,

**1**- All the particles do steps I and II simultaneously,

I -every automaton associated to particle *i* chooses one of their actions (0 or 1) according to their action probability vectors.

II- the particle *i* generates a new position (a corner of the hypercube) by (concatenating) combining the actions chosen by its set of learning automata. The particle then moves to that position. If the fitness value of new position is better than the best fitness value $LB_i$, $LB_i$ will be set to the new position.

**2**- Position of the particle with the best fitness, *GB,* is computed.

**3**-All the particles perform the following simultaneously

- Based on *GB*, $LB_i$ and the position of the particle, each particle *i* generates a reinforcement vector $\beta_i=(\beta_{i1}, ..., \beta_{in})$ which becomes the input to the set of learning automata associated to particle *i*.

The *j*th element of the reinforcement vector for particle *i*, $\beta_{ij}$, *is* computed as follows,

$$\beta_{ij} = \begin{cases} 0 & if \quad LB_{ij} = GB_{ij} = x_{ij} \\ 1 & otherwise \end{cases} \tag{4}$$

The reinforcement vector $\beta_i$ will be used to update the action probabilities vectors of the set of learning automata associated to particle *i*.

The set of learning automata associated to a particle helps the particle to gradually move to a position that confirms with the global goal of particles..

**The proposed Algorithm: A cellular learning automata based Discrete PSO:** In this section a new discrete PSO based on cellular learning automata is proposed. The proposed model is the combination of LA-PSO described before with cellular learning automata. In the proposed method every particle is assigned to one cell of a CLA with linear topology. Each cell of CLA is equipped with a set of learning automata. The set of learning automata in each cell (particle) determines the location (bits) of that particle in the search space. Figure 4 shows the placement of learning automata in cells (particles) of a CLA with linear topology. Each particle in this configuration has some neighboring particle residing in the neighboring cells. If the neighborhood radius is *r* then a particle has *2r+1*particles as its neighbors, r particles to the left, *r* particle to the right and itself. Figure  shows neighborhood of size 1 for cell *i*.

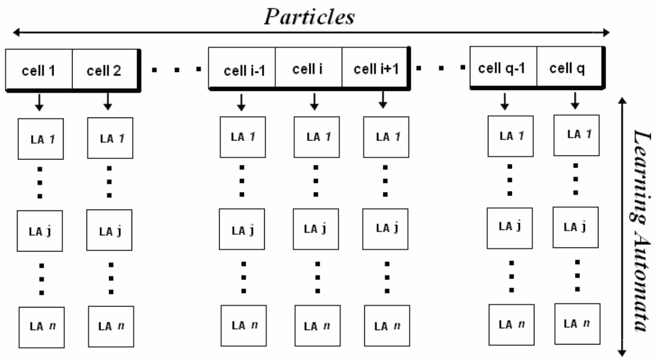



**Figure 4: placement of LA in one dimensional CLA**

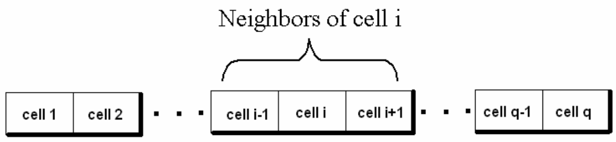



**Figure 5: Neighborhood of size 1 for cell *I* in linear CLA.**

Having CLA instead of LA in LA-PSO, the concept of neighborhood can be taken into consideration. In CLA-PSO each particle in a cell  finds its local best (*LB*) by comparing the local bests of its neighbors and  the global best (*GB*) paprmeter no longer exists. Each learning automaton *j* residing in cell *i* compares its action ($X_{ij}$) with the corresponding bits in its personal best ($PB_{ij}$) and local bests ($LB_{ij}$) *of* its neighbors. If these three bits are equal then reward signal is generated for that automaton ($\beta ij=0$) otherwise punishment signal is generated ($\beta ij=1$).

Two types of fixed and variable neighborhood are considered. In fixed neighborhood, radius of neighborhood (*r*) is set to be a fraction of the number of cells in CLA whereas in variable neighborhood, the radius of neighborhood increases from one to half of the array size in such a way that neighbors of a cell in the last generations is the set of all cells. That is, in the last generation the proposed algorithm operates exactly the same as LA-PSO. We call the proposed algorithm CLA-PSO when the neighborhood is fixed and VCLA-PSO when the algorithm it uses variable neighborhood.

We also introduce a new parameter called $P_{max}$(>0.5). This parameter forces the probability of the value of a bit to be 1 or 0 in each learning automaton lie in the range [1-$P_{max}$ , $P_{max}$]. This pramater prevents premature convergence of  LA-PSO that may happen because of early convergence of learning automata. It can also provide CLA-PSO with higher explorative ability in the late generations when learning automata are about to converge. The role of this parameter is similar to the role of mutation parobability in genetic algorithm. Note that in contrast to mutation probability in genetic algorithm, lower values of this parameter results in higher  exploration strength.

In order to use CLA-PSO for an optimization (maximization for example) problem, first a set of learning automata is associated to each particle of CLA-PSO. The number of learning automata associated to a cell (and the corresponding particle) is the number of bits in the binary representation of points in the search space. Each automaton has two actions 0 and 1. The learning automaton, corresponding to the *j*th bit of the solution in cell *i* selects 1 with probability $P_{ij}$ and 0 with probability 1-$P_{ij}$. Following steps will be repeated until a termination criterion is met.

1- Every automata in a cell *i* chooses one of its actions using its action probability vector

2- Particle $i$ generates a new position $X_i$ (a corner of the hypercube) by concatenating the actions chosen by its set of learning automata. If the fitness of new position is better than that stored in $PB_i$ , $PB_i$ will be updated.
3- Each particle computes its local best ($LB_i$) according to personal bests ($PB$) of its neighbors

$$LB_i = \{PB_j \mid j = \arg\max_{k \in N_i}(fitness(PB_k))\} \quad (5)$$

Where $N_i$ is set of topologic neighbors of cell $i$ in CLA array. (*argmax* is used because it is assumed that we are solving a maximization problem).
4- According to $PB_i$ and $LB_i$ and $X_i$ reinforcement signal $\beta_i=(\beta_{i1} ..., \beta_{in})$ for learning automata of cell $i$ is generated. If $PB_{ij}$ and $LB_{ij}$ and $X_{ij}$ are equal then the $j$th learning automaton in cell $i$ gets reward ($\beta_{ij}$ =0) otherwise it gets punishment($\beta_{ij}$ =1).
5- Reinforcement signal $\beta_i$ is used to update action probabilities of learning automata in particle $i$.
6- If $P_{ij}$ is not in the range of [1-$P_{Max}$ , $P_{Max}$] then.

$$P_{ij} = \min(\max(P_{ij}, 1 - P_{\max}), P_{\max}) \quad (6)$$

The overall operation of CLA-PSO and VCLA-PSO is summarized in the algorithm of figure 6.

```
While not done do
   For each particle i in CLA do
      Generate a new corner of hypercube and go to that location
      Evaluate the new corner of hypercube
      If fitness(new location)> fitness(PBi) then
         Update PBi
      End if
      Select LBi cells from neighbors of cell i
      Generate the reinforcement signal vector
      Update LAs of particle i
      Correct out of bound probabilities of LA in particle i
   End for
End while
```

**Figure 6: Psudo code of CLA-PSO and VCLA-PSO**

## 5. Simulation Results

In order to show the performance of the proposed algorithms five optimization problems are solved with CLA-PSO and VCLA-PSO and the results are compared with the results obtained with the results for discrete PSO (DPSO) and LA-PSO[3]. For each test problem population size is varied from 10 to 100 with step size of 5. The results are the average over 20 runs. Problems are two De Jong functions, Ackley function, Checker Board and Simple knapsack 0/1 problems. The test problems are briefly explained below:

**De Jong Functions:** De Jong functions used are borrowed from [7]. We call these two functions De Jong F1 and F2. In the experiments we have used 10 dimensional De Jong Functions:

$$F_1(X) = \sum_{i=1}^{n} x_i^2 \qquad -5.12 < x_i < 5.12$$

$$F_2(X) = \sum_{i=1}^{n-1} 100\,(x_i^2 - x_{i+1})^2 + (1 - x_i)^2 \qquad -2.048 < x_i < 2.048$$

**Ackley Function:** This function has many local optimums near and away from global optimum. Highly dimensional of this problem is very hard to optimize. In the experiments we have used 10 dimensional Ackley Function.

$$f_{ackley}(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \qquad -10 < x_i < 10$$

**CheckerBoard Problem**: This problem was used by Baluja to evaluate the performance of the PBIL algorithm [10]. In this problem a $s \times s$ grid is given. We used $s$=10 for our experiments. Each point of the grid can take two values 1 and 0. The goal of this problem is to create a checkerboard pattern of 0's and 1's on the grid. Each location with a value of 1 should be surrounded in all four directions by a value 0, and vise versa. Only the four primary directions are considered in the evaluation. The evaluation is measured by counting the number of correct surrounding bits for the present value of each bit position for a $(s-2)\times(s-2)$ grid. In this manner, the corners are not included in the evaluation. The chromosomes of this problem have dimension $n=s^2$. If we consider the grid as a matrix $C=[c_{ij}]$ $(i,j=1,\ldots,s)$ and interpret $\delta(a,b)$ as the kronecher's delta function, the checkerboard function can be written as follows,

$$F_{Checker\ Board}(C) = 4(s-2)^2 - \sum_{i=2}^{n-1}\sum_{j=2}^{n-1} \delta(c_{ij},c_{i-1\,j}) + \delta(c_{ij},c_{i+1\,j}) + \delta(c_{ij},c_{i\,j-1}) + \delta(c_{ij},c_{i\,j+1})$$

**Knapsack 0/1 Problem**: In the simple knapsack problem, there is a single bin of limited capacity *C*, and *n* elements of varying size. The problem is to select the elements that will yield the greatest usage bin capacity without exceeding it. If a solution selects too many elements, such that sum of elements' sizes become larger than bin capacity, some items are randomly removed from the bin until sum of elements' sizes be come smaller than *C*.

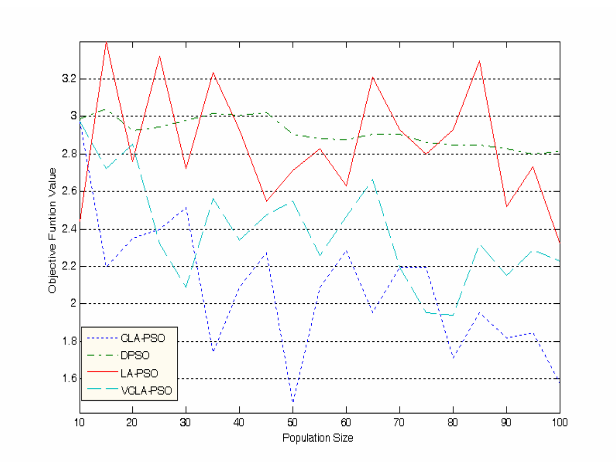
$$f_{Knapsack}(X) = C - \sum_{i=1}^{n} x_i$$

Number of elements are 100 and the sizes of them are selected uniformly between intervals of [0,100]. *C* is set to 0.95 of sum of elements' sizes. Answers for this problem are binary strings of size 100. Each 1 in string indicates presence of corresponding element in bin. Details of test problems are given in Table 1.

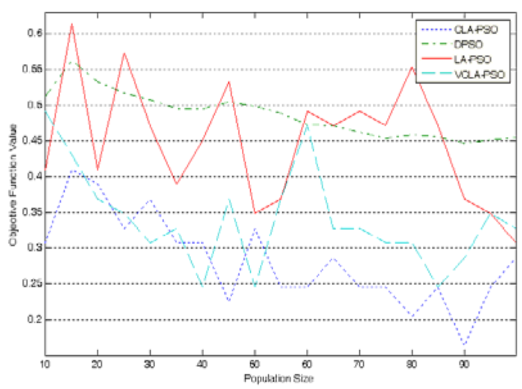
**Table 1 details of test problems**

| **Problem** | *Knapsack* | *Ackley* | $F_{chekcher\ board}$ | *De Jong* $F_2$ | *De Jong* $F_1$ |
|---|---|---|---|---|---|
| **Decimal Dimension** | - | 10 | - | 10 | 10 |
| **Binary Dimension** | 100 | 50 | 100 | 50 | 50 |
| **Type** | Min | Min | Max | Min | Min |
| **Optimal Value** | 0 | 0 | 256 | 0 | 0 |
| **Number of Generation** | 100 | 500 | 500 | 500 | 500 |

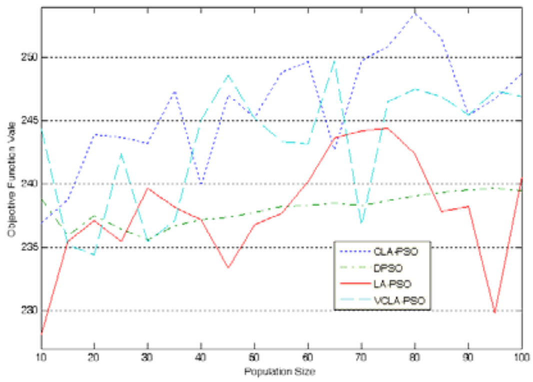
In CLA-PSO radius of neighborhood is set to 1/20 of CLA array size. In VCLA-PSO radius of neighborhood grows linearly form 1 to half of CLA array size as generation number increases. $P_{max}$ in both CLA-PSO and VCLA-PSO is set to 0.95. Learning automata in all experiments use $L_{RP}$ ($a$=$b$=0.1) learning scheme. DPSO is run with parameters reported in [2]. Mean of best answers in the last generation of algorithms over 20 runs for different population sizes are reported in Figure .



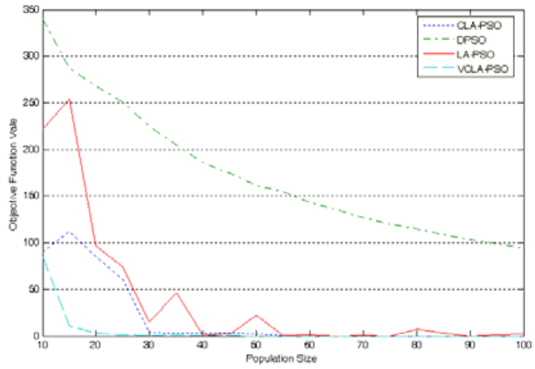

a) Ackley function



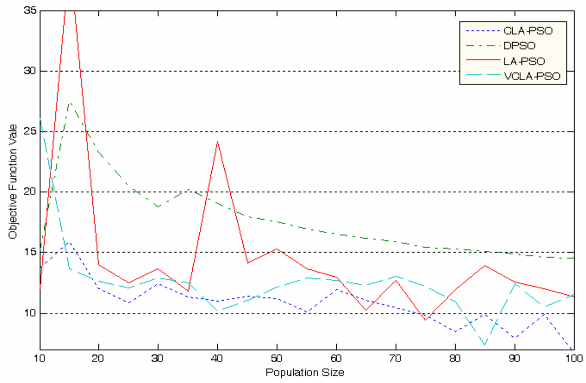

b) De Jong F1




c) Checker Board Problem




d) Knapsack Problem




e) De Jong F2

**Figure 7: Comparison of four discrete PSO methods: CLA-PSO, VCLA-PSO, LA-PSO, DPSO**

Figure  shows that CLA-PSO outperforms other methods on Knapsack, Checker Board ، De Jong F1 and Ackley problems. VCLA-PSO performance is between CLA-PSO and LA-PSO in these problems while it produces better solutions on Knapsack problem compared to CLA-PSO. Slight improvement by (CLA-,VCLA-)PSO can be seen in De Jong F2. DPSO shows poor performance in function optimization compared to other discussed methods specially CLA-PSO

## 6. Conclusions

In this paper in a new LA based Discreet PSO was proposed. The proposed algorithm is a combination of the only reported learning automata based DPSO (LA-PSO) algorithm and cellular learning automata.  Experimental results on five optimization problems show the superiority of the proposed algorithm.

## References


[1] Kennedy, J., and Eberhart, R. C., “Particle Swarm Optimization”, Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, , pp. 1942–1948, 1995.

[2] Kennedy, J., and Eberhart, R. C., “A Discrete Binary Version of The Particle Swarm Algorithm”, Proceedings of Conference on Systems, Man, and Cybernetics, pp. 4104-4108, IEEE Service Center, Piscataway, NJ, 1997.

[3] Rastegar, R., Meybodi, M. R. and Badie, K., “A New Discrete Binary Particle Swarm Optimization based on Learning Automata”, Proceedings of International Conference on Machine Learning and Applications (ICMLA2004), pp.456-462, USA, IEEE Press, 2004.

[4] Kennedy, J. and Mendes, R., “Population Structure and Particle Swarm Performance”, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii USA. 2002.

[5] Suganthan, P. N., "Particle Swarm Optimizer with Neighborhood Operator", Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), Piscataway, NJ, pp. 1958-1962, 1999

[6] Meybodi, M. R., Beigy, H., and Taherkhani, M., "Cellular Learning Automata", Proceedings of 6th Annual International Computer Society of Iran Computer Conference CSICC2001, Isfahan, Iran, pp. 153-163, 2001.

[7] De Jong, K. A., "The Analysis of the behavior of a class of genetic adaptive systems" Ph.D. dissertation, University of Michigan, Ann Arbor, 1975

[8] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, New York, 1989.

[9] Narendra, K. S., and Thathachar, M. A. L., *Learning Automata: An Introduction*, Prentice-Hall Inc, 1989

[10] Baluja, S., Caruana, R., "Removing The Genetics from The Standard Genetic Algorithm", Proceedings of ICML'95, PP. 38–46, Morgan Kaufmann Publishers, Palo Alto, CA, 1995.

[11] Sheybani, M., Meybodi, M. R., "PSO-LA: A New Model for Optimization", Proceedings of 12th Annual International Computer Society of Iran Computer Conference CSICC2007, Iran, pp. 1162-1169, 2007.

[12] Sheybani, M. and Meybodi, M. R., "CLA-PSO: A New Model for Optimization", Proceedings of 15th Conference on Electrical Engineering (15$^{th}$ ICEE), Volume on Computer, Telecommunication Research Center, Tehran, Iran, May 15-17, 2007.

[13] Wolfram, S., *Cellular Automata and Complexity*, Perseus Books Group, 1994.

[14] Beigy, H. and Meybodi, M. R., "A Mathematical Framework for Cellular Learning Automata", Advances on Complex Systems, Vol. 7, Nos. 3-4, pp. 295-320, September/December 2004.

[15] Rastegar, R., Meybodi, M. R. and Hariri, A., "A New Fine Grained Evolutionary Algorithm based on Cellular Learning Automata", International Journal of Hybrid Intelligent Systems, IOS Press, Volume 3, Number 2, pp. 83-98, 2006.

[16] Hariri, A., Rastegar, R., Zamani, M. S. and Meybodi, M. R., "Parallel Hardware Implementation of Cellular Learning Automata based Evolutionary Computing on FPGA", Proceedings of 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05), pp. 311-314, California, USA, April 17-20, 2005.

[17] Masoodi, B., Meybodi, M. R. and Hashemi M., "Cooperative CLA-EC", Proceedings of 12th Annual CSI Computer Conference of Iran, Shahid Beheshti University, Tehran, Iran, pp. 558-559, Feb. 20-22, 2007.

[18] Shi, Y. and Eberhart, R. C., "A Modified Particle Swarm Optimizer", IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, USA, 1998.

[19] Wang, X. H. and Li, J. J., "Hybrid Particle Swarm Optimization with Simulated Annealing", Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, China, 2004.

[20] Gary, G. Y., and Haiming, Lu., "Dynamical Population Strategy assisted Particle Swarm Optimization", Proceedings of the 2003 IEEE International Symposium on Intelligent Control, Houston, Texas, pp. 697-702, October 2003.

[21] Laskari, E. C., Parsopoulos, K. E., and Vrahatis, M. N., "Particle Swarm Optimization for Minimax Problems", Proceedings of the IEEE Congress on Evolutionary computation 2002 Honolulu, Hawaii USA, 2002b.

[22] Schoofs, L., and Naudts, B., "Swarm Intelligence on the Binary Constraint Satisfaction problem", Proceedings of the IEEE Congress on Evolutionary Computation 2002 Honolulu, Hawaii USA, 2002.

[23] Hu, X., and Eberhart, R. C., "Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization", Proceedings of the Sixth World Multiconference on Systemic, Cybernetics and Informatics 2002 (SCI 2002), Orlando, USA, 2002.

[24] Blackwell, T. and Bentley, P. J., "Improvised Music with Swarms", Proceedings of the IEEE Congress on Evolutionary Computation 2002 Honolulu, Hawaii USA, 2002b.

[25] Ciuprina, G., Ioan, D., and Munteanu, I., "Use of Intelligent Particle Swarms Optimization in Electromagnetics", IEEE Transactions on Magnetic, Vol. 38, No. 2, PP. 1037-1040. 2002.

[26] Abido, M. A., "Particle Swarm Optimization for Multimachine Power System Stabilizer Design", Power Engineering Society Summer Meeting, Vol. 3, PP. 1346-1359, 2001.

[27] Kennedy, J., "The Particle Swarm: Social Adaptation of Knowledge", Proceedings of IEEE International Conference on Neural Networks (Indianapolis, Indiana), pp. 303-308, IEEE Service Center, Piscataway, NJ, 1997.

[28] Clerc, M., and Kennedy, J.,"The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space", IEEE Transaction on Evolutionary Computation, Vol. 6, No. 1, PP. 58-73, February 2002.